In STeP: Speculative Tensor Parallelism for Concurrent Heterogeneous Inference of LLMs

NeurIPS 2026



Viren Luke Radhakrishnan



Dhruva Kashyap



Pranav K Nayak



Chiranjib Bhattacharyya



Prakash Raghavendra

AMD
together we advance_

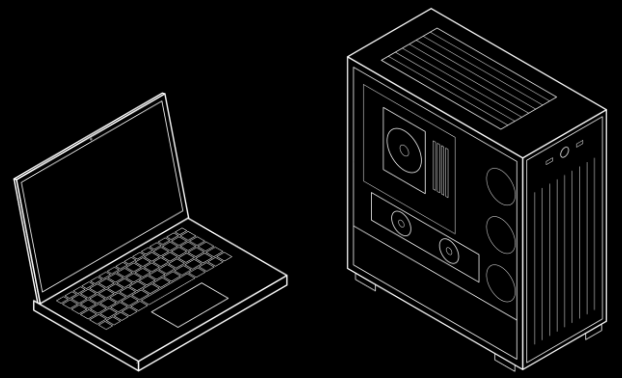
INDIAN INSTITUTE OF SCIENCE
भारतीय विज्ञान संस्थान

Running LLMs on Consumer PCs

- **Consumer PC:**

  Good CPU + large memory | powerful GPU + **limited** memory

Laptops	16C Ryzen 9 7945HX CPU	**16 GB** Radeon RX 7900M GPU
Desktops	16C Ryzen 9 9950X CPU	**32 GB** Radeon AI PRO R9700 GPU



- Running large LLMs (24B+) **locally** (e.g., on AMD Radeon GPUs in consumer and workstation PCs) is desirable for **privacy, cost, and offline use**

88.8% of gaming PCs have <= 16 GB of VRAM	**64 GB*** needed for a 24B model in BF16, 2x the largest consumer GPU	**6–17×** cheaper to run locally than cloud APIs at sustained volume

- A model may not fit on the GPU even when the system has plenty of CPU memory
- Layer-splitting approaches offer a naïve way to use both devices
- Heterogeneous speculative decoding is the dominant lossless-acceleration framework in this regime

---

Sources: Steam, Lenovo

* including KV cache

AMD
together we advance_

Background and Related Work

Speculative Decoding [4][19]
Accelerates autoregressive generation by having a fast draft model $M_{\theta D}$ propose γ candidate tokens, which a full verifier model $M_{\theta V}$ validates in a single forward pass with rejection sampling guaranteeing identical output distribution

- **Layer skipping methods** [9][43][48]
  Execute a subset of transformer layers
- **Auxiliary prediction heads** [3][23]
  Attach lightweight modules for parallel token prediction
- **Tree-based methods** [3][7][28][51]
  Produce a branching tree of candidates verified via tree attention masks in a single forward pass
- **SubSpec** [40]
  Generates low-precision substitute layers from the target

Structured pruning for LLMs
Removes entire channels from weight matrices
A subset of methods, like **ModHiFi** [18], **Wanda** [37], **SliceGPT** [1], and **SVD-LLM** [41], produce channel importance scores where selecting the top-k channels yields a smaller model that approximates the output of the original

Tensor Parallelism [35]
Distributes weight matrices across multiple devices, combining partial results via all-reduce. It targets training and autoregressive inference on homogeneous multi-GPU configurations

Offloading and memory-constrained inference

- **FlexGen** [34]
  Executes layers on the GPU, but pipelines them across GPU memory, CPU memory, and disk
- **SpecExec** [38] and **SpecInfer** [28]
  A trained draft model resides permanently on the GPU, while verifier layers are streamed in groups from CPU memory, to the remaining GPU memory, for execution
- **Dovetail** [49]
  Keeps a trained draft head on the GPU and partitions the verifier across the remaining GPU memory and CPU memory. The layers that fall in CPU memory are executed by the CPU
- **PowerInfer** [36]
  Profiles activation locality offline and assigns frequently-activated “hot” neurons to the GPU while “cold” neurons are computed on the CPU

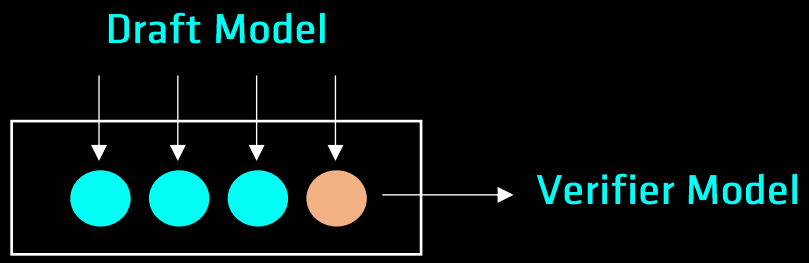
(Reference numbers are from the paper)

AMD
together we advance_

Overview of Speculative Decoding Approaches

- **Autoregressive Generation**
  One token generated at a time

- **Speculative Decoding**
  Fast “draft” model proposes tokens that the full “verifier” model processes in a single pass



- **Heterogeneous Speculative Decoding**
  The “draft” model sits on the GPU, the full “verifier” model on the CPU



- **Self-Speculative Decoding**
  Deriving the draft from the verifier itself, rather than a separate, expensively trained model



AMD
together we advance_

The Heterogeneity Gap

- Can we use **both CPU & GPU compute/memory** without sacrificing inference speed? What's the best we can do?

Existing approaches waste available hardware in 3 ways:

1. CPU & GPU do not work at the same time
Only one performs useful work, while the other idles

2. Unused GPU models are dead-weight
They waste VRAM when not used for inference

3. Available VRAM is not fully exploited
Particularly when I/O is hidden behind compute

We call this unused performance opportunity the **Heterogeneity Gap**

Any speculative-decoding-based solution must employ the following:

- **Co-Execution**
  Requires a split along the intermediate dimension of each layer rather than by layer
- **Subnetwork Draft**
  Derive the draft as a subnetwork of the verifier itself rather than a separate set of weights
- **Maximal VRAM Utilization**
  Use a near-full fraction of the available VRAM, reserving the remainder for the KV cache and activations

AMD
together we advance_

Our Contributions

Extraction of subnetwork drafts

We exploit the column-wise linearity of the FFN down-projection, leveraging saliency-based methods to reorder channels within the FFN layers of a model

Quality of subnetwork drafts

We prove that the rejection rate of speculative sampling is upper-bounded by the square root of the local reconstruction error of a masked FFN layer

Speculative Tensor Parallelism (STeP)

We propose STeP, a self-speculative inference algorithm where the GPU-resident channels serve as both a draft model and a shard of the full model

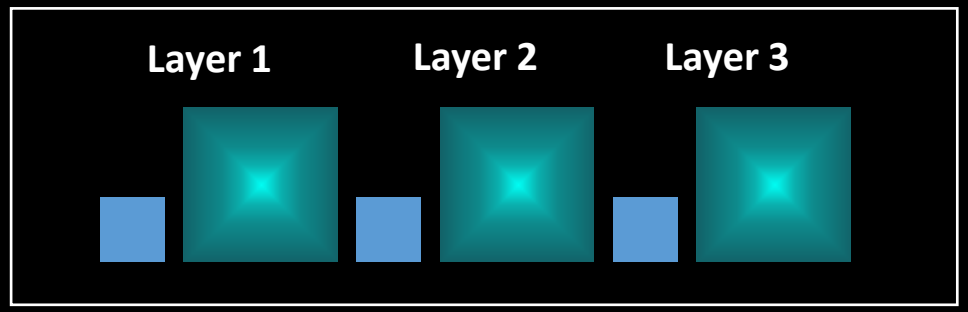
Empirical validation across scales

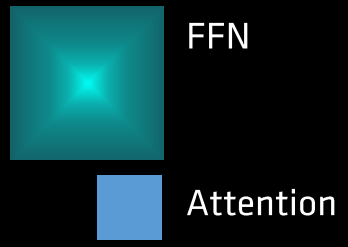
Across dense models from 24B to 123B, GPU memory budgets from 32 to 192 GB, and five benchmarks, STeP outperforms two leading approaches and closes the heterogeneity gap

AMD
together we advance_

Exact FFN Decomposition

- An LLM is a collection of **sequential layers**. Each has an "Attention" component and a "Feed Forward Network" (FFN) component





- The 3 FFN matrices ($W_{\text{gate}}, W_{\text{up}} \in \mathrm{R}_{d\text{ff}\times d}$ and $W_{\text{down}} \in \mathrm{R}_{d\times d\text{ff}}$) can be split at any point, and both pieces in each (prefix and residual) can be **computed independently**

Model dimension: d intermediate dimension: d_{ff} split point $k \in [0, d_{\text{ff}}]$:

$$\mathrm{FFN}(x) = \underbrace{W_{\text{down}}[:, :k]\left(\mathrm{SiLU}(W_{\text{gate}}[:k, :]x) \odot W_{\text{up}}[:k, :]x\right)}_{\mathrm{FFN}_{\text{prefix}}(x)} + \underbrace{W_{\text{down}}[:, k:]\left(\mathrm{SiLU}(W_{\text{gate}}[k:, :]x) \odot W_{\text{up}}[k:, :]x\right)}_{\mathrm{FFN}_{\text{remaining}}(x)}$$

Let $r \in (0, 1]$ denote the **active ratio** (fraction of FFN channels considered as the "prefix"), with $k = \lfloor r \cdot d_{\text{ff}} \rfloor$.

"Split a layer in two, compute the pieces separately, add them back, and you get the exact same answer."

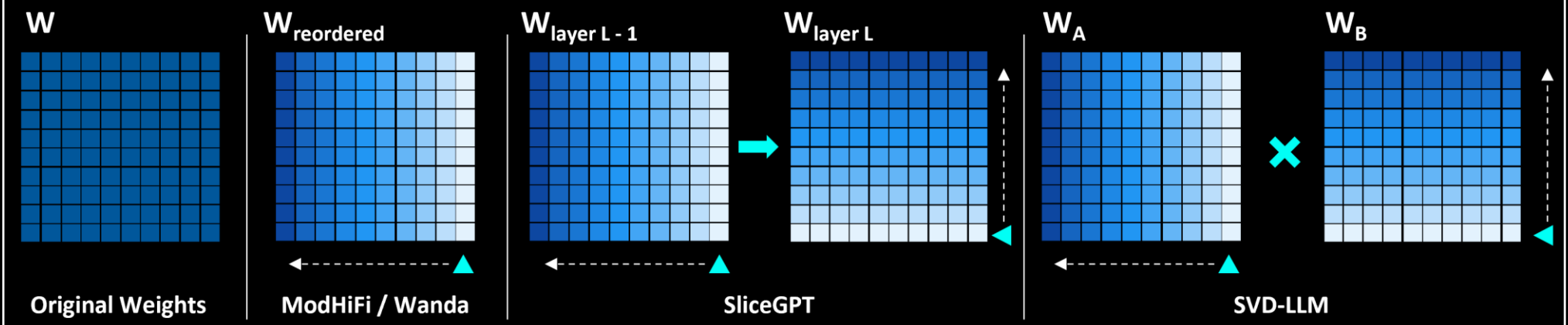
- This **"prefix"** can independently act as a **draft model**, and we place this shard on the **GPU**
- The **"remaining"** goes on the **CPU**

AMD
together we advance_

Subnetwork Drafts via Structured Pruning

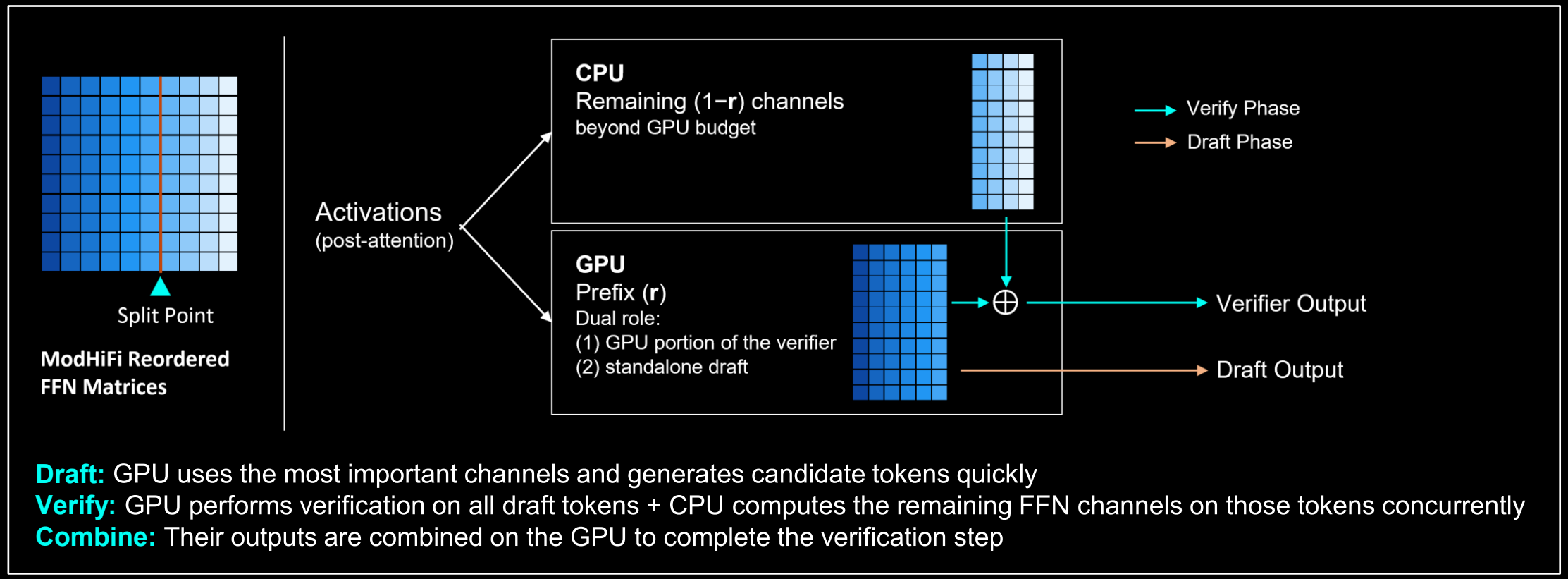
- **Ordering matters:** not all channels are equally useful. If we split the channels arbitrarily, the subnetwork on the GPU may be a poor draft model
- So before splitting, we **reorder the FFN channels by saliency**. Structured pruning methods give us a **hint** on how to do this

- **ModHiFi/Wanda** reorder channels by saliency so the top-k form a contiguous prefix
- **SliceGPT** applies orthogonal rotations across adjacent layers, concentrating information in leading dimensions
- **SVD-LLM** factorizes each weight matrix into $W_A \times W_B$ via activation-aware SVD



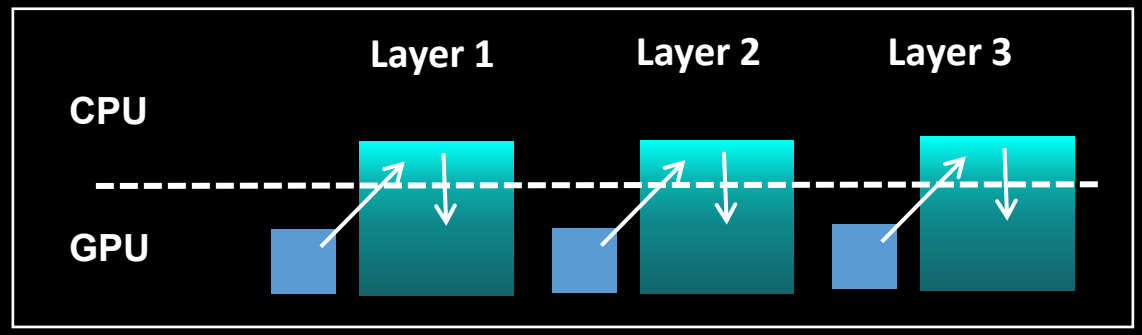
AMD
together we advance_

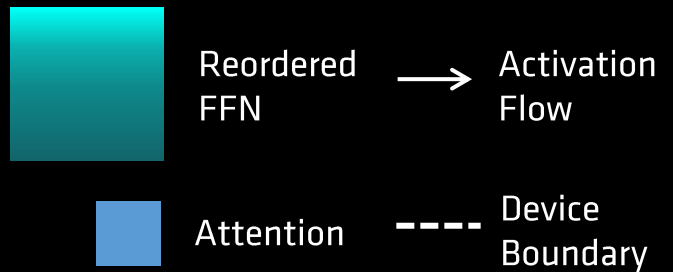
Speculative Tensor Parallelism (STeP)



Draft: GPU uses the most important channels and generates candidate tokens quickly
Verify: GPU performs verification on all draft tokens + CPU computes the remaining FFN channels on those tokens concurrently
Combine: Their outputs are combined on the GPU to complete the verification step

Attention remains entirely on the GPU and only **activations** cross the boundary.



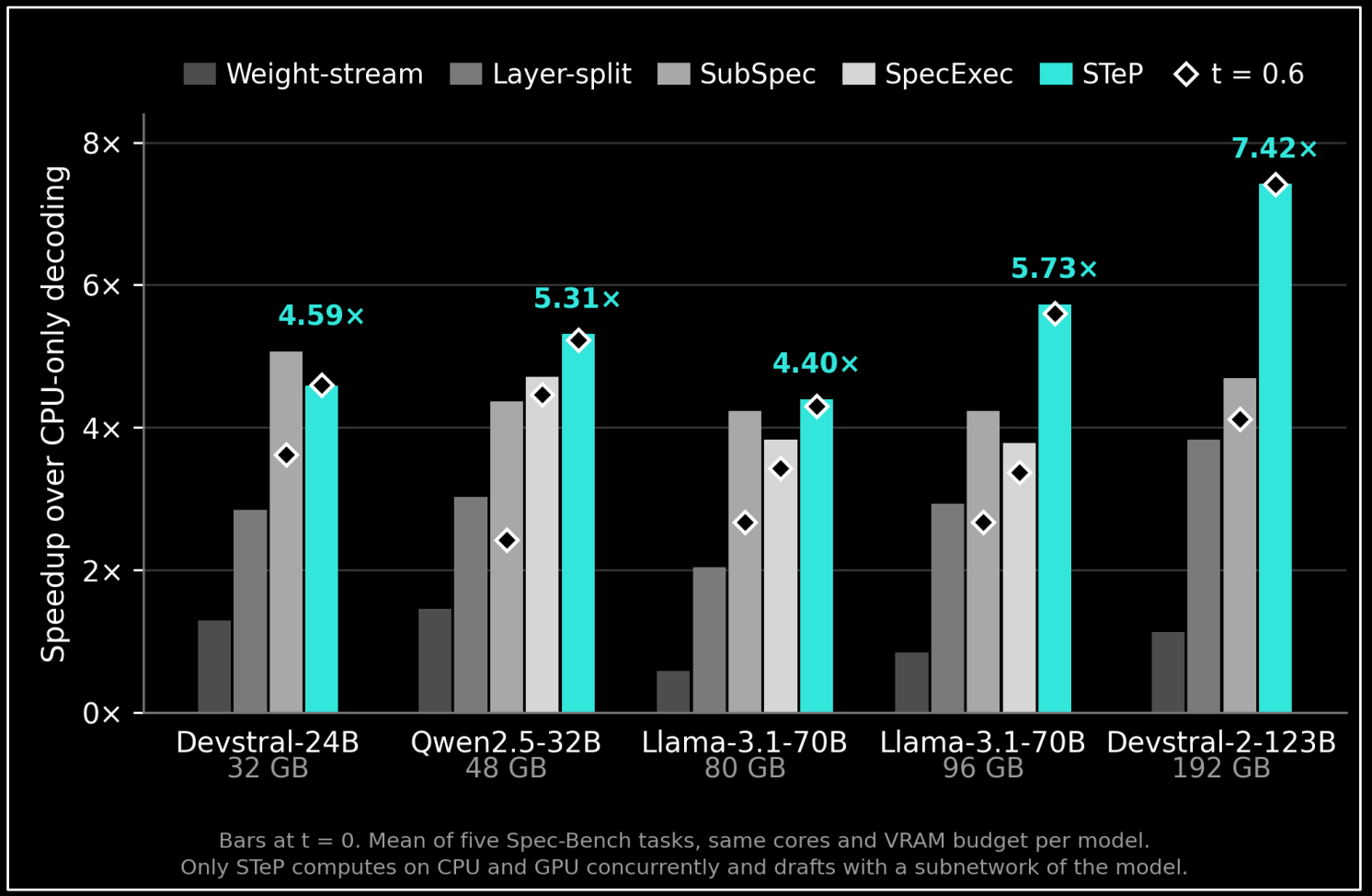


AMD
together we advance_

Results

Try it out today: https://github.com/amd/AMD-PACE

	➢ t=0, up to	➢ t=0.6, up to
	2.2x Layer-split	**2.1x** Layer-split
	7.6x Weight-stream	**7.4x** Weight-stream
State of the art	**1.6x** SubSpec	**2.2x** SubSpec
	1.5x SpecExec	**1.7x** SpecExec



- EPYC™ 9755 128-core **CPU**; Instinct™ MI355X **GPU**; **BF16** precision
- **SpecExec** pairs each target with an external trained draft, and **SubSpec** uses the authors' recommended quantized substitute
- **Throughput** averaged over five Spec-Bench benchmarks, at sampling temperatures t=0 and t=0.6 with 128 output tokens per sample

AMD
together we advance_