

CONSUMERBENCH: Benchmarking Generative AI Applications on End-User Devices

Yile Gu^{*1} Rohan Kadekodi^{*1} Sahi Chitrapu¹
Aradhya Agrawal² Camille Sawa¹ Hoang Doan Nguyen¹
Keisuke Kamahori¹ Yiyu Liu^{†1,3} Baris Kasikci¹

¹University of Washington ²Indian Institute of Technology Delhi

³Shanghai Jiao Tong University

Abstract

The recent shift in Generative AI (GenAI) applications from cloud-only environments to end-user devices introduces new challenges in resource management and system efficiency. This paper presents CONSUMERBENCH, a comprehensive benchmarking framework designed to evaluate the system efficiency and response time of GenAI models running on end-user devices. Unlike existing benchmarks that assume exclusive model access on dedicated GPUs, CONSUMERBENCH simulates realistic multi-application scenarios executing concurrently on constrained hardware. Furthermore, CONSUMERBENCH supports customizable workflows that simulate complex tasks such as multi-agent pipelines involving multiple GenAI models and cross-application dependencies. CONSUMERBENCH monitors Service-Level Objectives (SLOs) of applications, as well as system-wide metrics like power, CPU/GPU utilization and memory bandwidth. Through extensive experiments, CONSUMERBENCH reveals inefficiencies in existing GPU sharing strategies under realistic concurrent executions. The paper also provides practical insights for system and kernel designers, highlighting the importance of implementing SLO-aware GPU sharing strategies, and concurrency-aware kernel implementations to avoid resource contention.

1 Introduction

The deployment landscape for Generative AI (GenAI) models is undergoing a significant transformation: once confined to hyperscale data centers equipped with powerful GPUs, these models are now increasingly being adopted on local devices such as laptops and smartphones (Android Developers, 2025; Apple Inc., 2024; Qualcomm Technologies, Inc., 2023). This shift is primarily motivated by growing concerns over data privacy, latency, and availability under various network conditions.

Meanwhile, GenAI workflows have become substantially more complex, involving multiple stages and modalities: creating a video tutorial on an end-user device may invoke a language model for script planning, a deep research agent to survey tools, an image generation model for cover art, and a speech recognition model for live captions, often coordinated through agentic pipelines. (Aflalo et al., 2025; Fireworks AI, 2025). At the same time, users increasingly run multiple GenAI models concurrently across applications. For example, interacting with an on-device AI chatbot while a real-time image or audio model is executing, is a common practice (Qualcomm, 2024; Karami et al., 2025). Unlike the cloud, where such models can be distributed across dedicated servers and GPUs, end-user devices must accommodate *all* of these models on a constrained set of resources (such as a single GPU), shared across applications.

Existing GenAI models and inference frameworks (Dettmers et al., 2023; Frantar et al., 2023; Lin et al., 2024; Gerganov & ggml-org contributors, 2023) typically assume exclusive, dedicated hardware access, issuing kernels that perform poorly under concurrency (Xu et al., 2022; Strati et al., 2024), or holding static resource allocations that hoard the GPU

^{*}Both authors contributed equally. [†]Work done as intern at the University of Washington.

even when idle (Kwon et al., 2023; Yu et al., 2022). GPU sharing strategies for concurrently running GenAI applications (Zhao et al., 2025; Wu et al., 2023; Coppock et al., 2025) coarsely classify applications as high-priority or best-effort, ignoring diverse SLOs, inter-application dependencies, and application resource utilization. Existing benchmarks reflect this same assumption: benchmarks for GenAI inference evaluate models in isolation (Reddi et al., 2020; Li et al., 2025; Jayanth et al., 2024; Laskaridis et al., 2024; Xiao et al., 2024), while benchmarks for GPU sharing strategies treat concurrent applications as independent, ad-hoc processes with no semantic relationships (Han et al., 2022; Zhao et al., 2023; Chakraborty et al., 2025).

We present CONSUMERBENCH, a benchmarking framework that evaluates GenAI applications on end-user devices under realistic multi-application, resource-constrained conditions. Unlike prior work, CONSUMERBENCH models the *semantic structure* of real workloads: users describe applications as a workflow of typed steps, spanning diverse paradigms such as agentic pipelines, Retrieval-Augmented Generation (RAG), image generation, and live transcription, in a unified configuration, with explicit inter-step dependencies that wire the output of one step as the input to the next. Users additionally attach per-application SLOs to each step (*e.g.*, per-token latency for a chatbot, end-to-end latency for a deep research agent, real-time factor for live captions), and CONSUMERBENCH evaluates how well each SLO is met across a range of deployment scenarios, including static and dynamic GPU sharing strategies, hybrid CPU/GPU setups, and so on. Beyond application-level metrics, CONSUMERBENCH collects system-level telemetry (GPU/CPU utilization, memory bandwidth, power consumption) to identify inefficiencies in resource management that emerge when structured, dependency-aware workloads compete on shared resources.

Through the evaluation of a diverse set of GenAI applications and workflows on consumer-grade hardware, CONSUMERBENCH discovers novel bottlenecks and opportunities for efficient infrastructure tailored to support end-user devices. CONSUMERBENCH reveals that existing static and dynamic GPU sharing strategies are unable to effectively meet the SLOs of concurrent applications. Furthermore, GenAI models designed assuming exclusive GPU access cause severe resource contention when run concurrently, highlighting the need for concurrency-aware kernel implementations.

In summary, this paper makes the following contributions:

- It highlights unique challenges in running diverse GenAI applications concurrently on end-user devices, including resource contention and unmet SLOs, that are not exposed by prior benchmarks.
- It introduces a benchmarking framework called CONSUMERBENCH that supports realistic multi-application workflows, tracks both application- and system-level metrics, and allows flexible configurations for evaluation on end-user devices.
- It evaluates diverse applications and workflows such as agentic pipelines on different GPU sharing strategies, reveals key system inefficiencies, and offers unique insights into building efficient systems for end-user devices.

2 Related Works

2.1 Modern on-device AI applications

Recent advances in resource-efficient models (Abdin et al., 2024; Meta AI, 2024; Yang et al., 2024), model compression (Dettmers et al., 2023; Frantar et al., 2023; Lin et al., 2024), efficient runtimes (Gerganov & ggml-org contributors, 2023) and edge-oriented hardware (Apple Inc., 2024; Qualcomm Technologies, Inc., 2023; Android Developers, 2025) have made credible *on-device* inference possible.

GenAI inference has evolved from a simple chatbot, to complex multi-step workflows spanning different models with various input and output modalities, invoking language models for reasoning, vision models for image understanding and generation, and speech models for transcription, coordinated through agentic pipelines (Aflalo et al., 2025; Fireworks AI, 2025). Industry reports and user studies further show that users run multiple such applications concurrently on the same device (Qualcomm, 2024; Karami et al., 2025).

Benchmark	SLO-aware	System Perf.	Workflow Modeling	Supported Modalities	GPU Sharing
MLPerf Inference (Reddi et al., 2020)	Yes	Yes	–	Multi-modal	–
PalmBench (Li et al., 2025)	–	Yes	–	Text	–
MELT (Laskaridis et al., 2024)	–	Yes	–	Text	–
Mobile-Bench (Deng et al., 2024)	–	–	Yes	Text	–
MobileAIBench (Murthy et al., 2024)	–	Yes	–	Multi-modal	–
LithOS (Coppock et al., 2025)	–	Yes	–	Multi-modal	Yes
CONSUMERBENCH	Yes	Yes	Yes	Multi-modal	Yes

Table 1: Comparison with representative benchmarks and systems under 5 criteria: SLO-aware (evaluating application-specific metrics), System Perf (evaluating system-level performance metrics), Workflow Modeling (evaluating workflows and multi-step dependencies), Supported Modalities (types of models supported), and GPU Sharing (evaluating concurrent execution of applications on a GPU).

2.2 Benchmarks for on-device AI inference

As Table 1 summarizes, MLPerf Inference (Reddi et al., 2020), PalmBench (Li et al., 2025), MELT (Laskaridis et al., 2024), MobileAIBench (Murthy et al., 2024), and related benchmarks (Xiao et al., 2024; Jayanth et al., 2024) measure individual-model system performance. Mobile-Bench (Deng et al., 2024) evaluates the accuracy of multi-step agent tasks but not system performance. GPU-sharing studies (Strati et al., 2024; Han et al., 2022; Zhao et al., 2023; Chakraborty et al., 2025; Coppock et al., 2025) execute workloads concurrently but without semantic dependencies. In contrast, CONSUMERBENCH benchmarks the concurrent execution of *heterogeneous* workflows comprising multiple GenAI models, and inter-application dependencies, and diverse SLOs, thereby highlighting cross-application interference patterns and system inefficiencies.

3 CONSUMERBENCH

CONSUMERBENCH is a comprehensive benchmarking framework to evaluate the runtime performance of GenAI applications on end-user devices under realistic conditions.

3.1 Goals

- **G1: Evaluate diverse applications.** CONSUMERBENCH supports various GenAI applications. They range from long-running background services to short-lived, latency-sensitive interactive tasks with tight SLOs, and with different modalities (*e.g.*, text-to-text, text-to-image). See §3.3.
- **G2: Evaluate concurrent execution of applications.** CONSUMERBENCH evaluates performance and SLO attainment when multiple applications run concurrently on shared and limited hardware and use different GPU sharing strategies. See §3.2.
- **G3: Evaluate end-to-end user workflows.** CONSUMERBENCH evaluates workflows consisting of multiple applications that collaborate to perform end-to-end user tasks. This includes multi-agent pipelines consisting of GenAI models and tool invocations, Retrieval-Augmented Generation (RAG) pipelines, and so on. See §3.2.
- **G4: Collect system-level holistic metrics.** CONSUMERBENCH collects detailed metrics such as compute utilization, memory bandwidth utilization, and power consumption. These metrics help users not only understand whether an application meets its SLO, but also diagnose why it might fail, which is crucial for improving system design and optimizing on-device deployment strategies. See §3.2.

3.2 Methodology

Fig. 1 shows the design overview of CONSUMERBENCH. Given a user configuration of the workflow, CONSUMERBENCH orchestrates the execution of the workflow through a

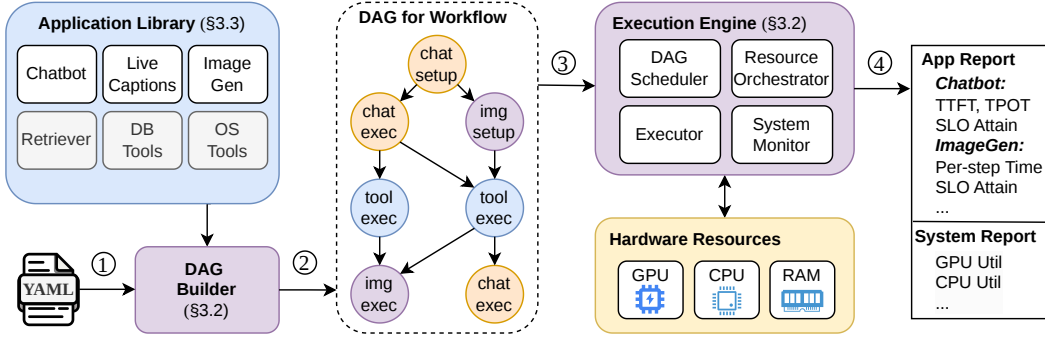


Figure 1: The overall design of CONSUMERBENCH.

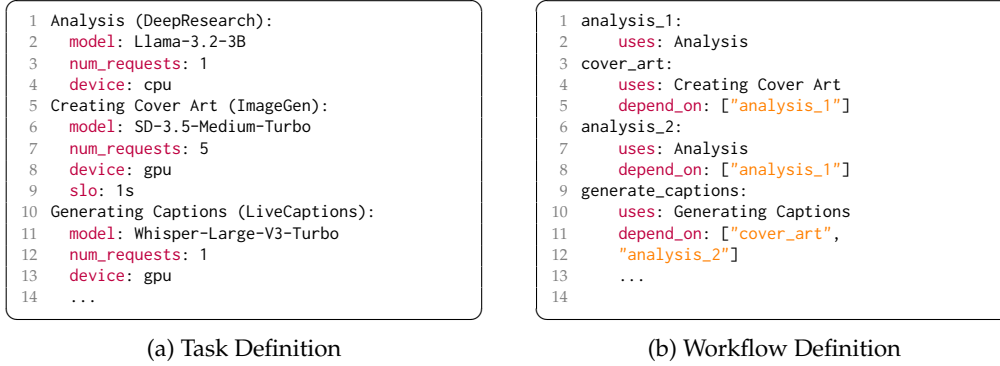


Figure 2: Example YAML configuration to define application tasks as well as user workflows.

graph representation. After the workflow finishes, it generates a benchmark report for SLO attainment and system-level metrics.

Input user configuration (①). The input to CONSUMERBENCH is a YAML configuration file that contains the set of applications to run, hardware placements (CPU, GPU, or CPU-GPU hybrid), SLOs, and input requests. Applications may be GenAI applications (*e.g.* Chatbot, Image Generation, etc), or external tools (*e.g.* database or OS tools, or retriever for RAG). The YAML configuration also specifies dependencies between applications. The YAML configuration can be used to specify complex scenarios such as a multi-agent pipeline, a RAG pipeline, or a user request containing multiple steps. Fig. 2 provides an example configuration for three applications (Deep Research agent, Image Generation, and Audio Captioning), with dependencies and SLOs.

Creating overall workflow (②). CONSUMERBENCH builds a directed acyclic graph (DAG) from the YAML configuration. Each node represents an application instance, with edges denoting dependencies. Nodes are of three types: setup (application startup), exec (execution), and cleanup (resource release). CONSUMERBENCH validates the DAG to ensure that there are no cycles and that each application includes a setup node before any exec nodes. Fig. 1 includes an example DAG.

Executing the workflow (③). The execution engine uses the DAG to coordinate application execution and collect metrics. Users can experiment with different dependencies between steps to generate different DAGs, and check execution.

- The **DAG scheduler** manages request scheduling, respects dependencies, and enables concurrent execution where possible, supporting realistic user-defined workflows.
- The **resource orchestrator** employs various GPU sharing strategies (such as time-slicing, static GPU partitioning, dynamic GPU scheduling, and so on) to execute workflows.
- The **executor** is responsible for loading and unloading of the GenAI models and executing user requests, obeying the DAG scheduler.

- The **system monitor** tracks system-wide resource usage during execution. GPU compute and memory bandwidth usage are monitored with the help of NVIDIA’s Data Center GPU Management (DCGM) utility (Corporation, 2025b).¹ CPU utilization is monitored using the `stat` tool, while CPU memory (DRAM) bandwidth utilization is monitored using the `pcm-memory` utility (Corporation, 2025a). For NVIDIA GPUs, CONSUMERBENCH monitors power consumption using NVML (NVIDIA Corporation, 2025) for NVIDIA GPUs and powermetrics utility (SS64, 2025) for Apple Silicon.

Generating benchmark report (④). After completing the workflow, CONSUMERBENCH automatically evaluates each application’s performance against the SLOs defined in the YAML configuration. It then generates a comprehensive report summarizing performance, SLO satisfaction, and resource efficiency for the entire workflow.

3.3 Supporting diverse applications and models

CONSUMERBENCH provides an API that enables users to integrate their own applications with either custom or existing models, making the framework highly extensible. To evaluate a custom application, users simply implement three functions: `setup()`, to initialize the application (for example, loading the model for a GenAI application to the GPU); `execute()`, to send requests to the application; and `cleanup()`, to release application resources. CONSUMERBENCH also allows users to configure datasets where the requests are sampled from, enabling evaluation of applications under realistic workloads. Table 2 shows the applications currently supported by CONSUMERBENCH, covering different modalities and common use cases on end-user devices.

Application	Model	SLO
Chatbot	Llama-3.2-3B (Meta AI, 2024)	TTFT: 1s, TPOT: 0.25s
DeepResearch	Llama-3.2-3B (Meta AI, 2024)	N/A
ImageGen	SD-3.5-Medium-Turbo (Studios, 2024)	Step Time: 1s
LiveCaptions	Whisper-Large-V3-Turbo (Radford et al., 2022)	Per-Segment Time: 2s
DBTools, OSTools	N/A	N/A
Retriever	MiniLM-L6-v2 (Wang et al., 2021)	N/A

Table 2: Summary of model and SLO used in each application.

Chatbot. This is a text-to-text generation app for chat and Q&A, featuring a frontend and a local backend with an OpenAI-compatible API (OpenAI, 2020). The backend can use any inference server, such as vLLM (Kwon et al., 2023), or llama.cpp (Gerganov & ggml-org contributors, 2023), for CPU-GPU co-execution. SLO targets are based on human reading speed (Liu et al., 2024): 1 second for Time to First Token (TTFT) and a generation speed of 4 tokens per second (Time Per Output Token - TPOT). Dataset: LMSYS (Zheng et al., 2024).

DeepResearch. This is an application for complex, multi-step reasoning and fact gathering, adapted from smolagent’s open-deep-research (Roucher et al., 2025) and uses LiteLLM (BerriAI, 2024) to interact with a local model via inference server. DeepResearch operates on long contexts and functions as a persistent background application. Dataset: DeepResearch-Bench (Du et al., 2026).

ImageGen. This is a text-to-image application with a simple frontend and a backend using stable-diffusion-webui (AUTOMATIC1111, 2022) in API server mode. Motivated by research on diffusion models on resource-constrained devices (Zhao et al., 2024; Li et al., 2023a), the SLO for ImageGen is set to 1 second per denoising step (Gujarati et al., 2020; Li et al., 2023b; Chen et al., 2025b). Dataset: COCOCaption (Chen et al., 2015).

LiveCaptions. This is an audio-to-text application for real-time scenarios (Macháček et al., 2023). The frontend chunks an audio file into segments and sends each segment to a backend adapted from whisper-online (Macháček et al., 2023) to support HTTP connections. LiveCaptions can either be latency-sensitive or background. For the latency-sensitive case, a 2-second audio segment is sent every 2 seconds, requiring the model to generate captions in

¹Although designed for datacenter GPUs, DCGM supports consumer-grade GPUs, including all GeForce GPUs.

time. The SLO for this scenario is thus 2 seconds. For background transcription of a large audio file (5–10 minutes), there is no SLO. Dataset: Earnings21 (Del Rio et al., 2021).

External Tools (DBTools, OSTools, Retriever). CONSUMERBENCH also includes three non-generative applications that represent common tool-use and retrieval workloads in agentic pipelines. DBTools executes SQL queries against ephemeral MySQL instances in Docker. OSTools executes bash commands in Ubuntu Docker containers for system administration tasks such as file manipulation. Both tools are ported from AgentBench (Liu et al., 2023). They are I/O- and CPU-bound with no GPU usage or latency SLOs, but contribute realistic resource contention alongside GPU-bound applications. Retriever is a RAG component that encodes queries and retrieves top-k passages from a FAISS index (Douze et al., 2025) via approximate nearest-neighbor search. It can run on CPU or GPU.

4 Experimentation

We present key benchmarking results of CONSUMERBENCH; additional results are in the Appendix. Our main goal is to understand GenAI applications’ resource utilization and performance when executing in isolation, concurrently, and or end-to-end user workflows.

We organize our experimentation into three scenarios. First, we characterize each application in isolation, profiling its GPU and CPU utilization, resource consumption patterns, and latency performance relative to SLOs (§4.1). Second, we study concurrent GPU execution of multiple applications with diverse SLOs under different static and dynamic GPU sharing strategies (§4.2). Third, we capture dependencies between applications in an end-to-end multi-agent workflow, where we monitor system-wide metrics and SLO violations for the different GPU sharing strategies (§4.3).

Experimental setup. We run experiments on a local server consisting of a single RTX 6000 GPU with 24GB VRAM. The server is equipped with an Intel Xeon Gold 6126 CPU (2.60GHz, 24 cores) and 32GB of system memory (DRAM). The applications evaluated are listed in Table 2. Note that CONSUMERBENCH is not limited to the use of these applications; users can add more applications by following the procedure in §3.3. For each application, CONSUMERBENCH samples requests from the dataset, measures per-request latency, and compares the results to the defined SLOs. To show the extensibility of CONSUMERBENCH, we additionally run a subset of our experiments on other platforms (Apple Silicon, NVIDIA Jetson Orin Nano, NVIDIA RTX 3060, NVIDIA RTX 4080), presented in §D.

System-wide metrics. While CONSUMERBENCH monitors a range of resource utilization metrics such as compute, memory, and power (see §3.2), we focus primarily on GPU utilization. Additional metrics are detailed in the §B. For GPU utilization, CONSUMERBENCH tracks two key metrics: SMACT, the percentage of Streaming Multiprocessors (SMs) reserved by an application, and SM occupancy (SMOCC), the percentage of SMs actively running kernels. If SMOCC is significantly lower than SMACT, it implies that the application is utilizing only a small portion of reserved SMs, suggesting inefficient use of GPU resources.

4.1 Running applications in isolation on GPU and CPU

We study the behavior and performance of each application running in isolation on the GPU and CPU. We use CONSUMERBENCH to collect the system-level metrics and application-level metrics for each application.

Fig. 3 shows results for our latency-sensitive applications; DeepResearch is excluded as it does not have an SLO.

Performance. When running on the GPU, all applications achieve 100% SLO attainment except for LiveCaptions, where only three out of 150 audio segments incur SLO violations. When running on the CPU, all applications experience reduced performance, though the impact varies. Chatbot narrowly misses its SLOs, while ImageGen and LiveCaptions suffer from significantly higher request latencies.

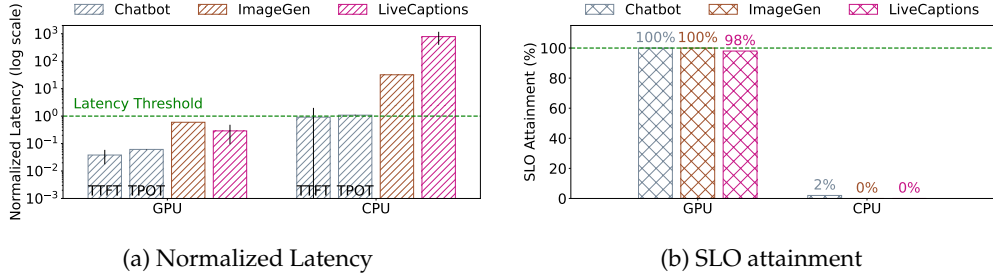


Figure 3: (a) Latencies normalized to SLO requirements and (b) SLO attainment for Chatbot, ImageGen, and LiveCaptions running exclusively on the GPU or CPU.

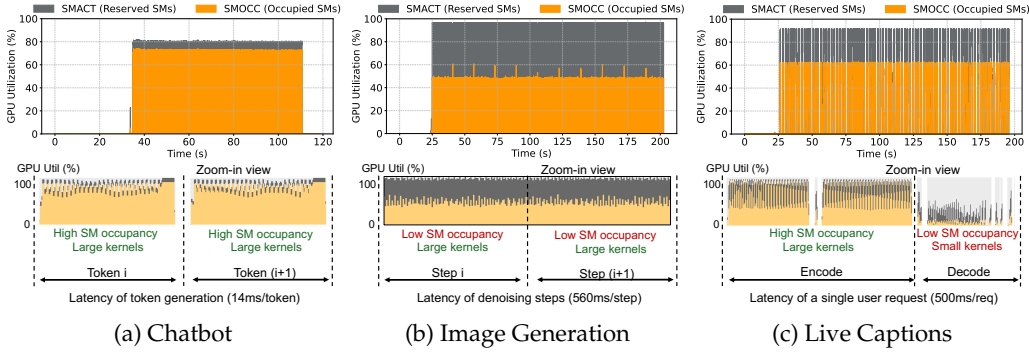


Figure 4: GPU utilization of each application running exclusively on the GPU.

System-level metrics. Fig. 4 shows the GPU utilization when applications run exclusively on GPU. All applications reserve almost all of the GPU cores when running exclusively, shown using SMACT. However, their efficiencies are very different, indicated by SMOCC. Chatbot utilizes its resources efficiently, while ImageGen and LiveCaptions under-utilize their reserved GPU cores.

Analysis of Individual Applications. We analyze the GPU kernels launched by each application to understand their performance and GPU utilization in Fig. 4.

- **Chatbot:** Spends majority of its time in decoding output tokens (*i.e.*, token generation). It achieves a high SMOCC (Fig. 4a) since llama.cpp customizes the thread block and grid dimensions for its kernels during prefill and decode according to the GPU architecture.
- **ImageGen:** Spends most of its time in the denoising phase, which uses an attention-based U-Net (Oktay et al., 2018). PyTorch’s attention kernel used by SD-3.5-Medium-Turbo requires over 150 registers per thread, limiting the number of threads that can run concurrently on each SM. This reduces SMOCC causing low GPU utilization (Fig. 4b).
- **LiveCaptions:** Uses the encoder-decoder Whisper-large-v3-turbo model. The encoder phase performs parallel operations on the input audio, involving softmax and large matrix multiplications that use tens of registers per thread with a high SMOCC. The decoder phase, however, spends most time in performing matrix multiplications with much smaller kernels than Chatbot or ImageGen. This phase also involves hundreds of registers per thread and high shared memory usage, largely due to inefficient kernel implementations. These factors contribute to the low SMOCC (Fig. 4c).

4.2 Challenges and Strategies for Concurrent Execution

We now evaluate concurrent GPU execution for applications studied in §4.1. CONSUMER-BENCH collects both system-level and application-level metrics across static and dynamic GPU sharing strategies whose primary goal is to maximize GPU utilization while satisfying

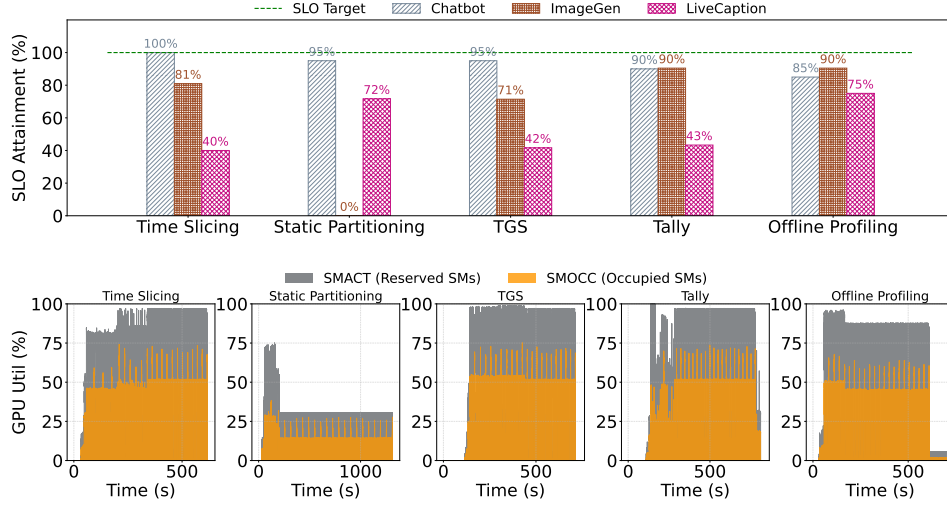


Figure 5: SLO attainment and GPU utilization under greedy allocation, static GPU partitioning, dynamic priority schedulers, and an oracle MPS upper bound.

per-application SLOs. We evaluate the following GPU sharing strategies. We choose per-application request counts such that all applications have approximately equal end-to-end latencies under default GPU time-slicing. This keeps every application active throughout the experiment and sustains a consistent fully contended period. Model-server setup, weight loading, and warm-up are excluded from request-latency and SLO accounting.

- **Time Slicing:** This is the default GPU time-slicing, where kernels from concurrent applications take turns occupying the full GPU in a first-come-first-serve (FCFS) manner.
- **Static GPU Partitioning:** This strategy uses NVIDIA MPS (Corporation, 2025c) to equally reserve GPU resources (33% of SMs) for each of the three latency-sensitive applications.
- **TGS (Wu et al., 2023):** A dynamic GPU sharing strategy that uses adaptive rate control at kernel-level granularity to prioritize high-priority tasks over best-effort tasks. We show the result for the best priority configuration found by brute force search.
- **Tally (Zhao et al., 2025):** A dynamic GPU sharing strategy performs fine-grained thread-block-level kernel scheduling to provide performance isolation for high-priority tasks over best-effort tasks. Similar to TGS, we show the result for the best priority configuration found by brute force search.
- **Offline MPS profiling:** In this strategy, we perform offline profiling for each application to find the minimum MPS value that just meets its SLO when running in isolation, and use that value during concurrent execution. Since each application’s allocation is chosen independently, the sum of allocations may exceed 100%.

Performance. Fig. 5 shows SLO attainment and GPU utilization across all five strategies. No strategy satisfies all application SLOs. Time slicing severely starves LiveCaptions, which attains its SLO for only 28% of requests. Static GPU partitioning improves fairness for LiveCaptions, but causes the SLO attainment for ImageGen to drop to 0%. TGS improves the SLO attainment of ImageGen to 71%, but starves LiveCaptions. Tally performs better than TGS, but also starves LiveCaptions. Offline profiling achieves the best overall trade-off (85%, 90%, and 75% for Chatbot, ImageGen, and LiveCaptions, respectively). All strategies except for static partitioning achieve high GPU utilization.

Analysis. Time-slicing and static GPU partitioning naively provision GPU resources, and achieve suboptimal performance. Dynamic GPU sharing policies (TGS and Tally) perform better, but still suffer from poor SLO attainment in one or more applications due to their coarse-grained priority settings for diverse SLOs. Offline profiling is able to achieve the best performance overall, due to SLO-focused resource allocation. This motivates the need for dynamic GPU sharing strategies that are aware of diverse application SLOs.

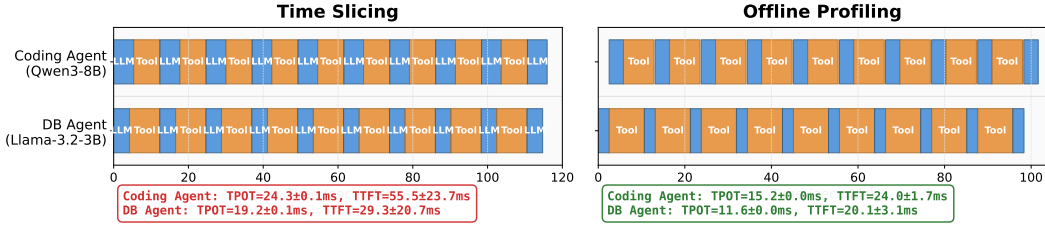


Figure 6: Latency breakdown of concurrent workflow of a Coding Agent and a Database Agent.

Step	Isolated	+ Chatbot
1	23.1s	78.2s
3	32.1s	62.3s
5	38.9s	79.9s
7	41.3s	87.8s
Total	270.0s	549.1s

Table 3: Background coding agent latency (s) running isolated or under bursty Chatbot load.

Application metric (SLO)	DAG	Opaque
DeepResearch E2E	594s	1649s
Brainstorm TPOT (0.25s)	0.50s	0.73s
Outline TPOT (0.25s)	0.30s	0.40s
ImageGen (1s)	0.67s	0.62s
LiveCaptions (2s)	19.0s	17.0s

Table 4: DAG-aware versus opaque execution of the content-creation workflow.

4.3 Multi-Agent Workflow

We evaluate concurrent agentic workflows that contain GenAI applications and external tools that collaborate to perform user tasks. These workflows expose opportunities and bottlenecks from multi-step dependencies and concurrent execution that are invisible to benchmarks treating applications as independent processes.

LLM calls and tool calls scheduling (Coding Agent + Database Agent). This workflow evaluates whether a scheduler can exploit the alternating LLM calls and tool calls effectively. The workflow contains two parallel agents: a Coding Agent running Qwen3-8B (Yang et al., 2025) with OStools and a Database Agent running Llama-3.2-3B (Dubey et al., 2024) with DBTools. Each agent executes a series of steps in which an LLM call is followed by a tool call. Fig. 6 shows the end-to-end latency breakdown under Time Slicing and Offline Profiling. With Time Slicing, overlapping LLM calls contend for the GPU, while the GPU sits idle during tool-call phases. With Offline Profiling, we avoid concurrent LLM calls and instead overlap each agent’s LLM call with the other’s tool call, reducing end-to-end latency from 118s to 102s and improving both TPOT and TTFT.

Long-running background agent with bursty agent (Coding Agent + Chatbot). This workflow evaluates whether a stateful background agent remains responsive when latency-sensitive requests arrive. We pair a background coding agent with a bursty high-priority Chatbot sharing the Qwen3-8B (Yang et al., 2025) model launched using the vLLM inference server on RTX 6000 GPU with 24GB VRAM. The inference server retains an approximately 50K-token KV cache for the coding agent. When bursty Chatbot requests arrive, vLLM preempts the coding agent and discards this KV cache, forcing later steps to repeat a long prefill. As Table 3 shows, contention increases total latency of the coding agent from 270.0s to 549.1s (2.03×).

Workflow preserving semantic dependencies (Content-creation Workflow). This workflow evaluates whether explicit workflow dependencies expose the correct system bottleneck. The workflow simulates a content creation workflow using a five-stage DAG, which runs a background DeepResearch analysis alongside Chatbot brainstorming, followed by outline generation that waits for both, after which ImageGen creates cover art and LiveCaptions generates captions (see Fig. 18 for the workflow definition). We run this workflow with its inter-application dependencies modeled correctly and compare with a dependency-unaware execution, where the same five applications are launched as opaque concurrent processes. As Table 4 shows, the opaque run is semantically invalid and changes the bottleneck: three LLM clients (DeepResearch and two Chatbot instances) contend on the CPU-side

attention of llama.cpp since CPU offloading is enabled. This causes all three LLM clients to degrade much more severely compared to the DAG execution that only runs two LLM clients concurrently, inflating DeepResearch latency from 594s to 1649s despite lower GPU occupancy. On the other hand, the workflow DAG preserves the intended ordering and exposes the realistic bottleneck.

5 Design Implications from CONSUMERBENCH

CONSUMERBENCH reveals and quantifies novel bottlenecks that affect end-user GenAI workflows, yielding concrete design targets.

Supporting SLO-aware dynamic GPU management. Static GPU sharing techniques (static GPU partitioning, offline MPS profiling) are not able to cater to changing user requests. On the other hand, dynamic GPU sharing techniques (*e.g.*, TGS, Tally) are not SLO-aware, and only support coarse-grained priority management, leading to starvation of applications when run concurrently (§4.2 and Fig. 5).

Scheduling applications based on resource requirements. Existing GPU sharing solutions do not consider resource utilization when executing workflows such as multi-agent pipelines. This leads to either resource contention, when multiple agents try to access the GPU simultaneously, violating SLOs, or periods of execution when the GPU is underutilized due to tool calling, causing resource waste (§4.3 and Fig. 6).

Dynamically configuring inference servers based on application SLOs. Inference servers are configured at boot time using parameters that determine placement of model layers, and KV cache, independent of GenAI applications. When applications with conflicting SLOs share a single inference server (for example, a Chatbot and a background DeepResearch agent sharing a single LLM), the static configurations cause the applications to violate their performance goals (§4.3, Table 3, and Fig. 15).

Having concurrency-aware kernel development. Today, GenAI models are designed and implemented assuming that they have dedicated, exclusive access of the GPU. This results in implementations that increase resource contention when run concurrently with other GenAI models, and results in SLO violations (for example, ImageGen when executed concurrently with LiveCaptions causes severe resource contention). GPU kernel libraries should be developed in a manner that allows the GPU scheduler to achieve both high GPU utilization and fairness in resource allocations across all running applications (Sections 4.1 and 4.2 and Figures 4 and 5).

6 Limitations

CONSUMERBENCH assumes local model deployment and does not cover edge-cloud collaborative speculative decoding (Hao et al., 2024; Oh et al., 2025). Its current implementation targets laptops and servers rather than more constrained mobile devices or sensors.

We omit Orion and REEF (Strati et al., 2024; Han et al., 2022), which do not support modern GenAI stacks, and LithOS (Coppock et al., 2025), which is not open source.

7 Conclusion

This paper presents CONSUMERBENCH, a comprehensive benchmarking framework for evaluating GenAI applications on end-user devices. It enables users to define realistic workflows and monitors application-level metrics and system performance under concurrent execution. CONSUMERBENCH reveals inefficiencies in GPU sharing and inference servers, offering insights to develop systems tailored to end-user devices.

8 Ethics Statement

To the best of our knowledge, CONSUMERBENCH does not raise questions regarding the Code of Ethics.

9 Reproducibility Statement

The implementation of CONSUMERBENCH, workflow configurations, and instructions for reproducing the paper’s results are available at <https://github.com/efeslab/ConsumerBench>. An archival snapshot is preserved at <https://doi.org/10.5281/zenodo.19364966>.

References

- Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.
- Estelle Aflalo, Zhuo Wu, and Tiep Le. Use ai on your pc: Optimize and deploy a multimodal agentic pipeline on ai pc powered by intel, September 2025. URL <https://huggingface.co/blog/estellea/use-ai-on-your-pc>.
- Stability AI. Stable diffusion 3 medium. <https://huggingface.co/stabilityai/stable-diffusion-3-medium>, 2026. Accessed: 2026-03-31.
- Android Developers. Gemini nano with the google ai edge sdk, 2025. URL <https://developer.android.com/ai/gemini-nano>.
- Apple. Metal performance shaders. <https://developer.apple.com/documentation/metalperformanceshaders>, 2026. Accessed: 2026-03-31.
- Apple Inc. Introducing apple intelligence, the personal intelligence system that puts powerful generative models at the core of iphone, ipad, and mac, 2024. URL <https://www.apple.com/newsroom/2024/06/introducing-apple-intelligence-for-iphone-ipad-and-mac/>.
- AUTOMATIC1111. Stable diffusion web ui, 2022. URL <https://github.com/AUTOMATIC1111/stable-diffusion-webui>. Accessed: 2025-05-11.
- BerriAI. Litellm: Python sdk and proxy server for unified llm access, 2024. URL <https://github.com/BerriAI/litellm>. Accessed: 2025-05-11.
- Abhinaba Chakraborty, Wouter Tavernier, Akis Kourtis, Mario Pickavet, Andreas Oikonomakis, and Didier Colle. Profiling concurrent vision inference workloads on nvidia jetson–extended. *arXiv preprint arXiv:2508.08430*, 2025.
- Hongtao Chen, Weiyu Xie, Boxin Zhang, Jingqi Tang, Jiahao Wang, Jianwei Dong, Shaoyuan Chen, Ziwei Yuan, Chen Lin, Chengyu Qiu, et al. Ktransformers: Unleashing the full potential of cpu/gpu hybrid inference for moe models. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pp. 1014–1029, 2025a.
- Junsong Chen, Shuchen Xue, Yuyang Zhao, Jincheng Yu, Sayak Paul, Junyu Chen, Han Cai, Song Han, and Enze Xie. Sana-sprint: One-step diffusion with continuous-time consistency distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 16185–16195, 2025b.
- Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollar, and C. Lawrence Zitnick. Microsoft coco captions: Data collection and evaluation server, 2015. URL <https://arxiv.org/abs/1504.00325>.
- CompVis. Stable diffusion v1-4 model card. <https://huggingface.co/CompVis/stable-diffusion-v1-4>, 2026. Accessed: 2026-03-31.

- Patrick H. Coppock, Brian Zhang, Eliot H. Solomon, Vasilis Kypriotis, Leon Yang, Bikash Sharma, Dan Schatzberg, Todd C. Mowry, and Dimitrios Skarlatos. Lithos: An operating system for efficient machine learning on gpus. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles, SOSP '25*, pp. 1–17, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400718700. doi: 10.1145/3731569.3764818. URL <https://doi.org/10.1145/3731569.3764818>.
- Intel Corporation. Intel[®] performance counter monitor. <https://github.com/intel/pcm>, 2025a. Accessed: 2025-05-12.
- NVIDIA Corporation. Nvidia data center gpu manager (dcmg). <https://github.com/NVIDIA/DCGM>, 2025b. Accessed: 2025-05-12.
- NVIDIA Corporation. Nvidia multi-process service (mps). <https://docs.nvidia.com/deploy/mps/index.html>, 2025c. Accessed: 2025-05-12.
- DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024.
- Miguel Del Rio, Natalie Delworth, Ryan Westerman, Michelle Huang, Nishchal Bhandari, Joseph Palakapilly, Quinten McNamara, Joshua Dong, Piotr Zelasko, and Miguel Jetté. Earnings-21: A practical benchmark for asr in the wild. *arXiv preprint arXiv:2104.11348*, 2021.
- Shihan Deng, Weikai Xu, Hongda Sun, Wei Liu, Tao Tan, Jianfeng Liu, Ang Li, Jian Luan, Bin Wang, Rui Yan, and Shuo Shang. Mobile-bench: An evaluation benchmark for llm-based mobile agents, 2024. URL <https://arxiv.org/abs/2407.00993>.
- Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023. URL <https://arxiv.org/abs/2305.14314>.
- Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *IEEE Transactions on Big Data*, 2025.
- Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. Deepresearch bench: A comprehensive benchmark for deep research agents. In *International Conference on Learning Representations*, 2026.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- Fireworks AI. Understanding function calling: The bridge to agentic ai, July 2025. URL <https://fireworks.ai/blog/function-calling>.
- Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL <https://arxiv.org/abs/2210.17323>.
- Georgi Gerganov and ggml-org contributors. llama.cpp: Llm inference in c/c++, 2023. URL <https://github.com/ggml-org/llama.cpp>. Accessed: 2025-05-11.
- Ruihao Gong, Shihao Bai, Siyu Wu, Yunqian Fan, Zaijun Wang, Xiuhong Li, Hailong Yang, and Xianglong Liu. Past-future scheduler for llm serving under sla guarantees. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 798–813, 2025.
- Google. Gemma model card. <https://huggingface.co/google/gemma-2b>, 2026. Accessed: 2026-03-31.
- Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. Serving {DNNs} like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 443–462, 2020.

- Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 539–558, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL <https://www.usenix.org/conference/osdi22/presentation/han>.
- Awni Hannun, Jagrit Digani, Angelos Katharopoulos, and Ronan Collobert. MLX: Efficient and flexible machine learning on apple silicon, 2023. URL <https://github.com/ml-explore>.
- Zixu Hao, Huiqiang Jiang, Shiqi Jiang, Ju Ren, and Ting Cao. Hybrid slm and llm for edge-cloud collaborative inference. In *Proceedings of the Workshop on Edge and Mobile Foundation Models, EdgeFM '24*, pp. 36–41, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706639. doi: 10.1145/3662006.3662067. URL <https://doi.org/10.1145/3662006.3662067>.
- Intel Corporation. Running average power limit (rapl) energy reporting. <https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html>, 2022. Accessed: 2025-05-14.
- Rakshith Jayanth, Neelesh Gupta, and Viktor Prasanna. Benchmarking edge ai platforms for high-performance ml inference, 2024. URL <https://arxiv.org/abs/2409.14803>.
- Rachid Karami, Rajeev Patwari, Hyounjun Kwon, and Ashish Sirasao. Exploring the dynamic scheduling space of real-time generative ai applications on emerging heterogeneous systems. *arXiv preprint arXiv:2507.14715*, 2025.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.
- Stefanos Laskaridis, Kleomenis Katevas, Lorenzo Minto, and Hamed Haddadi. Melting point: Mobile evaluation of language transformers, 2024. URL <https://arxiv.org/abs/2403.12844>.
- Yanyu Li, Huan Wang, Qing Jin, Ju Hu, Pavlo Chemerys, Yun Fu, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Snapfusion: Text-to-image diffusion model on mobile devices within two seconds. *Advances in Neural Information Processing Systems*, 36:20662–20678, 2023a.
- Yilong Li, Jingyu Liu, Hao Zhang, M Badri Narayanan, Utkarsh Sharma, Shuai Zhang, Pan Hu, Yijing Zeng, Jayaram Raghuram, and Suman Banerjee. Palmbench: A comprehensive benchmark of compressed large language models on mobile platforms, 2025. URL <https://arxiv.org/abs/2410.05315>.
- Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E Gonzalez, et al. {AlpaServe}: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pp. 663–679, 2023b.
- Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2024. URL <https://arxiv.org/abs/2306.00978>.
- Jiachen Liu, Jae-Won Chung, Zhiyu Wu, Fan Lai, Myungjin Lee, and Mosharaf Chowdhury. Andes: Defining and enhancing quality-of-experience in llm-based text streaming services. *arXiv preprint arXiv:2404.16283*, 2024.

- Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.
- Dominik Macháček, Raj Dabre, and Ondřej Bojar. Turning whisper into real-time transcription system. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics: System Demonstrations*, pp. 17–24, 2023.
- Meta AI. Llama 3.2 3b: Multilingual large language model. <https://huggingface.co/meta-llama/Llama-3.2-3B>, 2024. Accessed: 2025-05-11.
- Rithesh Murthy, Liangwei Yang, Juntao Tan, Tulika Manoj Awalganekar, Yilun Zhou, Shelby Heinecke, Sachin Desai, Jason Wu, Ran Xu, Sarah Tan, Jianguo Zhang, Zhiwei Liu, Shirley Kokane, Zuxin Liu, Ming Zhu, Huan Wang, Caiming Xiong, and Silvio Savarese. Mobileaibench: Benchmarking llms and lmms for on-device use cases, 2024. URL <https://arxiv.org/abs/2406.10290>.
- NVIDIA. Green contexts - cuda driver api. https://docs.nvidia.com/cuda/cuda-driver-api/group__CUDA__GREEN__CONTEXTS.html, 2025a. Accessed: 2025-04-30.
- NVIDIA. Nvidia multi-instance gpu (mig). <https://www.nvidia.com/en-us/technologies/multi-instance-gpu/>, 2025b. Accessed: 2025-05-15.
- NVIDIA. Data center gpu manager (dcgm). <https://developer.nvidia.com/dcgm>, 2026. Accessed: 2026-03-31.
- NVIDIA Corporation. Nvidia management library (nvml). <https://developer.nvidia.com/management-library-nvml>, 2025. Accessed: 2025-05-14.
- Seungeun Oh, Jinhyuk Kim, Jihong Park, Seung-Woo Ko, Tony Q. S. Quek, and Seong-Lyun Kim. Uncertainty-aware hybrid inference with on-device small and remote large language models, 2025. URL <https://arxiv.org/abs/2412.12687>.
- Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention u-net: Learning where to look for the pancreas. In *Medical Imaging with Deep Learning*, 2018. URL <https://openreview.net/forum?id=Skft7cijM>.
- OpenAI. Openai api, 2020. URL <https://openai.com/index/openai-api/>. Accessed: 2025-05-11.
- OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, Kai Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman, Dominik Kundel, Jason Kwon, Volodymyr Kyrylov, Elaine Ya Le, Guillaume Leclerc, James Park Lennon, Scott Lessans, Mario Lezcano-Casado, Yanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily, Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, Dmitry Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo Raso, Hongyu Ren, Kimmy Richardson, David Robinson, Bob Rotsted, Hadi Salman, Suvansh Sanjeev, Max Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singhal, Yang Song, Dane Stuckey, Zhiqing Sun, Philippe Tillet, Sam Toizer, Foivos

- Tsimpourlas, Nikhil Vyas, Eric Wallace, Xin Wang, Miles Wang, Olivia Watkins, Kevin Weil, Amy Wendling, Kevin Whinnery, Cedric Whitney, Hannah Wong, Lin Yang, Yu Yang, Michihiro Yasunaga, Kristen Ying, Wojciech Zaremba, Wenting Zhan, Cyril Zhang, Brian Zhang, Eddie Zhang, and Shengjia Zhao. gpt-oss-120b & gpt-oss-20b model card, 2025. URL <https://arxiv.org/abs/2508.10925>.
- Qualcomm. Unlocking on-device generative ai with an npu and heterogeneous computing, February 2024. URL <https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/Unlocking-on-device-generative-AI-with-an-NPU-and-heterogeneous-computing.pdf>.
- Qualcomm Technologies, Inc. Snapdragon x elite product brief. <https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/Product-Brief-Snapdragon-X-Elite.pdf>, October 2023.
- Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision, 2022. URL <https://arxiv.org/abs/2212.04356>.
- Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. Mlperf inference benchmark, 2020. URL <https://arxiv.org/abs/1911.02549>.
- Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. ‘smolagents’: a smol library to build great agentic systems. <https://github.com/huggingface/smolagents>, 2025.
- Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E Gonzalez, and Ion Stoica. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 965–988, 2024.
- SS64. powermetrics - mac os x command. <https://ss64.com/mac/powermetrics.html>, 2025. Accessed: 2025-05-22.
- Foteini Strati, Xianzhe Ma, and Ana Klimovic. Orion: Interference-aware, fine-grained gpu sharing for ml applications. In *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys ’24*, pp. 1075–1092, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704376. doi: 10.1145/3627703.3629578. URL <https://doi.org/10.1145/3627703.3629578>.
- TensorArt Studios. Stable diffusion 3.5 medium turbo, 2024. URL <https://huggingface.co/tensorart/stable-diffusion-3.5-medium-turbo>. Accessed: 2025-05-11.
- SYSTRAN. Faster whisper transcription with ctranslate2. <https://github.com/SYSTRAN/faster-whisper>, 2026. Accessed: 2026-03-31.
- Kandinsky Team. Kandinsky 2.2. <https://huggingface.co/kandinsky-community/kandinsky-2-2-decoder>, 2026a. Accessed: 2026-03-31.
- Qwen Team. Qwen2.5-7b. <https://huggingface.co/Qwen/Qwen2.5-7B>, 2026b. Accessed: 2026-03-31.
- Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, Dhruv Nair, Sayak Paul, William Berman, Yiyi Xu, Steven Liu, and Thomas Wolf. Diffusers: State-of-the-art diffusion models. <https://github.com/huggingface/diffusers>, 2022.

- Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. Minilmv2: Multi-head self-attention relation distillation for compressing pretrained transformers. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 2140–2151, 2021.
- Bingyang Wu, Zili Zhang, Zhihao Bai, Xuanzhe Liu, and Xin Jin. Transparent GPU sharing in container clouds for deep learning workloads. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pp. 69–85, Boston, MA, April 2023. USENIX Association. ISBN 978-1-939133-33-5. URL <https://www.usenix.org/conference/nsdi23/presentation/wu>.
- Jie Xiao, Qianyi Huang, Xu Chen, and Chen Tian. Large language model performance benchmarking on mobile platforms: A thorough evaluation. *arXiv preprint arXiv:2410.03613*, 2024.
- Fei Xu, Jianian Xu, Jiabin Chen, Li Chen, Ruitao Shang, Zhi Zhou, and Fangming Liu. igniter: Interference-aware gpu resource provisioning for predictable dnn inference in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 34(3):812–827, 2022.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX symposium on operating systems design and implementation (OSDI 22)*, pp. 521–538, 2022.
- Minchen Yu, Ao Wang, Dong Chen, Haoxuan Yu, Xiaonan Luo, Zhuohao Li, Wei Wang, Ruichuan Chen, Dapeng Nie, and Haoran Yang. Faaswap: Slo-aware, gpu-efficient serverless inference via model swapping, 2024. URL <https://arxiv.org/abs/2306.03622>.
- Minchen Yu, Ao Wang, Dong Chen, Haoxuan Yu, Xiaonan Luo, Zhuohao Li, Wei Wang, Ruichuan Chen, Dapeng Nie, Haoran Yang, et al. Torpor:{GPU-Enabled} serverless computing for {Low-Latency},{Resource-Efficient} inference. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pp. 597–612, 2025.
- Wei Zhao, Anand Jayarajan, and Gennady Pekhimenko. Tally: Non-intrusive performance isolation for concurrent deep learning workloads. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, ASPLOS ’25*, pp. 1052–1068, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400706981. doi: 10.1145/3669940.3707282. URL <https://doi.org/10.1145/3669940.3707282>.
- Yang Zhao, Yanwu Xu, Zhisheng Xiao, Haolin Jia, and Tingbo Hou. Mobilediffusion: Instant text-to-image generation on mobile devices, 2024. URL <https://arxiv.org/abs/2311.16567>.
- Zhihe Zhao, Neiwen Ling, Nan Guan, and Guoliang Xing. Miriam: Exploiting elastic kernels for real-time multi-dnn inference on edge gpu. In *Proceedings of the 21st ACM Conference on Embedded Networked Sensor Systems*, pp. 97–110, 2023.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric Xing, et al. Lmsys-chat-1m: A large-scale real-world llm conversation dataset. In *International Conference on Learning Representations*, volume 2024, pp. 22225–22257, 2024.

Appendices

A Measurement and Reproducibility Details

A.1 Metrics and Instrumentation

System-level metrics. On the Intel and NVIDIA platform, we use NVIDIA’s DCGM and NVML interfaces (NVIDIA, 2026; NVIDIA Corporation, 2025) to collect GPU telemetry, including GPU compute utilization, memory bandwidth utilization, GPU memory usage, and GPU power. For CPU-side telemetry (used in figures that report CPU utilization, DRAM bandwidth, and system power), we rely on standard Linux interfaces (e.g., /proc and performance counters) and package-level energy reporting, such as Running Average Power Limit (RAPL), when available. RAPL (Intel Corporation, 2022) is an energy/power reporting and control interface provided by Intel CPUs that allows monitoring of energy consumption at the CPU package level, enabling fine-grained measurement of processor and DRAM power usage.

On Apple Silicon devices, we use the Mac’s powermetrics tool (SS64, 2025) to monitor GPU activity and power consumption. Because powermetrics exposes a more limited set of GPU performance counters than the NVIDIA monitoring tools used on our server platform, we do not report additional metrics such as memory bandwidth utilization for Apple Silicon. Importantly, Apple and NVIDIA metrics are not directly comparable in absolute terms: powermetrics (SS64, 2025) reports Apple-specific accelerator activity and power metrics and provides limited visibility into memory-subsystem counters, while DCGM and NVML expose utilization and bandwidth counters specific to NVIDIA GPUs. We therefore interpret cross-platform results primarily through within-platform trends rather than one-to-one counter equivalence.

A.2 Platform-Specific Implementation Notes

Application Optimizations. Our GenAI optimizations for Apple Silicon include:

- **Chatbot & DeepResearch.** We configured the llama.cpp server with the `-metal` flag to optimize GPU kernel execution for Apple Silicon’s ARM-based architecture. The Llama-3.2-3B model runs efficiently through this Metal-accelerated backend, handling both conversational and research workloads.
- **ImageGen.** Our local image generation server uses the MPS (Metal Performance Shaders) backend (Apple, 2026), mirroring llama.cpp’s Apple Silicon optimizations. We employ the SD-v1-4 model (CompVis, 2026) instead of the NVIDIA-optimized SD-3.5-medium-turbo variant (Studios, 2024), as it demonstrates better performance on Apple’s unified memory architecture.
- **LiveCaptions.** The MLX framework (Hannun et al., 2023) accelerates our Whisper-Large-v3-turbo (Radford et al., 2022) implementation for real-time audio processing. While maintaining model parity with our default server, we adjusted the service-level objective (SLO) to 4 seconds (from 2 seconds) and modified audio chunking intervals to accommodate Apple Silicon’s higher transcription latency.

B Extended Telemetry for Main-Paper Experiments

B.1 Exclusive Execution Telemetry

B.1.1 System-level Metrics for Applications Running Exclusively on GPU

Fig. 7 shows GPU utilization, memory bandwidth, memory utilization, and power consumption when running latency-sensitive applications exclusively on the GPU, as reported by CONSUMERBENCH. These are augmented results for Section 4.1 in the paper. Overall, Chatbot consumes the most GPU bandwidth, while ImageGen requires the most GPU

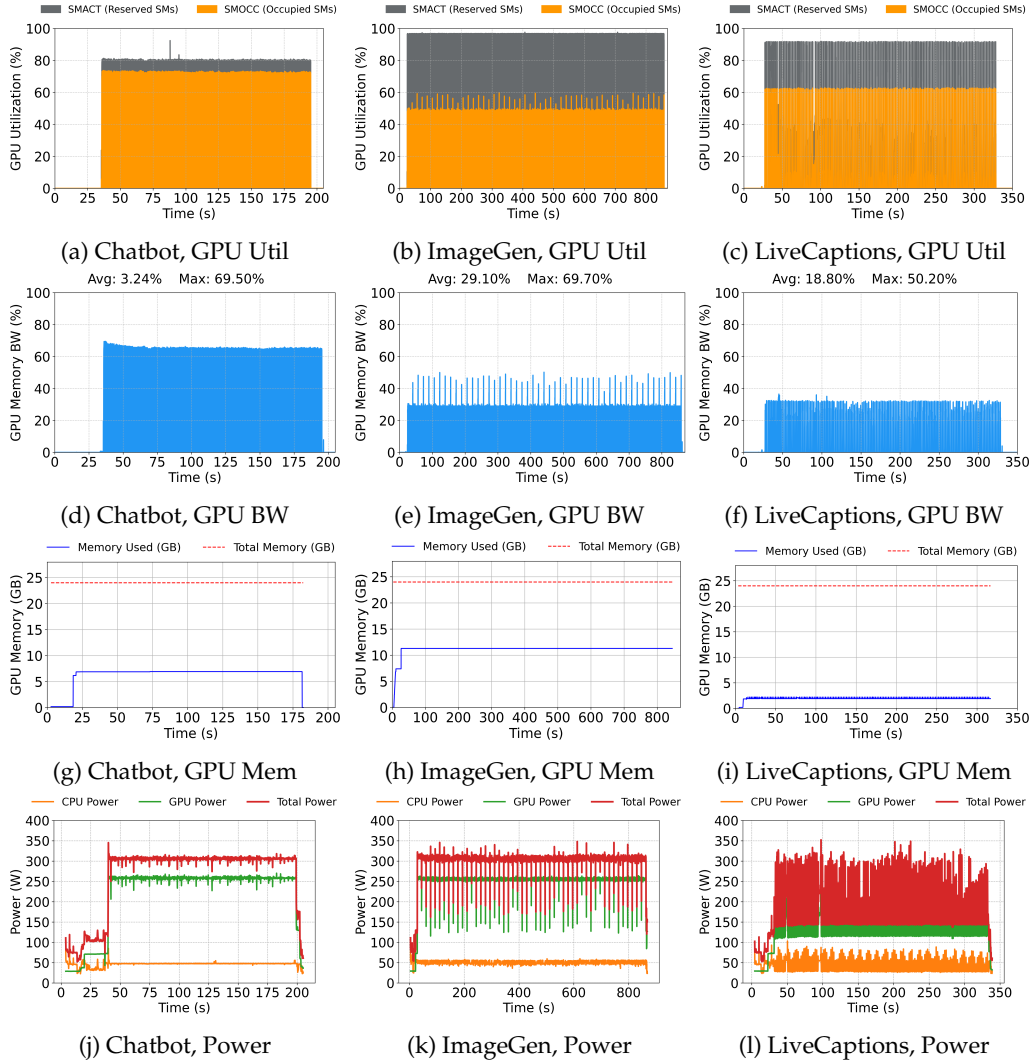


Figure 7: Running applications exclusively on the GPU.

memory. All applications have similar peak power consumption despite differences in GPU utilization.

B.1.2 System-level Metrics for Applications Running Exclusively on CPU

Fig. 8 shows resource utilization when running latency-sensitive applications exclusively on the CPU. These are augmented results for Section 4.1 in the paper. The applications are bottlenecked by compute rather than memory, as evidenced by high CPU utilization and low memory bandwidth consumption. Finally, applications require significantly less power when executing on the CPU compared to the GPU.

B.1.3 System-level Metrics for Running Chatbot with GPT-OSS-20B

Fig. 9 and Fig. 10 show the performance of running Chatbot with GPT-OSS-20B (OpenAI et al., 2025) exclusively on the GPU and CPU. Similar to running Chatbot with Llama-3.2-3B (Meta AI, 2024) (see Fig. 7a), SMOCC for GPU utilization is high due to efficient kernel implementation in llama.cpp. When running on CPU exclusively, Chatbot shows high CPU utilization and low CPU bandwidth utilization, similar to findings in Fig. 8.

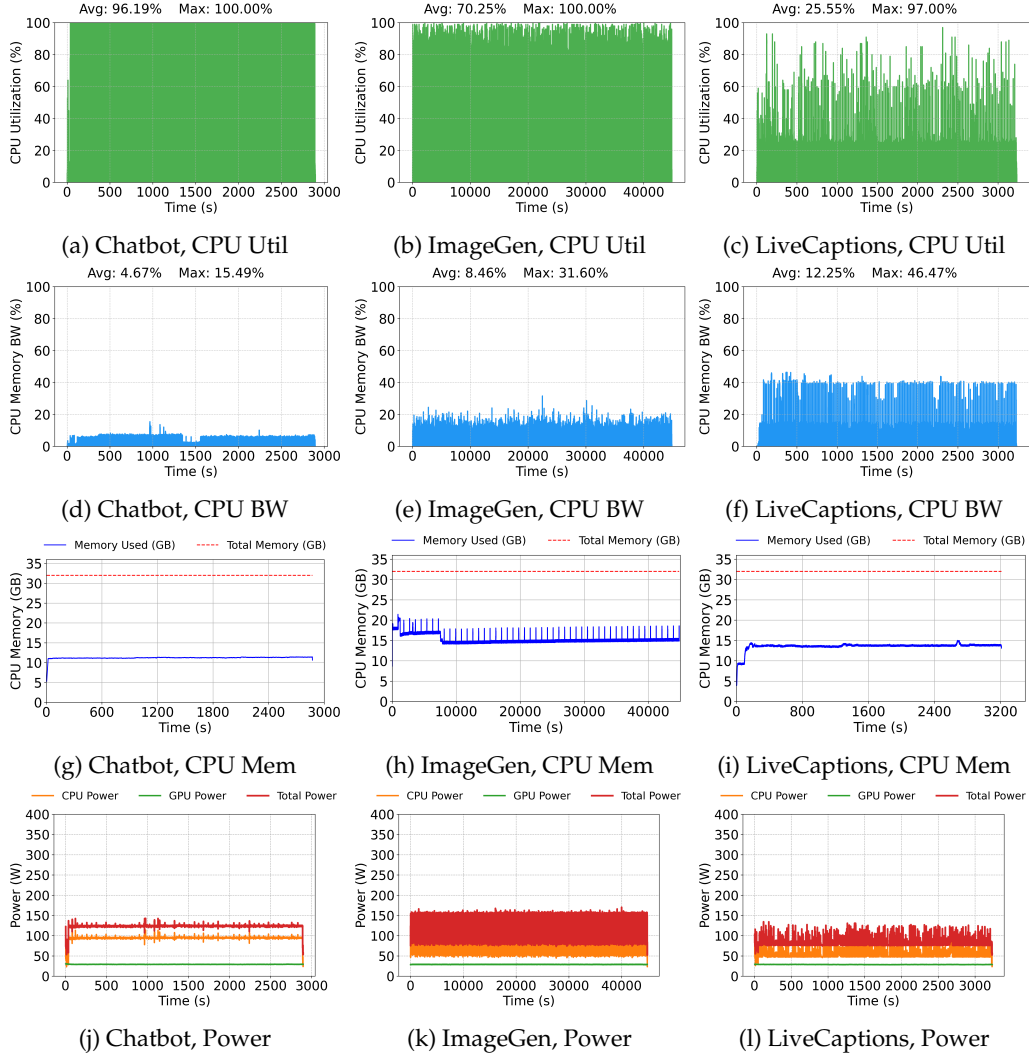


Figure 8: Running applications exclusively on the CPU.

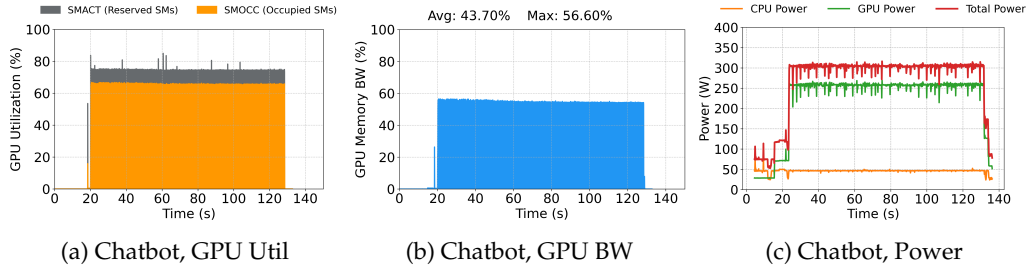


Figure 9: Running Chatbot with GPT-OSS-20B exclusively on the GPU

B.2 Concurrent Execution Telemetry

Fig. 11 shows the GPU metrics and power consumption when running Chatbot, ImageGen, and LiveCaptions concurrently on the GPU. These are augmented results for Section 4.2 in the paper. Greedy resource allocation consumes more power on average compared to static GPU partitioning. This directly follows from the fact that static GPU partitioning underutilizes the GPU, as shown in Fig. 11d.

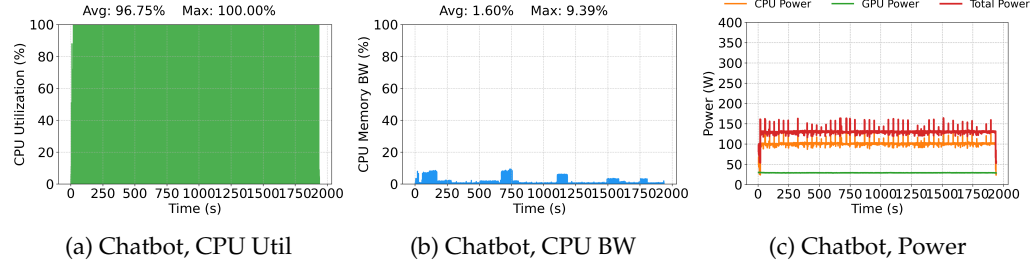


Figure 10: Running Chatbot with GPT-OSS-20B exclusively on the CPU

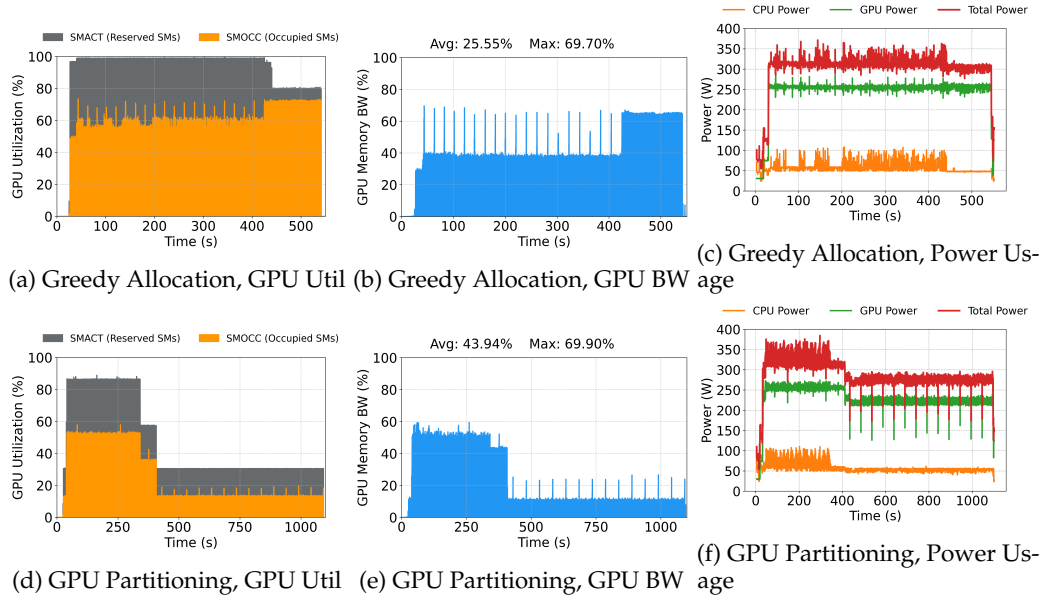


Figure 11: Running applications concurrently on the GPU.

B.2.1 SM-frequency throttling and workload switching

To separate frequency throttling from cross-workload interference, we instrument the RTX 6000’s achieved SM clock and DVFS throttle reasons and repeat the exclusive runs from §4.1 and the concurrent time-slicing run from §4.2. Fig. 12 shows the clock traces, and Table 5 summarizes the observed throttling and SLO outcomes.

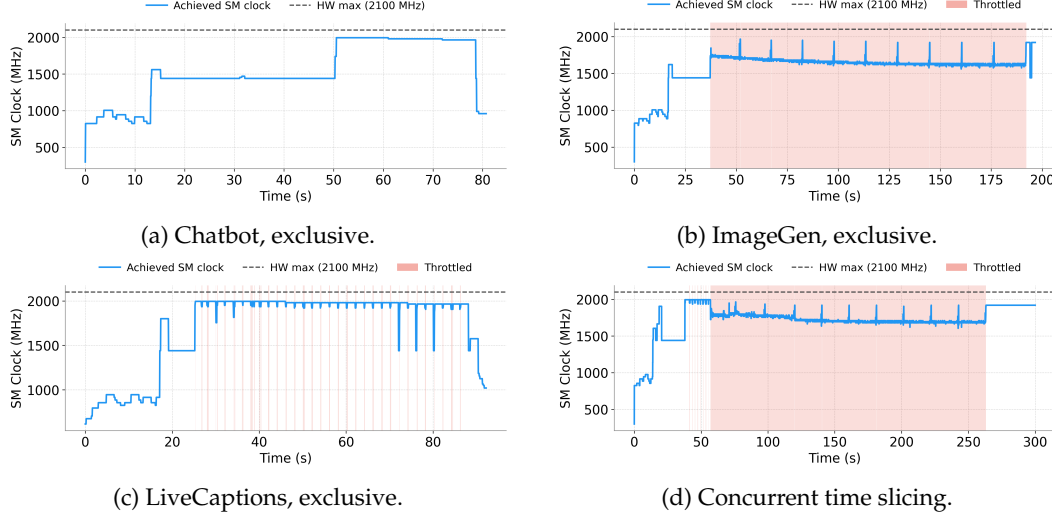


Figure 12: Achieved SM clock on the RTX 6000 during exclusive and concurrent execution. Shaded intervals indicate an active software power-cap throttle reason.

Run	Observed throttling	Throttle reason	SLO outcome
Chatbot, exclusive	None (0%)	None	Meets
ImageGen, exclusive	Most of run	SW power cap	Meets
LiveCaptions, exclusive	Brief, intermittent	SW power cap	Meets
Concurrent time slicing	Most contended period	SW power cap	LiveCaptions misses

Table 5: Frequency-throttling behavior and application SLO outcomes.

Frequency throttling alone does not explain the SLO misses. ImageGen is power-capped for most of its exclusive run because its large compute-bound kernels saturate the power budget, yet it meets its per-step SLO. The concurrent run exhibits the same throttle reason, while ImageGen generally remains within its SLO and LiveCaptions is starved. Thus, these measurements do not reveal an additional concurrency-specific DVFS penalty that accounts for the performance difference.

Instead, the dominant effect is workload interference at the GPU scheduler. Under time slicing, ImageGen’s large kernels delay LiveCaptions’ small decode kernels; switching among workloads also prevents each application from sustaining its isolation throughput. The lower effective SM occupancy in the concurrent telemetry (Fig. 11) and the resulting LiveCaptions slowdown therefore reflect kernel-level contention and scheduling gaps rather than frequency throttling. Offline-profiled partitioning mitigates this effect by reserving GPU capacity for the latency-sensitive workload.

B.3 Inference-Server and Static Model-Sharing Telemetry

Fig. 13 and Fig. 14 show the GPU metrics, CPU metrics, and power usage of running Chatbot with static model sharing via llama.cpp.

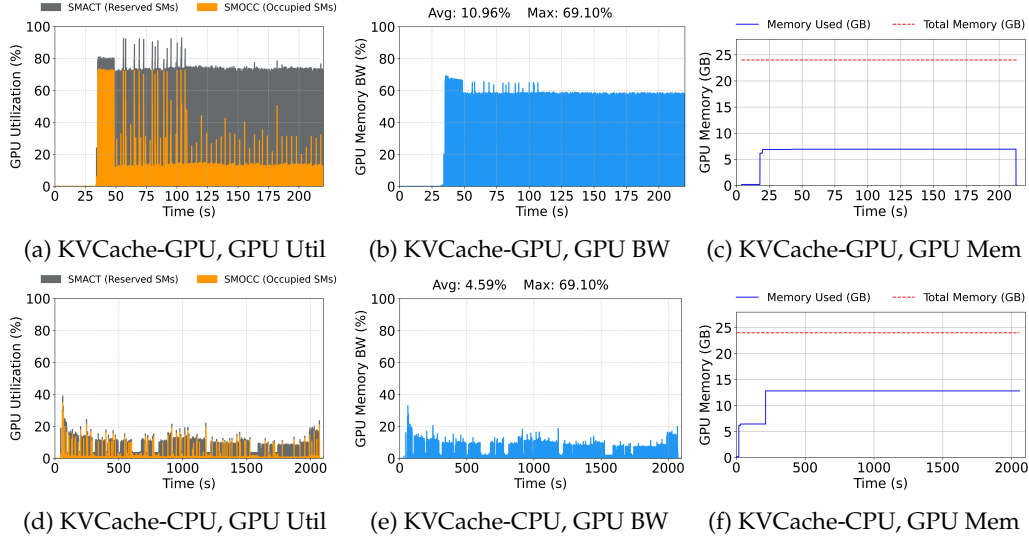


Figure 13: GPU metrics of running applications with static model sharing via inference servers.

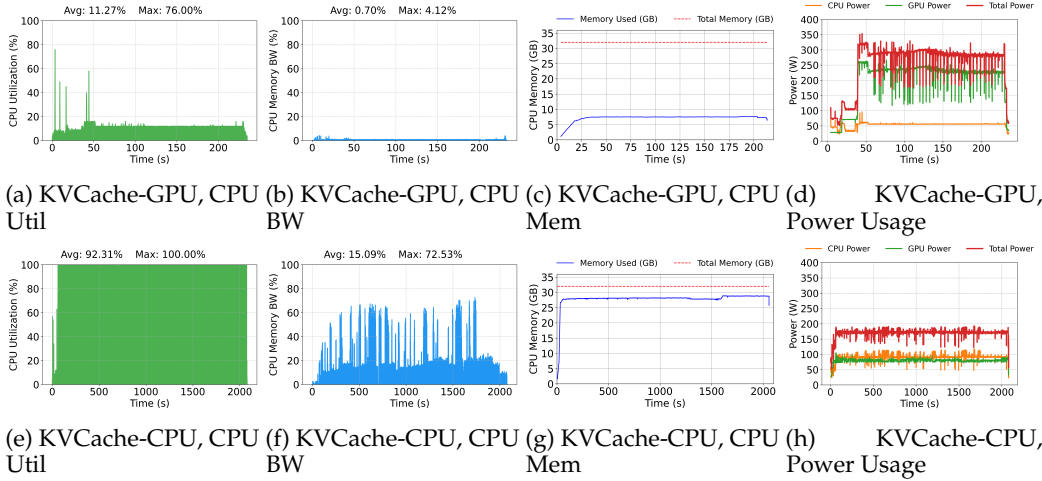


Figure 14: CPU metrics and power usage of running applications with static model sharing via inference servers.

Chatbot-KVCache-CPU overall consumes lower GPU resources, but is bottlenecked on the CPU compute due to attention operations on the CPU, evident from the high CPU utilization. Furthermore, performing CPU computations leads to lower power consumption for Chatbot-KVCache-CPU compared to Chatbot.

B.4 Hybrid execution on GPU and CPU

Given the limited GPU memory on end-user devices, applications can share a single foundation model across multiple GenAI applications with similar input-output modalities, with the help of an inference server. However, these inference servers are often statically configured, and cannot accommodate different performance needs of requests from different applications.

We use the locally deployed llama.cpp inference server with the Llama-3.2-3B model to serve requests for Chatbot (latency-sensitive task) and DeepResearch (background task). To support DeepResearch’s requirement for a large context window, we configure llama.cpp with a 16GB KV cache, matching the model’s 128K token context window. To conserve GPU

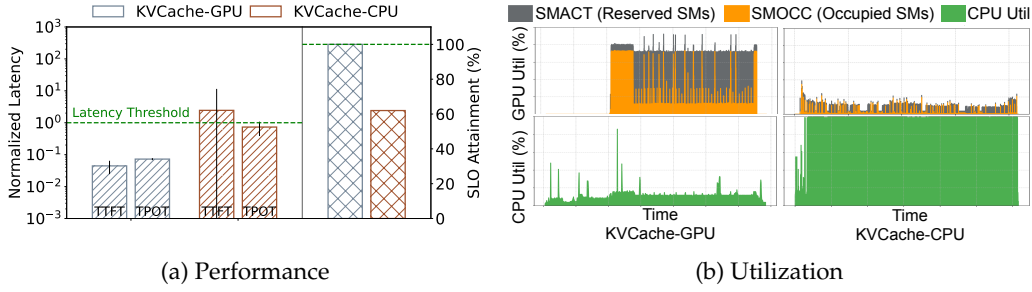


Figure 15: Comparison of Chatbot performance and resource utilization with GPU vs. CPU KV cache.

memory, we launch llama.cpp with the `-no-kv-offload` option that stores the KV cache in CPU memory (referred to as Chatbot-KVCache-CPU). In contrast, running the default Chatbot configuration with a smaller KV cache allows both applications to execute concurrently on the GPU, but forces DeepResearch to use a smaller context window, resulting in degraded output quality. Fig. 15 shows the GPU/CPU utilization and latency for both Chatbot and Chatbot-KVCache-CPU when running alongside DeepResearch.

Performance & system-level metrics. Compared to Chatbot, Chatbot-KVCache-CPU exhibits high variance in results and misses its SLO for approximately 40% of its requests. To avoid the overhead of loading the KV cache into the GPU, Chatbot-KVCache-CPU performs attention operations on the CPU, leading to high CPU utilization and correspondingly low GPU utilization.

Analysis. Inference servers use static configurations for shared models, which cannot accommodate different performance needs of multiple applications. Configuring a large KV cache for DeepResearch negatively impacts the latency of Chatbot. Hence, naively sharing models via inference servers leads to resource conflicts and suboptimal performance when applications require conflicting settings.

B.5 Workflow Telemetry

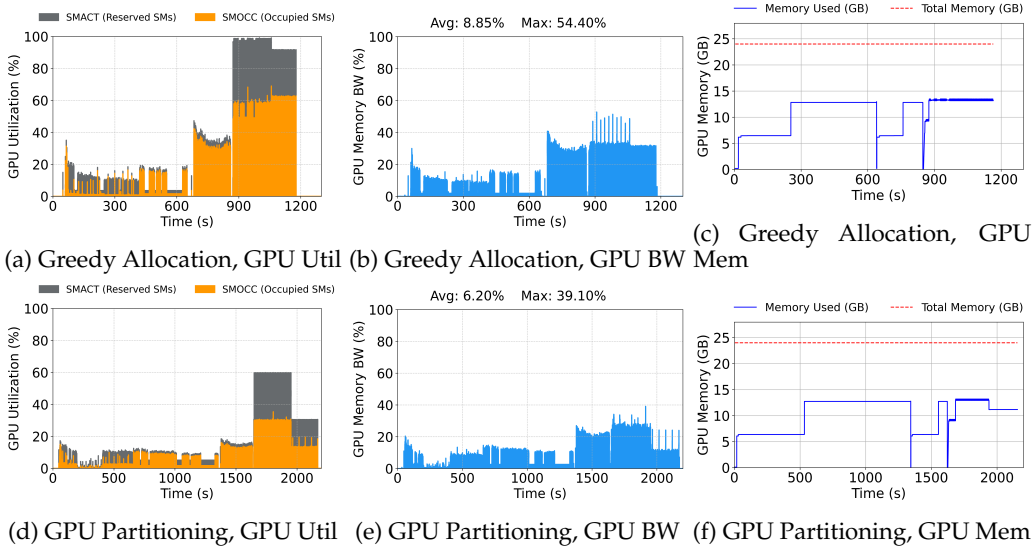


Figure 16: GPU metrics of running the digital content creation workflow.

Fig. 16 and Fig. 17 show the GPU metrics, CPU metrics, and power usage of running the digital content creation workflow using greedy GPU resource allocation and static GPU partitioning. These are augmented results for §4.3 in the paper.

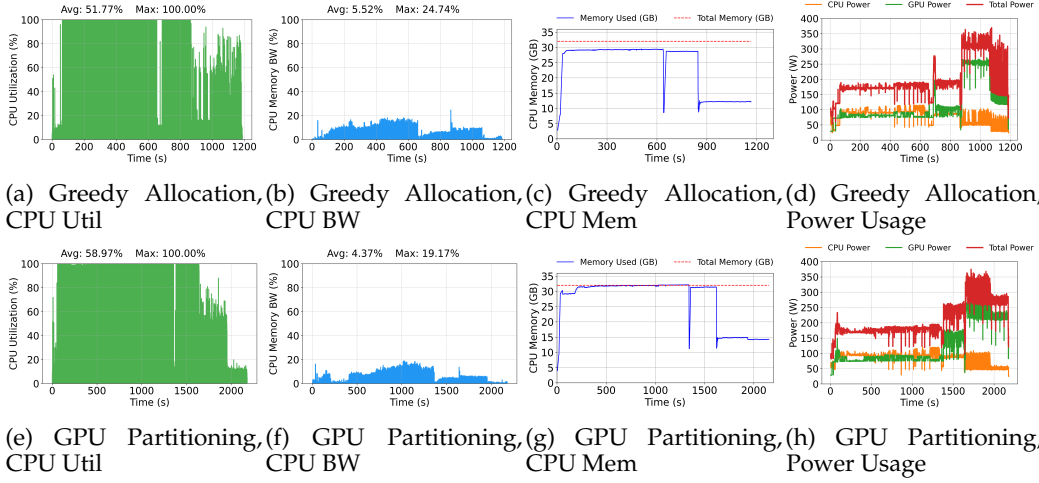


Figure 17: CPU metrics and power usage of running the digital content creation workflow.

Greedy GPU resource allocation overall consumes more GPU resources, and has a higher peak power consumption compared to static GPU partitioning. However, the end-to-end time required for greedy resource allocation is 45% lower than GPU partitioning. This implies that the total power consumed in executing the digital content creation workflow is lower for greedy allocation compared to GPU partitioning.

C Additional Workflow Evaluation

We evaluate end-user experience by simulating a digital content creation workflow with sequential and concurrent tasks. Fig. 18 shows the YAML configuration for the digital content creation workflow. This workflow is a representative end-to-end user scenario with sequential and concurrent tasks:

- **Brainstorming** (Chatbot): ideation to brainstorm ideas.
- **Analysis** (DeepResearch): analysis of existing content and tools to gather relevant information.
- **Preparing outline** (Chatbot): preparing a script or outline to plan the content flow.
- **Creating cover art** (ImageGen): generating a cover image or thumbnail.
- **Generating captions** (LiveCaptions): generating captions for the final content.

Notably, DeepResearch and Chatbot may share a single model instance via llama.cpp (depending on the configuration) to mimic real-world resource constraints on end-user devices.

Fig. 19 shows latencies for workflow tasks with and without GPU partitioning.

Analysis: Greedy GPU allocation shortens the end-to-end workflow time by 45% compared to partitioning, mainly due to faster completion of DeepResearch. In contrast, GPU partitioning limits the resources for DeepResearch, causing delays for subsequent tasks. GPU partitioning, on the other hand, avoids stalling of LiveCaptions by reserving 33% of the GPU while gracefully reducing ImageGen performance by 1.8 $\times$. Thus, despite better fairness, partitioning increases total workflow runtime, highlighting a trade-off: greedy allocation risks starvation but optimizes utilization, while partitioning ensures fairness at the cost of efficiency.

D Cross-Platform and Cross-Configuration Generalization

Here, we provide additional experimental results to show generality across hardware and model choices and to evaluate whether existing systems such as KTransformers (Chen

```

1  Brainstorm (chatbot):
2    model: openai/meta-llama/Llama-3.2-3B-Instruct
3    num_requests: 10
4    device: gpu
5    type: chatbot
6    mps: 100
7    slo: [1s, 0.25s]
8
9  Analysis (deep_research):
10   model: openai/meta-llama/Llama-3.2-3B-Instruct
11   num_requests: 1
12   device: gpu
13   type: deep_research
14   mps: 100
15
16  Preparing Outline (chatbot):
17   model: openai/meta-llama/Llama-3.2-3B-Instruct
18   num_requests: 20
19   device: gpu
20   type: chatbot
21   mps: 100
22   slo: [1s, 0.25s]
23
24  Creating Cover Art (imagegen):
25   server_model: stable-diffusion-3.5-medium-turbo
26   num_requests: 10
27   device: gpu
28   type: imagegen
29   mps: 100
30   slo: 1s
31
32  Generating Captions (live_captions):
33   num_requests: 1
34   device: gpu
35   type: live_captions
36   mps: 100
37   slo: 2s
38
39  workflows:
40   analysis:
41     uses: Analysis (deep_research)
42     background: true
43
44   brainstorm:
45     uses: Brainstorm (chatbot)
46
47   outline:
48     uses: Preparing Outline (chatbot)
49     depend_on: ["brainstorm", "analysis"]
50
51   cover_art:
52     uses: Creating Cover Art (imagegen)
53     depend_on: ["outline"]
54
55   generate_captions:
56     uses: Generating Captions (live_captions)
57     depend_on: ["outline"]
58
59

```

Figure 18: Full YAML configuration of the content creation workflow.

et al., 2025a) already resolve the bottlenecks exposed by CONSUMERBENCH. Overall, the added results reinforce the main findings of the paper: (1) starvation of latency-sensitive workloads persists across a wider set of hardware, software, and model configurations, and (2) systems optimized for a single model do not directly solve interference among concurrent heterogeneous GenAI applications.

D.1 Apple Silicon Replication

We conducted all experiments on an Apple MacBook Pro (M1 Pro chip) with 32GB unified memory, 6 performance cores, 2 efficiency cores, and 200 GB/s memory bandwidth.

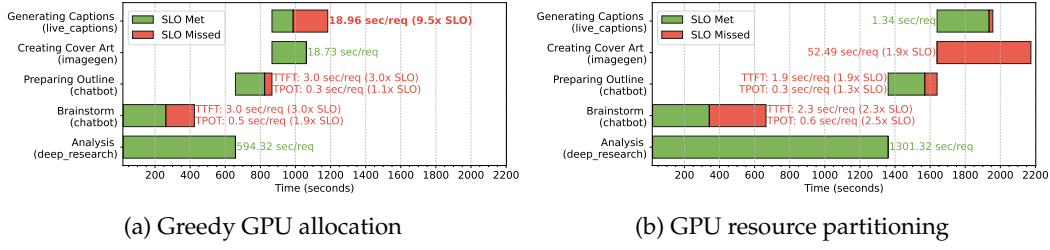


Figure 19: E2E latency & SLO attainment for content-creation workflow w/ and w/o GPU partitioning.

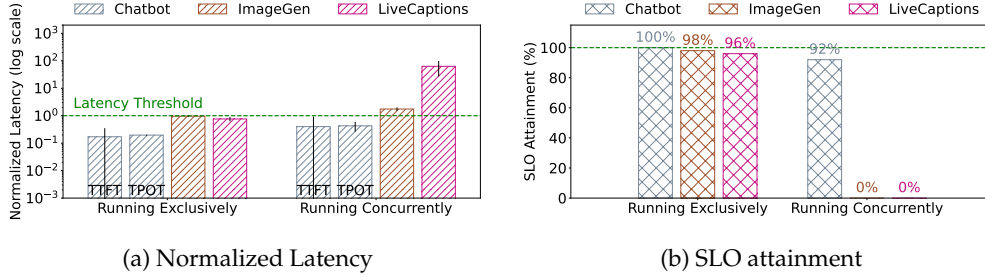


Figure 20: (a) Latencies normalized to SLO requirements and (b) SLO attainment for Chatbot, Image Generation, and Live Captions running exclusively or concurrently on Apple Silicon.

D.1.1 Running Applications Exclusively vs. Concurrently

Fig. 20 shows the normalized latency and SLO attainment of the applications when they run exclusively and concurrently on Apple Silicon. Fig. 21 further shows the GPU utilization and power usage of each scenario. When applications are run in isolation, they are able to meet their SLOs for the majority, if not all, of their requests. However, when applications are executed concurrently, we observe a different pattern compared to the Intel server setup. Specifically, ImageGen experiences a slight performance degradation, while LiveCaptions suffers a significant decline. This behavior suggests that Apple Silicon attempts to fairly schedule GPU compute resources among applications, but the scheduling is suboptimal, leading to resource starvation for LiveCaptions. Additionally, Apple Silicon does not support static GPU partitioning, and we were unable to explore alternative application configurations.

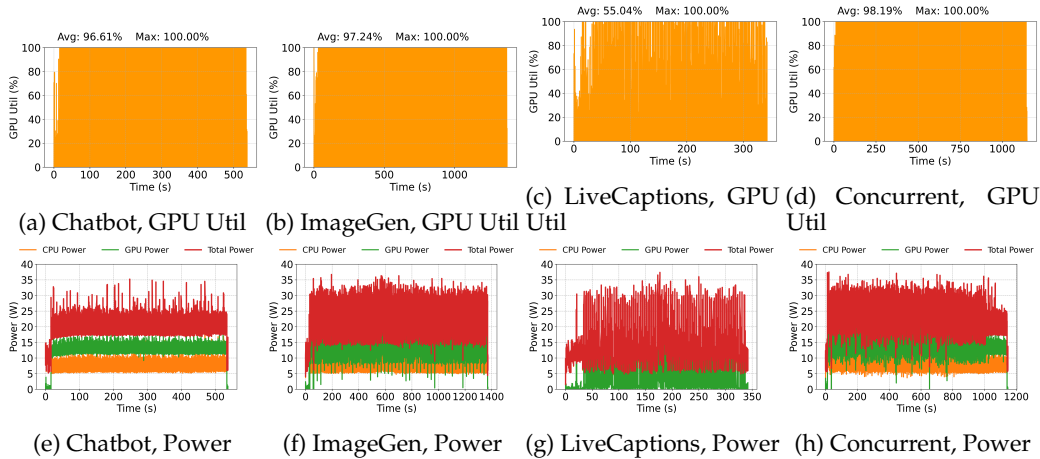


Figure 21: Metrics of running applications exclusively and concurrently on Apple Silicon.

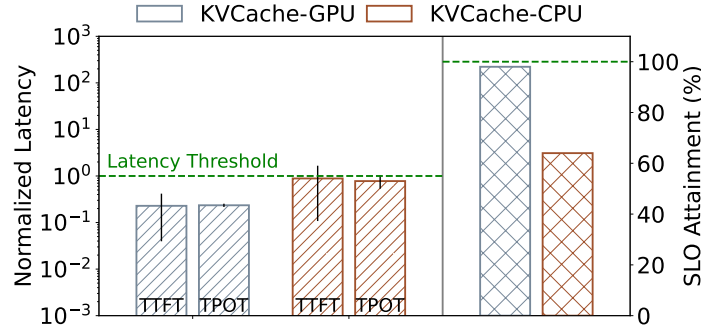


Figure 22: Normalized latency and SLO attainment for Chatbot and Chatbot-KVCache-CPU on Apple Silicon.

D.1.2 Static Model Sharing via Inference Servers

Fig. 22 shows the normalized latency and SLO attainment of running Chatbot and Chatbot-KVCache-CPU on Apple Silicon. Fig. 23 (a, b) show the GPU utilization, while Fig. 23 (d, e) shows the power usage of each scenario. The performance for Chatbot-KVCache-CPU is similar on Apple Silicon compared with the Intel server. This is for similar reasons. Chatbot-KVCache-CPU does not use GPU cores for attention computation in Chatbot-KVCache-CPU, incurring slowdowns compared to Chatbot.

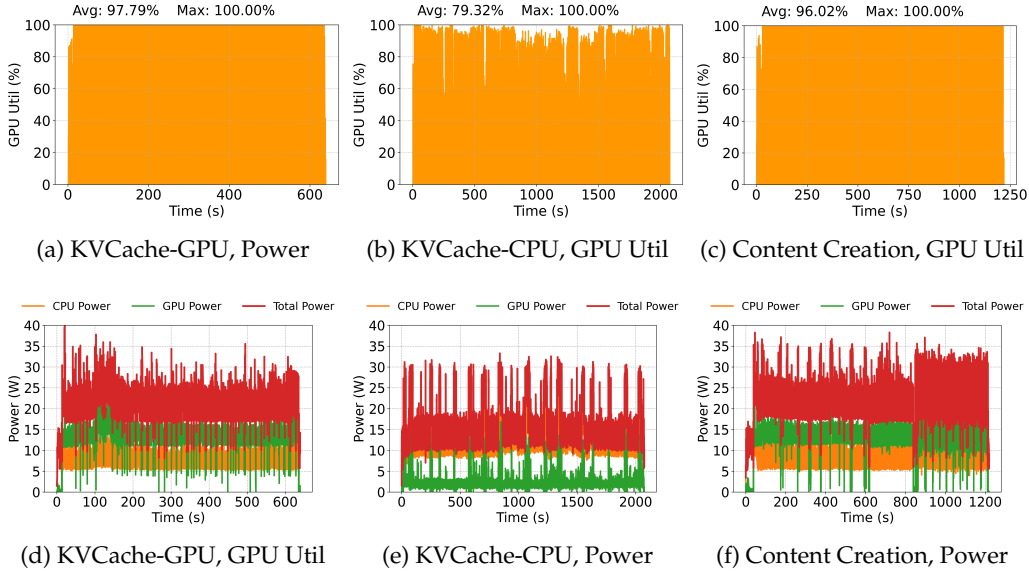


Figure 23: Metrics of running Chatbot, Chatbot-KVCache-CPU, and the content creation workflow on Apple Silicon.

D.1.3 End-to-end Workflow

We execute the digital content creation workflow on Apple Silicon and present its performance in Fig. 24. Fig. 23c and Fig. 23f show the workflow's GPU utilization and power usage. This workflow demonstrates improved fairness in resource allocation on Apple Silicon. Overall, the end-to-end application latency remains comparable to that of the Intel server, while LiveCaptions experiences slightly less resource starvation than greedy resource allocation on the Intel platform (8× on Apple Silicon compared to 9.5× on the Intel server). Consequently, Apple Silicon achieves a slightly more balanced trade-off between application SLO adherence and overall workflow efficiency compared to the Intel server.

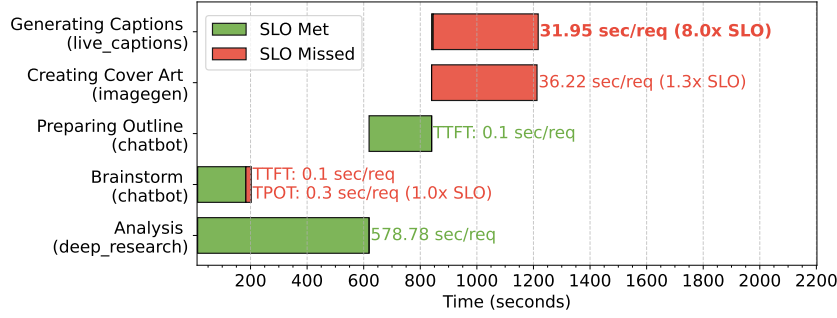


Figure 24: E2E latency & SLO attainment for the content-creation workflow running on Apple Silicon.

D.2 Other Hardware and Models

D.2.1 Kernel Level Generalization

To confirm the kernel contention mechanism is not specific to the RTX 6000, we add kernel-level profiling on an NVIDIA Jetson Orin Nano, which is a consumer/embedded platform commonly used for on-device AI, confirming the mechanism generalizes. Following the setup in Table 2, we provide profiles for Chatbot and LiveCaptions respectively.

Chatbot in Jetson Nano does not occupy the SMs efficiently. Each decode step launches kernels that reserve a large number of SMs but only utilize 60-70% of SMs reserved. On the Jetson Nano’s small SM count, one Chatbot does not saturate compute, yet it still monopolizes the GPU at the scheduling granularity.

LiveCaptions’ decode phase issues small kernels at low SM occupancy. These are exactly the kernels that should interleave into the gaps Chatbot leaves on the SMs, but under default time-slicing they are queued behind Chatbot’s large kernels instead. This is the same pattern as shown in Fig. 4, now confirmed on a consumer-grade platform.

For Apple Silicon, equivalent kernel-level analysis is currently limited because Apple’s Metal System Trace (<https://developer.apple.com/metal/tools/>) does not expose a directly comparable Nsight-style per-kernel profiler with launch metrics such as registers per thread, shared-memory usage, and occupancy-related limits. We will note this as an important direction for future Apple-platform support in ConsumerBench.

To test whether our observations generalize beyond the main setup in the paper, we evaluated CONSUMERBENCH on six additional configurations spanning consumer GPUs, an embedded device, multiple operating systems, and different model choices for each modality. We consider the RTX 4080 and RTX 3060 on Windows WSL, the Jetson Orin Nano on Ubuntu, and several RTX 6000 configurations on Ubuntu with different chatbot, image generation, and speech models. The models span multiple open model families, including Gemma-2B (Google, 2026), Llama (Meta AI, 2024), Qwen2.5-7B (Team, 2026b), DeepSeek (DeepSeek-AI, 2024), Stable Diffusion variants (CompVis, 2026; AI, 2026; Studios, 2024), Kandinsky (Team, 2026a), and Whisper (Radford et al., 2022).

Because MPS is unavailable in WSL and on Jetson Orin Nano, these runs use greedy allocation. The resulting SLO attainment is shown in Table 7. Across all six configurations, the same qualitative trend holds: LiveCaptions is consistently the most vulnerable workload under concurrent execution and substantially exceeds its 2 s SLO, even when chatbot and image generation often remain near or within their SLOs. For example, LiveCaptions reaches 23.242 s on configuration A, 91.504 s on B, and 40.274 s on D. These results suggest that the starvation pattern identified in the main paper is not tied to a single GPU, model family, or software stack.

Takeaway. The central failure mode from the main paper persists across a broader set of environments: when multiple GenAI applications share the same device, lightweight latency-sensitive tasks such as live captioning are easily starved by heavier GPU workloads.

Config.	Hardware	OS	Chatbot Model	ImageGen Model	LiveCaption Model
A	RTX 4080	WSL	Gemma-2B	SD-3-medium	Whisper-large-v3-turbo
B	RTX 3060	WSL	Llama-3.2-1B	SD-3.5-medium-Q8	Whisper-small
C	Jetson Orin Nano	Ubuntu	Llama-3.2-1B Q8_0	–	Whisper-tiny
D	RTX 6000	Ubuntu	Llama-3.2-3B	SD-3.5-medium-turbo	Whisper-large-v3-turbo
E	RTX 6000	Ubuntu	Qwen2.5-7B	kandinsky-2-2-decoder	Whisper-medium
F	RTX 6000	Ubuntu	GPT-OSS-20B	kandinsky-2-2-decoder	Whisper-small

Table 6: Additional configurations spanning more hardware platforms, operating systems, and model choices.

Hardware Setup	Chatbot TTFT (SLO = 1s)	Chatbot TPOT (SLO = 0.25s)	ImageGen (SLO = 1s)	LiveCaption (SLO = 2s)
A	0.070	0.026	0.621	23.242
B	0.062	0.026	1.651	91.504
C	0.097	0.031	–	45.488
D	0.100	0.039	0.874	40.274
E	0.060	0.017	0.126	6.217
F	0.098	0.011	0.121	15.479

Table 7: Additional results on different hardware/software/model configurations under greedy allocation.

D.2.2 Kernel-level Profiling on Jetson Orin Nano

To test whether the kernel-contention mechanism in §4.1 is specific to the RTX 6000, we repeat the kernel-level profiling on the Jetson Orin Nano, an embedded consumer platform. We use configuration C from Table 6: a Llama-3.2-1B Q8_0 Chatbot and Whisper-tiny LiveCaptions.

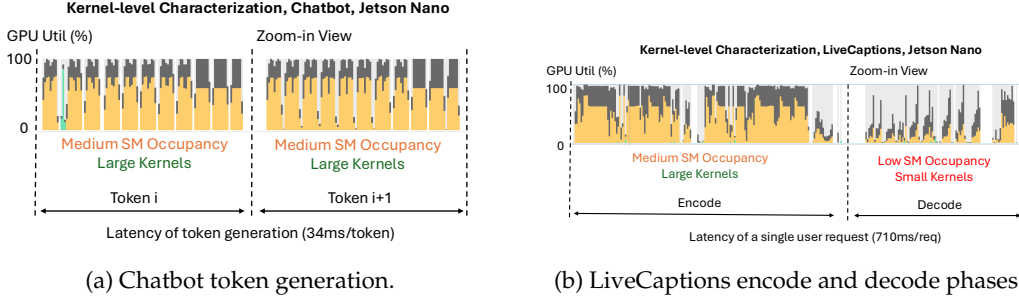


Figure 25: Kernel-level characterization on Jetson Orin Nano. Gray denotes reserved SMs (SMACT), while yellow denotes occupied SMs (SMOCC).

As Fig. 25 shows, each Chatbot decode step launches large kernels that reserve many SMs but occupy only approximately 60–70% of the reserved capacity. On the Jetson’s small SM count, the Chatbot therefore does not saturate compute, yet it can monopolize the GPU at the scheduler’s kernel granularity. LiveCaptions exhibits a complementary profile: its encode phase uses larger kernels, while its decode phase issues small, low-occupancy kernels. Those decode kernels could use the capacity left idle by Chatbot, but default time slicing queues them behind the larger kernels. This reproduces the contention pattern from Fig. 4 on a second NVIDIA platform.

Equivalent kernel-level validation is currently limited on Apple Silicon because Metal System Trace does not expose directly comparable system performance metrics. We therefore restrict the kernel-level generalization claim to the NVIDIA platforms evaluated here.

D.3 Concurrently Executing Applications with Larger Models

We use CONSUMERBENCH to evaluate the performance of applications with larger models that do not concurrently fit in the GPU memory of our server. Specifically, we change the model used in Chatbot from Llama-3.2-3B to Llama-3.1-8B, which requires 16GB of memory (not counting the KV cache). We execute Chatbot exclusively on the CPU while concurrently running ImageGen and LiveCaptions on the GPU. Fig. 26 shows the performance of the applications using greedy GPU allocation and static GPU partitioning. These are augmented results for §4.2 in the paper. Fig. 27 and Fig. 28 further show the GPU metrics, CPU metrics, and power usage of the different GPU management strategies.

Overall, Chatbot exhibits reduced performance with the larger model compared to Llama-3.2-3B, leading to SLO violations. Although LiveCaptions also experiences SLO violations, its resource starvation is alleviated due to reduced contention when not all three applications share the GPU simultaneously. Lastly, partitioning the GPU between ImageGen and LiveCaptions effectively eliminates starvation for LiveCaptions, though it causes ImageGen to run slightly slower compared to scenarios with greedy resource allocation.

E Comparison to Existing Systems and Scheduling Mechanisms

E.1 Single-Model Optimization vs. Cross-Application Scheduling

KTransformers (Chen et al., 2025a) is a system designed to accelerate *single* MoE-style LLM inference through heterogeneous CPU/GPU execution, including mechanisms such

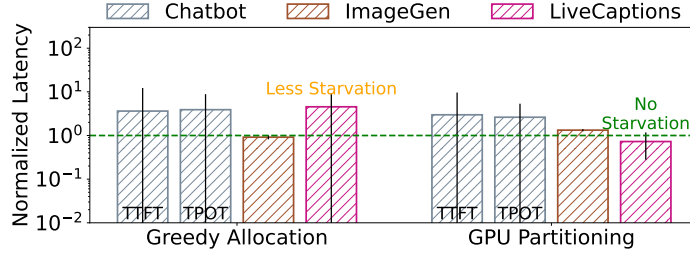


Figure 26: Normalized latency of running larger applications concurrently using greedy allocation and static GPU partitioning

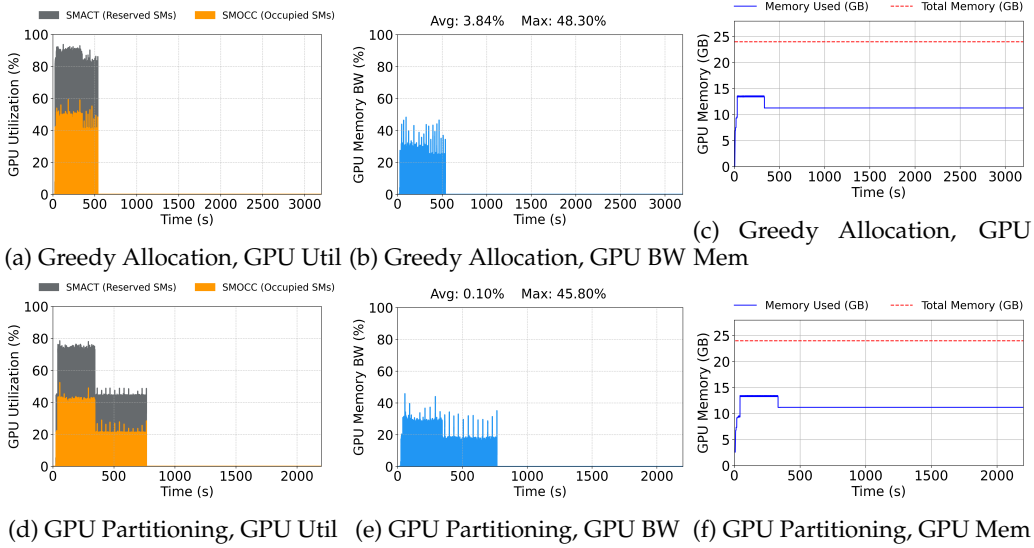


Figure 27: GPU metrics of running larger applications concurrently.

as expert placement and specialized kernels. In other words, it optimizes one model’s execution path by distributing model components across heterogeneous resources. This design point is fundamentally narrower than the setting studied by CONSUMERBENCH, which focuses on *multiple concurrent applications* with different modalities and competing SLOs sharing an end-user device. In particular, CONSUMERBENCH highlights needs such as global cross-application resource management, request-level SLO-aware scheduling, and inference-server configurability for mixed workloads, none of which are the primary target of KTransformers.

To test this directly, we evaluated KTransformers on DeepSeek-V2-Lite (DeepSeek-AI, 2024) models in both standalone and concurrent settings, where the LLM runs alongside LiveCaptions and ImageGen. The results are shown in Table 8. Under standalone execution, both quantization settings comfortably meet the chatbot SLOs. Under concurrent execution, however, TTFT increases substantially: DeepSeek-V2-Lite Q4 rises from 0.205 s to 1.035 s, and Q8 rises from 0.719 s to 1.180 s. TPOT also degrades, though it remains within the 0.25 s SLO in these measurements. These results indicate that a system optimized for efficient single-model execution does not automatically prevent contention-induced latency inflation once multiple heterogeneous applications share the device.

Takeaway. KTransformers is effective for optimizing a single model, but our results show that this is not sufficient to resolve cross-application interference in realistic concurrent GenAI workflows. CONSUMERBENCH therefore targets a complementary systems problem: scheduling and resource management across heterogeneous applications rather than optimization of a single model alone.

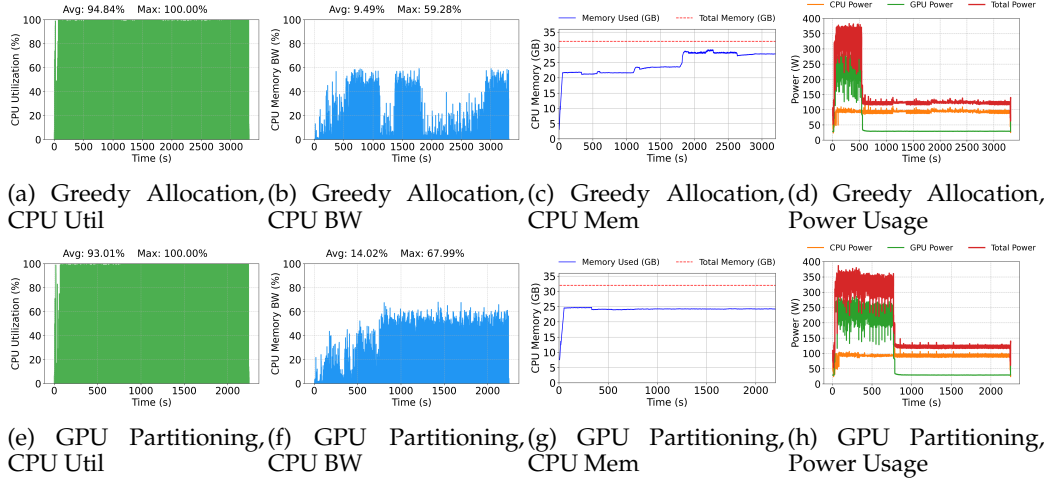


Figure 28: CPU metrics and power usage of running larger applications concurrently.

Model Setting	TTFT (SLO = 1s)	TPOT (SLO = 0.25s)
DeepSeek-V2-Lite Q4 Standalone	0.205	0.025
DeepSeek-V2-Lite Q4 Concurrent	1.035	0.142
DeepSeek-V2-Lite Q8 Standalone	0.719	0.037
DeepSeek-V2-Lite Q8 Concurrent	1.180	0.161

Table 8: Additional evaluation of KTransformers in standalone and concurrent settings.

E.2 Comparison of GPU Scheduling and Runtime Mechanisms

This subsection provides a framework-level comparison of prior GPU-sharing and scheduling mechanisms. The main point is not that adaptive scheduling is unimportant; rather, many existing systems do not transparently support arbitrary, unmodified GenAI applications running concurrently on a single end-user GPU. We refer to representative systems including TGS (Wu et al., 2023), Orion (Strati et al., 2024), REEF (Han et al., 2022), NVIDIA MPS (Corporation, 2025c) and MIG (NVIDIA, 2025b), FaaSSwap (Yu et al., 2024) and Torpor (Yu et al., 2025), CUDA Green Contexts (NVIDIA, 2025a), CUDA priority streams, and LLM-serving runtimes such as LightLLM (Gong et al., 2025).

Table 9 summarizes the comparison of representative scheduling strategies and runtime systems. The comparison focuses on three properties that matter in our setting: whether the system transparently supports arbitrary GenAI models on a single GPU, whether it schedules resources *across* applications rather than only within one application, and whether it supports an arbitrary number of concurrent applications.

- **CUDA Priority Streams and Green Context.** These mechanisms are useful for prioritization within a single CUDA context (NVIDIA, 2025a), but they do not provide effective scheduling across independent applications. As a result, they are not sufficient for CONSUMERBENCH, where multiple applications run concurrently with separate contexts.
- **REEF and Orion.** These schedulers (Han et al., 2022; Strati et al., 2024) are closer in spirit to cross-application GPU management, but they are limited in practice for this workload class: REEF supports only a restricted set of DNN models, while Orion requires offline kernel profiling and application-level adaptation, and is likewise oriented toward DNN kernels rather than arbitrary modern GenAI stacks.

Takeaway. Table 9 summarizes how existing mechanisms differ at a high level; we present additional scheduler-specific empirical results in the next subsection (§E.3).

System	Transparent support for arbitrary GenAI models on a single GPU	Cross-application scheduling	Arbitrary number of concurrent applications
TGS	Y	Y	N (only one high-priority app)
Orion	N	Y	Y
REEF	N	Y	Y
MPS, MIG	Y	Y	Y
FaaS, Torpor	N	Y	Y
Green Context	Y	N	N
Priority Streams	Y	N	N
KTransformers, llama.cpp	N	N	N
Apple Silicon	Y	Y	Y

Table 9: Comparison of existing GPU scheduling and runtime strategies in the CONSUMER-BENCH setting.

Configuration & Priority	ImageGen	LiveCaptions
ImageGen (Low) + LiveCaptions (High)	Severe SLO Violation	Meets SLO
ImageGen (Low) + LiveCaptions (Low)	Meets SLO	Starvation

Table 10: Evaluation of TGS on concurrent ImageGen and LiveCaptions.

E.3 Additional Scheduler-Specific Results

This subsection reports additional empirical results and sensitivity checks for specific schedulers and mechanisms (see [Tables 10 to 12](#)).

- **TGS.** Among the more relevant dynamic schedulers, TGS ([Wu et al., 2023](#)) still has important restrictions: it supports only two concurrent applications with predefined binary priority. It also has implementation-level incompatibilities with newer frameworks such as diffusers ([von Platen et al., 2022](#)) and faster-whisper ([SYSTRAN, 2026](#)). In an added experiment with ImageGen and LiveCaptions, TGS exhibits a strong trade-off: giving LiveCaptions high priority lets it meet its SLO but causes severe slowdown for ImageGen; assigning equal low priority allows ImageGen to meet its SLO but starves LiveCaptions.
- **MPS and MIG.** These are among the few practical mechanisms that can be applied transparently to unmodified applications ([Corporation, 2025c](#); [NVIDIA, 2025b](#)). We add an MPS baseline on RTX 6000 and show that assigning MPS=100 to all applications reduces contention compared with greedy allocation, but still leaves LiveCaptions far above its 2 s SLO (11.756 s). Static partitioning helps LiveCaptions more than default MPS but hurts ImageGen, illustrating the fairness–efficiency trade-off.
- **Oracle adaptive MPS.** To quantify the potential headroom of adaptive scheduling, we simulate an oracle policy by grid-searching per-application MPS thresholds and selecting the smallest value that preserves standalone SLOs. Under this optimal setting, all applications satisfy their SLOs, with LiveCaptions improving from 11.756 s under default MPS to 1.25 s. This does not provide a deployable scheduler on its own, but it does show that the resource conflict is, in principle, schedulable.
- **Other systems.** Fairness in Serving Large Language Models ([Sheng et al., 2024](#)), LithOS ([Coppock et al., 2025](#)), and Torpor/FaaS ([Yu et al., 2025](#); [2024](#)) are incomplete fits for the target setting. Fairness in Serving Large Language Models is specialized to LLM-serving stacks such as LightLLM ([Gong et al., 2025](#)) rather than arbitrary multimodal applications; LithOS was not open-sourced, and Torpor/FaaS targets serverless or non-concurrent execution settings rather than continuous concurrent GPU sharing on end-user devices.

Application (SLO)	MPS All 100	Static Partitioning	Greedy Allocation
Chatbot – TTFT (1s)	0.092	0.124	0.103
Chatbot – TPOT (0.25s)	0.022	0.034	0.039
ImageGen (1s)	0.687	1.870	0.980
LiveCaptions (2s)	11.756	2.436	28.916

Table 11: Comparison of default MPS, static partitioning, and greedy allocation.

Application (SLO)	MPS Optimal	MPS All 100	Greedy Allocation
Chatbot – TTFT (1s)	0.154	0.092	0.103
Chatbot – TPOT (0.25s)	0.039	0.022	0.039
ImageGen (1s)	0.927	0.687	0.980
LiveCaptions (2s)	1.25	11.756	28.916

Table 12: Oracle adaptive MPS configuration from grid search.

Summary. The existing GPU scheduling work is highly relevant, but most current systems either optimize a narrower workload class, require model/framework-specific modifications, or do not directly support concurrent heterogeneous GenAI applications on a single consumer device. This gap motivates CONSUMERBENCH as a benchmarking framework for identifying where deployable cross-application scheduling remains inadequate.