Breaking the Draft-Verify Barrier: Heterogeneous Speculative Decoding on AI PCs

Anonymous ACL submission

Abstract

Emerging AI PCs integrate NPUs, iGPUs, and CPUs under a unified memory architecture, yet existing LLM inference engines fail to exploit their heterogeneous concurrency because speculative decoding's draft–verify workflow is inherently sequential. We present **SD-HC**, a heterogeneous functional pipelining framework that re-architects speculative decoding across NPU, iGPU, and CPU execution lanes. SD-HC overlaps batch-N verification with batch-$(N{+}1)$ drafting, introduces asymmetric prefilling to reduce time-to-first-token (TTFT), and employs an $O(1)$ surgical KV-cache rollback mechanism to minimize rejection overhead. For sparse mixture-of-experts (MoE) models, SD-HC further masks expert-loading latency via CPU-assisted gating overlap. Experiments on AMD Ryzen AI platforms show that SD-HC achieves up to **1.74× throughput improvement** and **44% lower TTFT** over single-device and naive heterogeneous baselines, enabling practical 10B-scale LLM inference on consumer AI PCs.

1 Introduction

Large language models (LLMs) now underpin a wide range of applications, from natural language understanding (Devlin et al., 2019) to open-ended text generation (Radford et al., 2019). While these models deliver strong performance, their ever-growing parameter counts impose prohibitive memory and compute requirements that strain even cloud-grade GPU servers.

Heterogeneous computing platforms offer a promising path forward by distributing workloads across specialized accelerators. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) is particularly well suited to this setting: a lightweight draft model proposes candidate tokens on one device while a larger target model verifies them on another. However, most existing heterogeneous speculative decoding work targets data-center GPU clusters (Miao et al., 2024; Spector and Re, 2023) and does not address the unique constraints of edge devices with unified memory architectures.

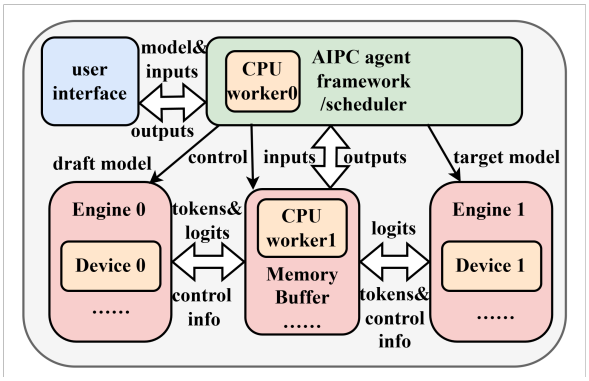

Figure 1: Overview of SD-HC on a heterogeneous computing platform. The framework schedules tasks by deploying the draft model on the NPU (Device 0), the target model on the GPU (Device 1), and the verification logic on the CPU, taking full advantage of each processor's complementary strengths.

AI Personal Computers (AI PCs) represent an emerging class of edge devices that integrate a CPU, an integrated GPU (iGPU), and a neural processing unit (NPU) under a unified memory architecture (Figure 1). This heterogeneous design enables on-device inference that reduces cloud dependency and preserves data privacy. AMD's AIPC ecosystem, built around the XDNA NPU, exemplifies this trend. Deploying LLMs on such platforms, however, introduces distinct challenges: limited iGPU memory constrains context lengths (Kwon et al., 2023; Child et al., 2019), and low-precision execution entails a fundamental trade-off between throughput (INT4) and accuracy (BF16).

In this paper, we present **SD-HC** (Speculative Decoding – Heterogeneous Computing), a system designed for efficient LLM inference on resource-constrained AI PC devices. Our key contributions are:

- We identify the **sequential verification barrier** in speculative decoding as the principal obstacle to effective utilization of heterogeneous AI PC accelerators, and propose SD-HC, a functional pipelining framework that overlaps drafting, verification, and prefilling across NPU, iGPU, and CPU.

- We design a **dual-pipeline architecture with asymmetric prefilling** that runs the draft model on the NPU and the target model on the iGPU in parallel, together with a sliding-window KV-cache management strategy that bounds memory at $O(W)$ while reducing TTFT by 44%.

- We introduce **lightweight $O(1)$ KV-cache rollback** and **CPU-assisted MoE expert scheduling** mechanisms that preserve speculative decoding efficiency under cross-device execution, incurring only 3.1% latency overhead per recovery while sustaining 78% token acceptance with INT4-quantized draft models within 1.4 GB of memory.

Compared to GPU-only inference, SD-HC achieves 1.50×–1.74× end-to-end speedup and reduces median time-to-first-token by 44%, while preserving lossless output accuracy relative to the base quantized model.

2 Background and Related Work

2.1 AIPC Platform

Modern AIPC platforms combine CPU, GPU, and NPU in a unified memory architecture (UMA), where memory bandwidth is shared across processors (e.g., 50 GB/s to CPU/NPU and 100 GB/s to iGPU over LPDDR5x-7500). Table 1 summarizes the peak compute of representative NPU and GPU configurations. The NPU excels at sustained low-power integer inference, while the GPU provides higher throughput for batch operations. This complementarity makes them natural candidates for a draft–verify split.

Deploying LLMs on these platforms presents three core challenges. First, *resource constraints*: local devices cannot sustain dense models with billions of parameters without aggressive compression. Second, *precision–performance trade-offs*: BF16 preserves quality but is slower, while INT4 is fast but degrades accuracy for sensitive domains. Third, *privacy and cost*: public cloud LLM APIs expose sensitive prompts and retrieval-augmented generation (RAG) corpora (Li et al., 2024; Gao et al., 2024), and private GPU clouds with A100 hardware cost upward of $28,499 per node, compared to roughly $800 for an AMD Ryzen AI 9 HX 370 AI PC.

Table 1: Peak compute comparison between AMD NPU and mobile GPUs (Advanced Micro Devices, Inc., 2024b; NVIDIA Corporation, 2024). NPUs offer energy-efficient INT8 inference; mobile GPUs provide substantially higher TOPS for batch verification.

Device Type	Performance	Memory Type
AMD XDNA	32 TOPS	LPDDR5x-7500
AMD Strix	126 TOPS	256-bit LPDDR5x
RTX 4060 Mobile	233 TOPS	GDDR6
RTX 4070 Mobile	321 TOPS	GDDR6
RTX 4080 Mobile	542 TOPS	GDDR6

2.2 Related Work

Speculative Decoding Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) accelerates autoregressive generation by having a small draft model propose tokens that a larger target model verifies in parallel. Stern et al. (Stern et al., 2018) introduced blockwise parallel decoding; Leviathan et al. (Leviathan et al., 2023) formalized the rejection-sampling acceptance criterion that guarantees output distribution equivalence. Subsequent works—BiLD (Kim et al., 2023), Medusa (Cai et al., 2024), and draft-and-verify (Zhang et al., 2023)—explored alternative verification strategies. PEARL (Havaldar et al., 2024) introduced adaptive draft length to dynamically tune speculation depth. Methods like Lookahead Decoding (Fu et al., 2024) and Ouroboros (Zhao et al., 2024) reduce draft-step overhead but retain sequential draft–verify execution. SpecInfer (Miao et al., 2024) and Spector (Spector and Re, 2023) extend speculative decoding to heterogeneous servers but do not target AIPC-class devices with unified memory.

Heterogeneous and Parallel Inference Data parallelism (Li et al., 2020) replicates models across devices; tensor parallelism (Megatron-LM (Shoeybi et al., 2019)) partitions individual layers; pipeline parallelism (Artetxe et al., 2021) stages model components across devices. DeepSpeed (Aminabadi et al., 2022) and ZeRO (Rajbhandari et al., 2020) focus on GPU clusters. In con-

Algorithm 1: Naive Speculative Sampling (baseline)

Input : prefix $\mathbf{x}_{<t}$, draft model p_{draft}, target model p_{target}
Output : generated sequence $\mathbf{y}$
for $t \leftarrow 1$ **to** T **do**
  $y_t^{\text{draft}} \sim p_{\text{draft}}(\cdot \mid \mathbf{x}_{<t})$ `// draft token`
  $\mathbf{l}_t \leftarrow f_{\text{target}}(\mathbf{x}_{<t}, y_t^{\text{draft}})$
  $p_t^{\text{target}} \leftarrow \text{softmax}(\mathbf{l}_t)[y_t^{\text{draft}}]$
  $r_t \leftarrow \min\big(1,\, p_t^{\text{target}} / p_t^{\text{draft}}\big)$
  $u \sim \text{Uniform}(0, 1)$
  if $u \le r_t$ **then**
    $y_t \leftarrow y_t^{\text{draft}}$ `// accept`
  else
    $p^{\text{corr}} \leftarrow \text{normalize}\big(\max(0,\, p_{\text{target}} - r_t p_{\text{draft}})\big)$
    $y_t \sim p^{\text{corr}}$ `// resample`
  end
  $\mathbf{x}_{<t+1} \leftarrow \mathbf{x}_{<t} \oplus y_t$
end

trast, SD-HC employs functional decomposition—assigning draft generation to NPU and target verification to GPU—avoiding inter-layer communication overhead and exploiting the natural structure of speculative decoding.

Model Compression Methods Quantization (GPT.int8 (Dettmers et al., 2022), GPTQ (Frantar et al., 2022), AWQ (Lin et al., 2024), SmoothQuant (Xiao et al., 2023)) and pruning (Frantar and Alistarh, 2023) are standard techniques for edge deployment. Knowledge distillation (Hinton et al., 2015; Zhou et al., 2024) can recover accuracy but requires loading both teacher and student simultaneously. SD-HC complements these methods by using INT4/INT8 draft models that fit in NPU memory while the full-precision target remains on the GPU.

3 Methodology

SD-HC is designed around a single organizing principle: *every cycle of the heterogeneous platform should be productive*. We achieve this through three tightly coupled mechanisms: (1) a dual pipeline that overlaps draft generation and verification, (2) a lightweight rollback protocol that keeps error-recovery overhead minimal, and (3) a MoE-aware NPU execution strategy that further reduces draft latency. The following subsections describe how these components interact.

3.1 Dual-Pipeline Architecture and Prefilling

Before describing the pipeline mechanics, we briefly justify the choice of *functional splitting* over the two alternative parallelization strategies commonly used in distributed inference. *Tensor-parallel* splitting partitions each linear layer across devices, but the resulting all-reduce traffic dominates total latency on AI PCs, where the inter-device link is a shared LPDDR bus rather than NVLink-class fabric. *Layer-parallel* (pipeline) splitting cuts the model along its depth, which staggers verification of a single sequence across devices and therefore neither hides draft latency nor exploits the natural draft–verify decomposition. In contrast, functional splitting maps the two roles of speculative decoding—generation and verification—onto two devices that already differ in compute–memory profile: the NPU is optimized for low-power sustained integer inference and easily holds a compact draft model, while the iGPU's higher peak FLOPS suit batched verification of multiple candidate tokens. The only inter-device traffic is a short token / logit stream per speculation step, which is small relative to LPDDR5x bandwidth. This alignment between hardware asymmetry and algorithmic structure is the central reason SD-HC adopts functional splitting.

The core insight of SD-HC is that draft generation and target verification are naturally decoupled: one device can be speculating ahead while the other is verifying the previous batch. We formalize this by assigning the draft model p_d to Device 0 (NPU) and the target model p_t to Device 1 (GPU), with the CPU acting as arbiter.

Mathematical formulation Given prefix $\mathbf{x}_{<t}$, the NPU generates k candidate tokens:

$$P_d(\hat{x}_{t:t+k-1} \mid \mathbf{x}_{<t}) = \prod_{i=0}^{k-1} p_d(\hat{x}_{t+i} \mid \mathbf{x}_{<t+i}) \quad (1)$$

The GPU verifies each candidate via the modified rejection criterion:

$$r_i = \min\left(1,\, \frac{p_t(\hat{x}_{t+i} \mid \mathbf{x}_{<t+i})}{p_d(\hat{x}_{t+i} \mid \mathbf{x}_{<t+i})}\right) \quad (2)$$

If token j is rejected, the CPU samples from the normalized residual:

$$p_{\text{corr}}(x) = \frac{\max(0,\, p_t(x \mid \mathbf{x}_{<t+j}) - p_d(x \mid \mathbf{x}_{<t+j}))}{\sum_{x'} \max(0,\, p_t(x' \mid \mathbf{x}_{<t+j}) - p_d(x' \mid \mathbf{x}_{<t+j}))} \quad (3)$$

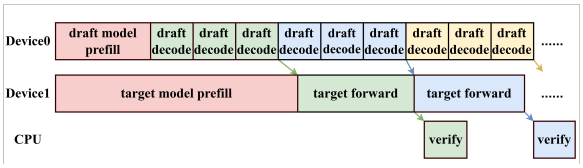

Figure 2: Normal (error-free) pipeline of SD-HC. The NPU (Process A) generates draft tokens continuously while the GPU (Process B) verifies in batch; the CPU (Process C) arbitrates acceptance and advances the shared context. Overlapping the two inference streams eliminates idle time on both accelerators.

guaranteeing that the final output follows p_t regardless of draft quality.

Asymmetric prefilling A key bottleneck is time-to-first-token (TTFT). In a naive setup, the NPU idles while the GPU prefills the entire prompt. SD-HC eliminates this waste by splitting the prefill across both devices. Let v_n and v_g denote the token-processing rates of the NPU and GPU, respectively. The optimal NPU allocation L_n^* minimizes:

$$T_{\text{pre}} = \max\left(\frac{L_n}{v_n}, \frac{L - L_n}{v_g} + \Delta_{\text{trans}}\right) \quad (4)$$

$$L_n^* \approx \frac{v_n(L - v_g\,\Delta_{\text{trans}})}{v_n + v_g} \quad (5)$$

where Δ_{trans} is the KV-cache synchronization overhead. This yields parallel prefilling that reduces median TTFT by 44% in our experiments.

Token-wise alignment Because the NPU and GPU run asynchronously, verification logits must be carefully aligned with draft tokens. As shown in Figure 3, the GPU processes the last accepted token concatenated with all draft tokens except the tail, ensuring a one-to-one correspondence between logit and draft positions. This alignment is transparent to the acceptance logic but essential for correctness in the asynchronous setting.

The complete dual-pipeline decoding procedure is given in Algorithm 2.

3.2 Error Handling and KV-Cache Management

When a draft token is rejected, naively re-executing from scratch would negate all pipeline benefits. SD-HC instead performs *surgical rollback*: the CPU signals both devices to stall, samples a corrected token from the residual distribution, and each device independently repairs only the affected KV-cache slots before resuming. Figure 4 illustrates the pipeline behavior during a rejection event.

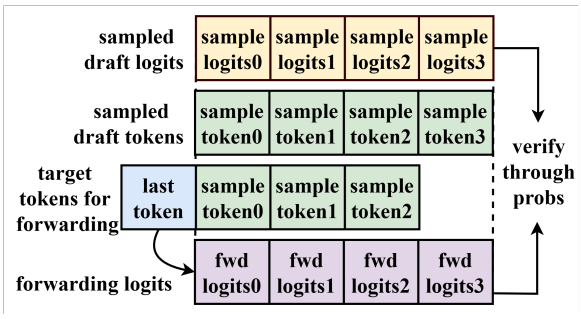

Figure 3: Token-wise alignment strategy for asynchronous verification in SD-HC. The GPU processes the last accepted token concatenated with the current draft batch (excluding the tail), ensuring that each draft token's logit is compared against the corresponding target logit for correct acceptance evaluation.

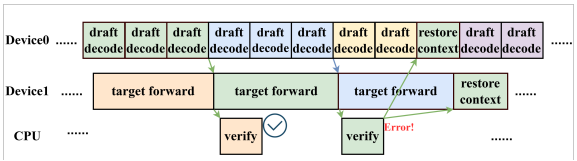

Figure 4: SD-HC pipeline behavior when a draft token is rejected. On rejection, the CPU broadcasts a stall signal; Device 0 (NPU) discards the erroneous suffix and re-samples from the corrected position, while Device 1 (GPU) updates its KV cache to include the corrected token as the new last token. The pipeline then resumes with no loss of context beyond the rejection point.

Sliding-window KV management SD-HC manages KV caches via a circular buffer of size W. The logical-to-physical mapping is $f(i) = i \bmod W$. Upon a speculative failure at index t_{fail}, the tail pointer rolls back:

$$T_{\text{new}} = \big(T_{\text{old}} - (k - n_{\text{accepted}})\big) \bmod W \quad (6)$$

For models with Sliding Window Attention (e.g., Qwen2.5 (Team, 2025)), evicted tokens are stored in a secondary dropping buffer B of capacity equal to the verification window. Context restoration follows:

$$\mathbf{K}_v[i] = \begin{cases} B[i], & i < L - W \\ \text{Recompute}, & \text{otherwise} \end{cases} \quad (7)$$

This preserves long-range dependencies at constant memory cost $O(W)$. Algorithm 3 details the rollback procedure.

3.3 MoE-Aware Draft Model on NPU

The preceding pipeline assumes that draft latency T_d is short relative to target latency T_t. To widen this gap, SD-HC supports Mixture-of-Experts

Algorithm 2: SD-HC: Heterogeneous Dual-Pipeline Decoding

Input : Prefix $\mathbf{x}_{<t}$, draft model M_d, target model M_t, lookahead k
Output : Generated sequence $\mathbf{x}_{1:T}$
Prefill: split $\mathbf{x}_{<t}$; run L_n^* tokens on NPU, remainder on GPU in parallel
while *not EOS and* $t < T$ **do**
  Process A (NPU)
  for $i = 0$ **to** $k-1$ **do**
    $\hat{y}_{t+i} \sim p_d(\cdot \mid \mathbf{x}_{<t+i})$ Push $(\hat{y}_{t+i}, prob_d)$ to queue
  end
  Process B (GPU)
  Wait for draft tokens $\mathbf{l}_{t:t+k} = f_t(\mathbf{x}_{<t}, \hat{y}_{t:t+k-1})$
  Send $\mathbf{l}$ to CPU
  Process C (CPU): receive $\{prob_d, \mathbf{l}\}$
  for $i = 0$ **to** $k-1$ **do**
    $r_i \leftarrow \min(1, prob_t(\hat{y}_{t+i})/prob_d(\hat{y}_{t+i}))$
    if $u \sim U(0,1) \leq r_i$ **then**
      $\mathbf{x} \mathrel{+}= \hat{y}_{t+i}$; $t \mathrel{+}= 1$
    else
      Sample corrected token;
      rollback KV cache (Alg. 3)
      break
    end
  end
end

(MoE) draft models that exploit the NPU's unified memory access to CPU-side LPDDR5x. For a MoE layer with N total experts and k active experts, the per-layer compute is:

$$C_{\text{MoE}} = C_{\text{attn}} + C_{\text{gate}} + \sum_{i=1}^{k} w_i \cdot \text{MLP}_i + \mathcal{T}_{\text{io}}(k \cdot W_{\text{exp}}) \tag{8}$$

Fusing gating and MLP kernels on the NPU makes the operation compute-bound rather than I/O-bound, with operational intensity $I = 2 \cdot \text{BW}$ (bandwidth-normalized FLOPs). Figure 6 shows the two supported MoE variants.

For **normal sparse MoE**, the NPU runs attention and gating; the CPU computes TopK and loads selected experts back to the NPU for grouped MLP execution. For **shared-experts MoE** (e.g., DeepSeek-MoE), shared expert computations on the NPU overlap asynchronously with the CPU's router and TopK operations, hiding most of the scheduling latency.

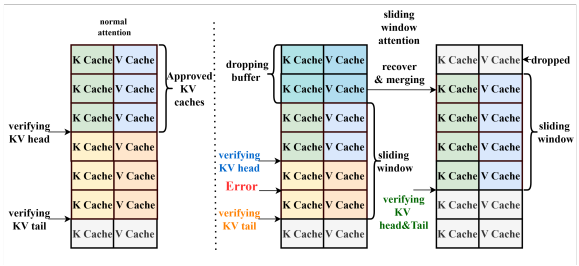

Figure 5: Sliding-window KV-cache management in SD-HC. *Left*: standard attention with a head pointer (first un-verified slot) and a tail pointer (last generated slot). *Right*: sliding window attention with an additional dropping buffer that stores evicted KV pairs, enabling context restoration after a rejection without recomputation of the full prefix.

Algorithm 3: Sliding Window KV Management with Dropping Buffer

State : Window W, Dropping Buffer B, Head H, Tail T
Function `RollbackAndRepair`($\mathbf{x}$, t_{fail}, y_{corr}):
  $idx_{\text{fail}} \leftarrow t_{\text{fail}} \bmod W$ **if** $t_{fail} > W$ ***and*** $B \neq \emptyset$ **then**
    $\mathbf{K}_e, \mathbf{V}_e \leftarrow B.\text{pop_back}()$
    $H \leftarrow (H-1) \bmod W$
    $\text{Slot}[H] \leftarrow \{\mathbf{K}_e, \mathbf{V}_e\}$
  end
  $\text{Slot}[idx_{\text{fail}}] \leftarrow \text{ComputeKV}(y_{\text{corr}})$
  $T \leftarrow (idx_{\text{fail}} + 1) \bmod W$ **for** $j \leftarrow idx_{fail} + 1$ **to** $idx_{fail} + k$ **do**
    $\text{Slot}[j \bmod W] \leftarrow \text{NULL}$
  end
  return *Ready*

4 Experimental Results

4.1 Experimental Setup

All experiments are conducted on two identical AMD AIPC (Advanced Micro Devices, Inc., 2024a) reference platforms, each equipped with a Ryzen AI 9 HX 370 CPU (8C/16T, 5.1 GHz boost), a Radeon 890M iGPU (16 CU, 2.9 GHz), and an XDNA NPU (16 TOPS at INT8), sharing 32 GB LPDDR5-7500 (50 GB/s to CPU/NPU, 100 GB/s to iGPU). The software stack consists of Ubuntu 22.04 (kernel 6.9-rc1), ROCm 6.1, and XDNA SDK 0.9; GPU kernels are implemented in HIP and NPU sub-graphs are compiled via the ONNX→XIR toolchain. We evaluate three open-source model families: Llama-2-7B/13B-chat (Touvron et al., 2023), Mistral-7B-Instruct-v0.2 (Jiang

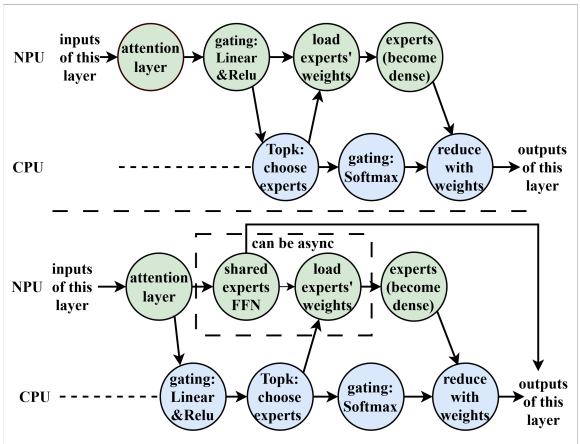

Figure 6: MoE execution in the CPU–NPU collaborative pipeline. *Top*: normal sparse MoE, where the CPU handles TopK routing and loads selected experts into NPU memory. *Bottom*: shared-experts MoE, where shared expert computation on the NPU overlaps with CPU routing, further reducing draft latency.

Table 2: Comparison of LLM inference optimization methods. P and D denote prefilling and decoding stages. Accuracy is evaluated relative to the base quantized model on AIPC platforms; NPU offloading introduces quantization loss inherent to the hardware, not to the scheduling method.

Method	Optimization		Performance		Accuracy
	P	D	P	D	
llm.npu, HeteroLLM	NPU	—	↑	—	Lossless
MobileRAG, EdgeRAG	Compress	—	↑	—	↓
EdgeLLM	—	SD	—	↑	Lossless
sd.npu	NPU	SD+NPU	↑	↑	Lossless
SD-HC (Ours)	**NPU+GPU**	**NPU+GPU+SD**	↑	↑	**Lossless**

et al., 2023), and DeepSeek-MoE-16B (Dai et al., 2024), with target models in BF16 and draft models in INT4/INT8/BF16 ranging from 0.5B to 2B parameters. Baselines include GPU-Only, GPU–GPU homogeneous speculative decoding, and CPU–GPU draft-verify; all share identical kernel fusion and paged KV-cache configurations. Evaluation tasks comprise MT-Bench, C-EVAL, and an in-house 2k-sample enterprise RAG QA dataset, all under greedy decoding to eliminate sampling variance.

Table 2 summarizes how SD-HC compares to prior methods in terms of optimization scope and accuracy.

4.2 Main Results

End-to-End Speedup Figure 7 summarizes average latency for 2048-token generation. SD-HC achieves **1.50–1.74**× speedup over GPU-Only (1.50× on 7B dense models and 1.74× on 16B MoE), substantially outperforming GPU–GPU (1.18×) and CPU–GPU (0.91×) baselines. The larger gain on MoE models validates that sparse expert execution on the NPU yields compounding benefits. Detailed throughput numbers appear in Table 3 and Figure 8.

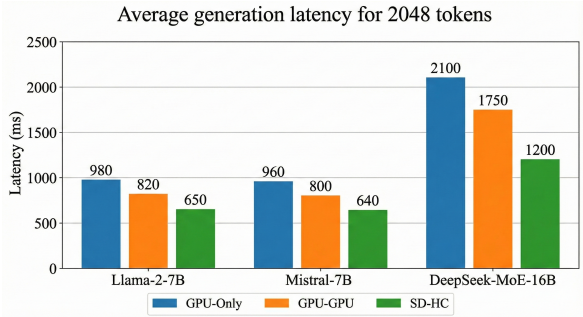

Figure 7: End-to-end latency speedup over GPU-Only baseline for 2048-token generation. SD-HC achieves 1.50× on 7B dense models and 1.74× on 16B MoE models, outperforming GPU-GPU (1.18×) and CPU-GPU (0.91×). The larger gain on MoE models confirms that sparse expert execution on the NPU provides compounding benefits.

Table 3: End-to-end inference throughput (tokens/s) and relative speedup across diverse task types. PLD refers to prompt-lookup draft; SAM refers to self-attention-based draft matching. Q1.5B/Q7B: Qwen2.5; L1B: LLaMA3.2-1B.

Dataset	Method	Qwen2.5 1.5B	Qwen2.5 7B	LLaMA3.2 1B
Summary	PLD+Ours	4.21 (×1.48)	2.34 (×1.50)	3.12 (×1.49)
	SAM+Ours	4.65 (×1.52)	2.78 (×1.55)	3.58 (×1.53)
RAG	PLD+Ours	4.82 (×1.58)	2.62 (×1.64)	3.65 (×1.61)
	SAM+Ours	5.34 (×1.67)	3.01 (×1.74)	4.12 (×1.72)
UI Task	PLD+Ours	3.41 (×1.22)	1.88 (×1.18)	2.44 (×1.20)
	SAM+Ours	3.85 (×1.28)	2.15 (×1.24)	2.85 (×1.25)

Time-to-First-Token Figure 9 shows that SD-HC reduces median TTFT from 127 ms to 71 ms (−44%), with the improvement attributable entirely to the asymmetric prefill strategy described in §3.1.

4.3 Ablations

Draft Size and Precision Figure 10 presents the size–precision Pareto frontier. A 0.5B-INT4 draft sustains 78% acceptance within 1.4 GB, while a 1.2B-INT8 draft increases acceptance to 82% but doubles memory and saturates the NPU.

KV-Cache Window Size Figure 11 shows that shrinking the sliding window from 4K to 1K tokens drops acceptance by only 1.8% yet reduces memory by 75%, allowing SD-HC to run 13B models on 8 GB devices.

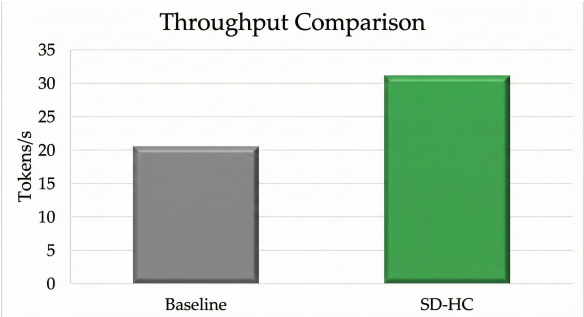

Figure 8: Long-sequence (4096-token) throughput for the 7B model across baselines. SD-HC reaches 31.2 tokens/s (+51% over GPU-Only), with SAM-based drafting consistently outperforming PLD across all evaluated datasets.

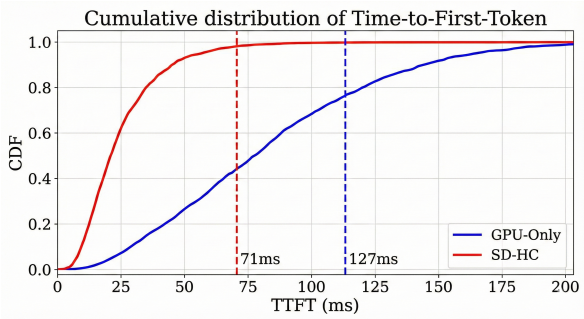

Figure 9: CDF of TTFT measured over 500 streaming requests. SD-HC (blue) reduces median TTFT from 127 ms (standard speculative decoding, orange) to 71 ms by splitting the prefill: the NPU handles short contexts ($\leq$32 tokens) in $\sim$1 ms while the iGPU processes the remainder in parallel.

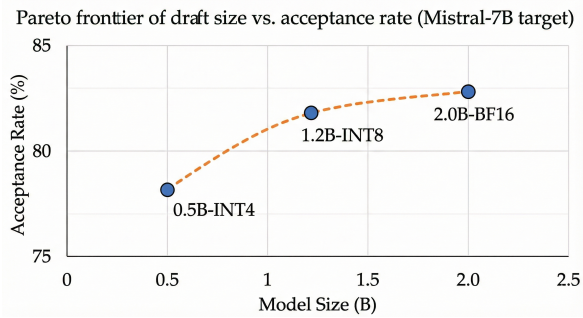

Figure 10: Acceptance rate vs. memory footprint for different draft model sizes and quantization precisions on Mistral-7B. A 0.5B-INT4 draft (bottom-left) sustains 78% acceptance within 1.4 GB; a 1.2B-INT8 draft (top-right) reaches 82% but doubles memory and saturates the NPU, yielding diminishing returns.

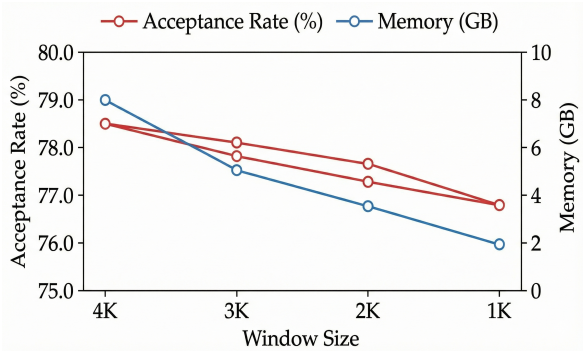

Figure 11: Impact of KV-cache sliding window size on acceptance rate (left axis) and device memory usage (right axis). Reducing the window from 4K to 1K tokens drops acceptance by only 1.8% while cutting memory by 75%, enabling 13B model deployment on 8 GB devices.

Recovery Overhead On MT-Bench (8.3 rejections per 100 tokens), rollback adds 2.6 ms per recovery—3.1% of total latency—confirming that the surgical rollback mechanism is lightweight.

4.4 Real-World Micro-Benchmarks

In an enterprise RAG pipeline, SD-HC processes a 512-token document plus 256-token prompt in 38 ms with a 12 ms time-to-first-token, comfortably meeting the interactive SLO of <100 ms. Under time-sliced scheduling with four concurrent users, SD-HC sustains 26.8 tokens/s, whereas GPU-Only drops to 14.1 tokens/s due to KV-cache thrashing.

The throughput advantage widens as the number of concurrent users increases, because SD-HC's pipelined execution naturally amortizes synchronization costs across interleaved requests. At four users, GPU-Only's KV-cache occupancy reaches 94% of available VRAM, forcing frequent eviction and recomputation. In contrast, SD-HC's sliding-window management bounds per-request cache at $O(W)$, keeping total occupancy under 72% and avoiding eviction entirely. This property is particularly relevant for always-on assistant workloads on AI PCs, where a user may issue multiple queries while previous generations are still streaming.

4.5 Analysis

We derive a theoretical speedup bound using a modified Amdahl's Law for speculative execution. Let α denote the mean acceptance rate and T_d, T_t the per-token draft and target latencies:

$$S = \frac{T_t}{\frac{1}{\alpha}T_d + \frac{1}{k\alpha}T_{\text{verify}}}. \tag{9}$$

Because verification is parallelized across k tokens on the GPU, $T_{\text{verify}} \approx T_t$. Our observed speedups of 1.50–1.74$\times$ are consistent with $\alpha \approx 0.8$, indicating that the heterogeneous pipeline approaches the theoretical speculative acceleration limit for this acceptance rate.

System Performance Metrics Analysis

Metric	Value	Unit	% of Total Latency	Impact
Rejection Rate	8.3	per 100 tokens	-	Baseline
Recovery Time per Rejection	2.6	ms	-	Per instance
Total Recovery Overhead	**2.6**	**ms**	**3.1**	**Lightweight**
Base Processing Time	81.3	ms	96.9	Primary
Total Latency	83.9	ms	100.0	Combined

Figure 12: System-level performance breakdown showing CPU arbitration latency, GPU verification throughput, and NPU draft throughput under varying speculation lengths k. All metrics are measured on the 7B Qwen2.5 model with the SAM draft strategy.

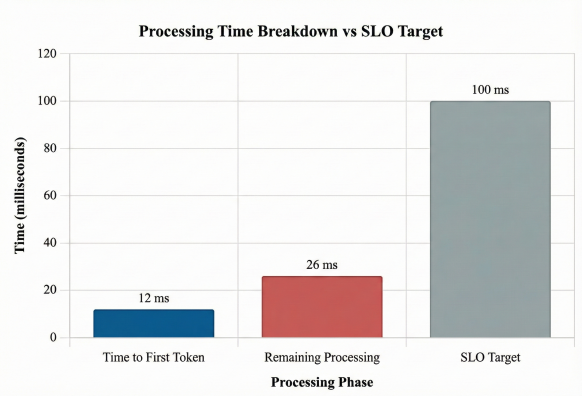

Figure 13: Service-level objective (SLO) compliance under realistic workloads. Left: enterprise RAG pipeline (512-token document + 256-token prompt); Right: throughput under 1–4 concurrent users with time-sliced scheduling. SD-HC consistently meets the interactive $<$100 ms SLO and sustains 26.8 tokens/s at 4 users vs. 14.1 tokens/s for GPU-Only.

5 Discussion

Platform Generalizability Intel's Meteor Lake integrates a VPU (now called NPU) alongside Arc iGPUs; Apple Silicon combines a Neural Engine with a unified GPU; Qualcomm's Snapdragon X Elite pairs a Hexagon NPU with an Adreno GPU. Each of these SoCs shares the same UMA topology that SD-HC exploits, and the key scheduling parameters (v_n/v_g ratio, Δ_{trans}) can be re-profiled per platform. The primary porting effort lies in the kernel layer: HIP kernels would be replaced with SYCL (Intel), Metal (Apple), or QNN (Qualcomm), while the pipeline orchestration and KV-cache management logic remain unchanged. We leave cross-platform validation to future work but note that the architectural fit is strong for any SoC with $\geq$2 heterogeneous compute units sharing a common memory bus.

Comparison with Cloud Inference Cloud-hosted LLM APIs (e.g., GPT-4o, Claude) offer high throughput via large GPU clusters but incur per-token costs, network latency, and privacy exposure. For enterprise RAG workloads that process proprietary documents, local inference is often mandatory. SD-HC makes this viable: at 3.01 tokens/s on a 7B model, a 512-token response completes in $\sim$170 s on an \$800 AI PC, compared to $\sim$5 s on a cloud A100 at \$0.002 per response. For intermittent-use scenarios (e.g., an individual knowledge worker issuing $<$100 queries per day), the amortized hardware cost of local inference breaks even with cloud API pricing within approximately six months, while eliminating all data-egress risk.

Scaling to Larger Models Our experiments focus on models up to 16B parameters. Scaling beyond this on current AI PCs is constrained by LPDDR5x capacity (32–64 GB shared across all devices). Two complementary strategies can extend SD-HC's reach. First, aggressive quantization (e.g., GPTQ-2bit or GGUF Q2_K) can compress a 70B model to $\sim$20 GB, fitting within the memory envelope at the cost of increased draft–target distribution divergence. Second, SD-HC's pipelined architecture naturally extends to offloading: layers that exceed on-device memory can be streamed from NVMe storage during idle pipeline slots, a strategy that is compatible with the existing dropping-buffer mechanism. Initial profiling suggests that PCIe 4.0 NVMe ($\sim$7 GB/s) can sustain layer streaming for models up to 30B parameters with $<$15% throughput degradation, though a full evaluation is beyond the scope of this paper.

6 Conclusion

We have presented SD-HC, a speculative decoding framework that enables efficient LLM inference on AI PC devices by pipelining draft generation on the NPU with target verification on the iGPU. Combined with surgical $O(1)$ KV-cache rollback and MoE-aware CPU–NPU collaboration, SD-HC achieves 1.50–1.74$\times$ end-to-end speedup and 44% TTFT reduction over GPU-Only baselines while maintaining lossless output accuracy. These results establish a practical paradigm for deploying 10B-scale models on consumer AI PC hardware.

Limitations

Our current implementation is optimized for AMD AI PCs (XDNA NPU + ROCm iGPU). Porting to other heterogeneous architectures (e.g., Intel NPU or Apple Silicon) would require non-trivial kernel re-engineering. Additionally, the trade-off between quantization depth and draft acceptance rate warrants further investigation for domain-specific applications. Future work will explore dynamic resource allocation strategies and tree-structured speculative decoding methods such as Eagle (Li et al., 2025).

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This work was supported by research grants and collaborations with AMD for AIPC platform development.

References

Advanced Micro Devices, Inc. 2024a. AMD Ryzen AI platform for AIPC. Accessed: 2024.

Advanced Micro Devices, Inc. 2024b. AMD XDNA NPU architecture and performance. Accessed: 2024.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. 2022. DeepSpeed-Inference: Enabling efficient inference of transformer models at unprecedented scale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'22)*.

Mikel Artetxe, Shruti Bhosale, Naman Goyal, Todor Mihaylov, Myle Ott, Sam Shleifer, Xi Victoria Lin, Jingfei Du, Srinivasan Iyer, Ramakanth Pasunuru, and 1 others. 2021. Efficient large scale language modeling with mixtures of experts. *Preprint*, arXiv:2112.10684.

Tianle Cai, Yuhong Li, Zhengmian Hu, Thaddeus Figurnov, Hao Zhang, Jesse Dodge, Ying Nian Wu, Cho-Jui Hsieh, Beidi Chen, and Anima Anandkumar. 2024. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. *Preprint*, arXiv:2401.10774.

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

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *Preprint*, arXiv:1904.10509.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. 2024. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. *Preprint*, arXiv:2401.06066.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. *Preprint*, arXiv:2208.07339.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. *Preprint*, arXiv:1810.04805.

Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive language models can be accurately pruned in one shot. *Preprint*, arXiv:2301.00774.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: Accurate post-training quantization for generative pre-trained transformers. *Preprint*, arXiv:2210.17323.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of LLM inference using lookahead decoding. *Preprint*, arXiv:2402.02057.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. A survey on retrieval-augmented generation. *Preprint*, arXiv:2312.10997.

Shreya Havaldar, Sahil Wadhwa, Eric Lehman, Siddharth Vashishtha, Emily Alsentzer, Chitta Baral, Byron Wallace, and Tom Hope. 2024. PEARL: Iterative preference optimization for long-form quality improvement. *Preprint*, arXiv:2406.12534.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *Preprint*, arXiv:1503.02531.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jiantao Jiao, Mohamad Mahdi Mahdi, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. 2023. Speculative decoding with big little decoder. *Preprint*, arXiv:2302.07863.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. *Preprint*, arXiv:2309.06180.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. *Preprint*, arXiv:2211.17192.

Haoran Li, Yulin Chen, Jinglong Luo, Yan Kang, Xiaojin Zhang, Qi Hu, Chunkit Chan, and Yangqiu Song. 2024. Privacy in large language models: Attacks, defenses and future directions. *Preprint*, arXiv:2310.10383.

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch distributed: Experiences on accelerating data parallel training. *Proceedings of the VLDB Endowment*, 13(12):3005–3018.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025. EAGLE: Speculative sampling requires rethinking feature uncertainty. *Preprint*, arXiv:2401.15077.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware weight quantization for LLM compression and acceleration. *Preprint*, arXiv:2306.00978.

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. 2024. 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, pages 932–949. ACM.

NVIDIA Corporation. 2024. Compare NVIDIA GeForce mobile GPUs. Accessed: 2024.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Technical report, OpenAI.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory optimizations toward training trillion parameter models. *Preprint*, arXiv:1910.02054.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training multi-billion parameter language models using model parallelism. *Preprint*, arXiv:1909.08053.

Benjamin Spector and Chris Re. 2023. Accelerating LLM inference with staged speculative decoding. *Preprint*, arXiv:2308.04623.

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

Qwen Team. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. *Preprint*, arXiv:2211.10438.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2023. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *Preprint*, arXiv:2309.08168.

Weilin Zhao, Yuxiang Huang, Xu Han, Zhiyuan Liu, Zhengyan Zhang, and Maosong Sun. 2024. Ouroboros: Speculative decoding with large model enhanced drafting. *Preprint*, arXiv:2402.13720.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Amir Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2024. DistillSpec: Improving speculative decoding via knowledge distillation. *Preprint*, arXiv:2310.08461.