RT-SWAP: ADDRESSING GPU MEMORY BOTTLENECKS FOR REAL-TIME MULTI-DNN INFERENCE

Jaeyoung Lee

Yonsei University

January 23, 2026

Before We Go Deeper: Real-Time for ML

Recent Research in Timing Guarantees for ML

- **RT-MOT (2022)**: Confidence-aware scheduling for multi-object tracking.
- **DNN-SAM (2022)**: Split-and-merge execution for real-time detection.
- **LaLaRAND (2021)**: Layer-by-layer CPU/GPU scheduling.
- **RT-Swap (2024)** : Addressing GPU memory capacity bottlenecks.
- **RT-MDM (2024)**: Multi-DNN on MCUs using external memory.

Categorization of Approaches

1. **Pipeline Optimization**: Focusing on dividing pipelines to ensure timing (e.g., RT-MOT, DNN-SAM).
2. **Resource Constraint Management**: Managing hardware limitations such as memory or compute units (e.g., LaLaRAND, RT-Swap, RT-MDM).

Part I

Introduction and Design Principles

Research Motivation: The GPU Memory Wall

The Problem: Memory Capacity Bottleneck

- DNN complexity and memory demands are increasing rapidly.
- GPU memory growth lags behind computational power.
- Physical Limit : Restricted memory capacity impedes the deployment of sophisticated models.

Real-Time Multi-DNN Challenges

- Existing swapping (e.g., Unified Memory with LRU) is unpredictable.
- Worst-case timing guarantees are difficult to maintain during data movement.
- Need for efficient memory management that ensures both oversubscription and timeliness .

RT-Swap: Framework Overview

What is RT-Swap?

RT-Swap is a real-time memory management framework that can

- Uses transparent and efficient swapping mechanism.
- Consistent virtual memory management mechanism to ensure seamless data access for DNN tasks regardless of memory swapping.

Key Contributions

1. **Transparency**: Works with popular ML frameworks (PyTorch, Darknet) without model or driver modifications.
2. **Determinism**: Provides predictable swapping overheads for real-time scheduling.
3. **Efficiency**: Task-level VMM to mitigate fragmentation issues.

Part II

BACKGROUND

How DNN Tasks Work in Real-Time Systems

Task Characteristics

- **Model**: A single DNN model per task.
- **Job**: Processes one inference request per job instance.
- **Timing**: Must complete execution within a specific deadline.

Two-Step Execution Process

1. **Initialization Step**: Model parameters (weights, feature maps, etc.) are preloaded into GPU memory.
2. **Inference Stage**: Computations are performed using the preloaded parameters.

STANDARD GPU MEMORY MANAGEMENT

Memory Allocation

- Uses `cudaMalloc` to allocate memory as memory objects .
- **VA to PA Mapping**: Managed entirely by the GPU driver.

The Data Access Constraint

- GPU can only access data stored in its local physical memory.
- Accessing CPU memory requires manual data migration (e.g., `cudaMemcpy`).
- Standard drivers do not provide timing predictability or schedulability for real-time systems.

Unified Memory (UM) and Predictability Issues

Concept of Unified Memory

- Shared virtual address space between CPU and GPU.
- Uses `cudaMallocManaged` for on-demand paging.

The LRU Problem

- GPU driver evicts pages using the Least Recently Used (LRU) policy when memory is full.
- Unpredictability: The amount and timing of page evictions are stochastic.
- Infeasible for real-time multi-DNN systems requiring worst-case guarantees.

Leveraging Low-level GPU VMM APIs

The Need for Full Control

- To ensure predictability, the user (framework) must control physical memory allocation and virtual mapping.

Benefit: Enables independent management of VA and PA, removing reliance on the default GPU driver's unpredictable behavior.

Part III

Design Principles

ADDRESSING GPU MEMORY BOTTLENECKS

Objective

- Enabling DNN models larger than physical GPU capacity.
- Ensuring timely execution during CPU-GPU swapping.

The Three Key Challenges

- **C1: Capacity Expansion**
  - How to allocate memory exceeding physical limits?
    $\Rightarrow$ Data must be pre-loaded fluently between GPU and CPU.
- **C2: Access Transparency**
  - How to provide seamless access during oversubscription?
    $\Rightarrow$ Requires consistent mapping of the original virtual address .
- **C3: Timing Guarantees**
  - How to provide offline guarantees with swapping overhead?
    $\Rightarrow$ Requires determining the maximum swap volume.

Principle 1: Task-level Predictable Memory Switching

Runtime Swap Manager

- Orchestrates data movement at the task level to ensure predictability.

Management Logic

1. **Tracking**: Keeps track of all allocated memory objects for each task.
2. **Eviction**: Determines which objects to swap out when GPU memory is required.
3. **Restoration**: Reallocates and swaps in necessary objects before the execution starts.

PRINCIPLE 2: TRANSPARENT GPU VIRTUAL MEMORY MANAGEMENT

Utilizes CUDA low-level VMM APIs to ensure virtual address consistency.

Comparison of Management Approaches

- **Standard Allocation**: Assigns a new VA during reallocation, breaking DNN execution.
- **Object-level VMM**:
  - Ensures VA consistency but causes severe fragmentation.
  - Fixed 2MB chunks lead to unused space for objects $<$ 2MB.
- **Task-level VMM (RT-Swap)**:
  - Maintains a contiguous task-level VA range .
  - Maps uniform physical chunks sequentially to minimize fragmentation.

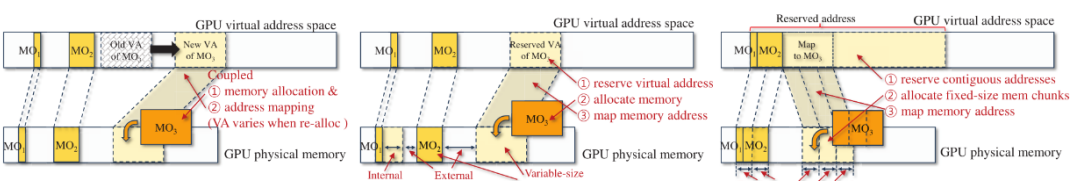


Fig. 1: Three different GPU memory management approaches

Principle 3: Swap-aware Real-time Scheduling

Offline Optimization

- Ensure continual data retrieval from the GPU memory during execution
- Ensure the completion of execution before the deadline

Key Strategies

- **Efficiency**:
  - Minimizes unnecessary swap-in/out operations.
  - Latency Hiding: Maximizes the overlap between GPU computation and data transfer.
- **Schedulability Analysis**: Proven timing guarantees for multi-DNN task sets.

Part IV

System Design

RT-Swap Library: Implementation and Initialization

RT-Swap Library
Implementation

- Implemented as a shared library (`LD_PRELOAD`).
- No source code modifications required for DNN models or GPU drivers.
- Intercepts CUDA runtime API functions and executes corresponding wrapper functions.

Initialization Phase

- Holds essential information: Maximum swap volume and swap candidate memory objects.
- Reserves a single contiguous VA range equivalent to the maximum swap volume.

System Architecture Overview

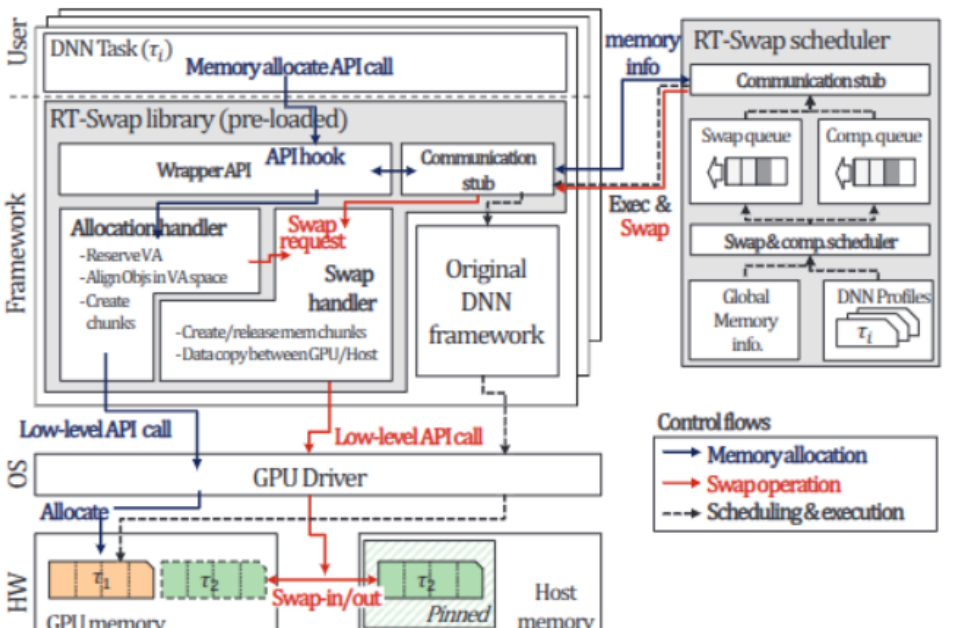


Fig. 2: System architecture overview

Memory Allocation and Swapping Mechanisms

Allocation/Deallocation Logic

- **Non-swap Candidates**: Allocated persistently in GPU memory via original API calls.
- **Swap Candidates**: Assigned fixed placements within the reserved VA range, created as multiple uniform physical chunks .
- Communicates with the Scheduler to free GPU memory when capacity is reached.

Swapping Operations

- Swapping is directed by the RT-Swap Scheduler at runtime.
- **Optimization**: Uses Host-pinned memory instead of Pageable memory.
  - Permits direct data transfer (DMA) without temporary buffers.
  - Eliminates CPU allocation stage by pre-allocating swap volume.

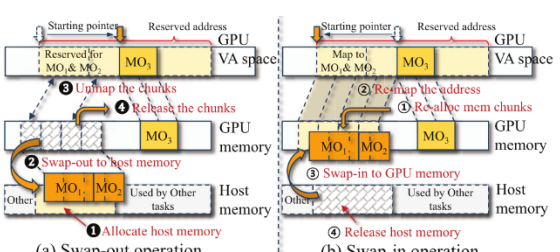


Fig. 3: Swap-out/in operation procedures

Optimizing Swapping Performance

Chunk Size Trade-off

- **Small Swap Chunks**: Higher swapping overhead due to many management operations.
- **Large Swap Chunks**: Longer copy time and higher internal fragmentation.
- Goal : Find optimal size to minimize total latency.

Mitigating Fragmentation Overheads

- **Internal**: Standard VMM requires 2MB chunks. RT-Swap bounds this overhead within the task-level VA range.
- **External**: Solved by using uniform-sized physical chunks rather than variable object-level sizes.

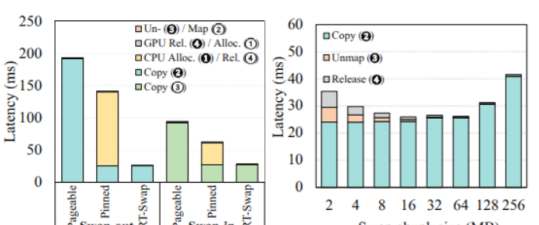


Fig. 4: Swap latency of 300MB memory: (a) latency breakdown of three methods; (b) impact of swap chunk sizes

RT-SWAP SCHEDULER: GLOBAL MANAGEMENT

RT-Swap Schedeuler

An independent process that maintains system-wide information and coordinates task-level libraries.

Initialization Stage

- Determines offline swap volume and swap chunk size assignments.
- Maintains task profiles: Timing constraints, WCET, swap latency, and memory objects.

Inference Stage

- Manages task execution timing and schedules swap-in/out operations.
- Monitors available free GPU memory and tracks swapped-out physical chunks.
- Proactive Action: Issues a swap-in request before the execution of a DNN task to hide latency.

Part V

SWAP-AWARE REAL-TIME SCHEDULING

Real-Time DNN Task Model

System Model

- CPU-GPU platform with GPU capacity m^D and large CPU memory.
- Swap chunk size δ as the unit of memory swapping.

Periodic Task Model: $\tau_i = (M_i, C_i, T_i, D_i)$

- $D_i = T_i$ (Implicit deadline).
- M_i: Memory profile determined offline.
- m_i: Total footprint (m_i^s swappable, $m_i - m_i^s$ non-swappable).
- $O^{In}(x_i, \delta)$, $O^{Out}(x_i, \delta)$: Max swap-in/out time.

Execution Features

- EDF Priority: Earliest Deadline First ordering.
- Parallelism : Separate engines allow computation and copy in parallel.
- Non-preemptive execution for both tasks and swap operations.

Target Scheduling Problems (P1 & P2)

Swap Requirements

- **R1**: Necessary data (up to x_i) must be swapped in before execution.
- **R2**: Sufficient free GPU memory must be ensured by swapping out other tasks.
- **R3**: All jobs must meet deadlines for any legal arrival sequence.

Problem Definitions

- **P1 (Algorithm Design)**: Bound swap operations (G1) and maximize computation/swap overlap (G2).
- **P2 (Optimization)**: Determine optimal $\{x_i\}$ and δ to satisfy R1–R3.

Swap Schedule Generation (G1)

Two-Queue Management

- **Computation Queue** & **Swap Queue** (both EDF-scheduled).
- $\tau^{CPU}(t)$: Subset of tasks with volumes in CPU at time t.
- Execution is blocked until necessary swap-in is completed.

Property 1: Addressing G1 [Bound swap operations]

- No preemption allowed between a task's swap-in and its execution.
- Every job performs at most one swap-in and one swap-out.

PROPERTY 1

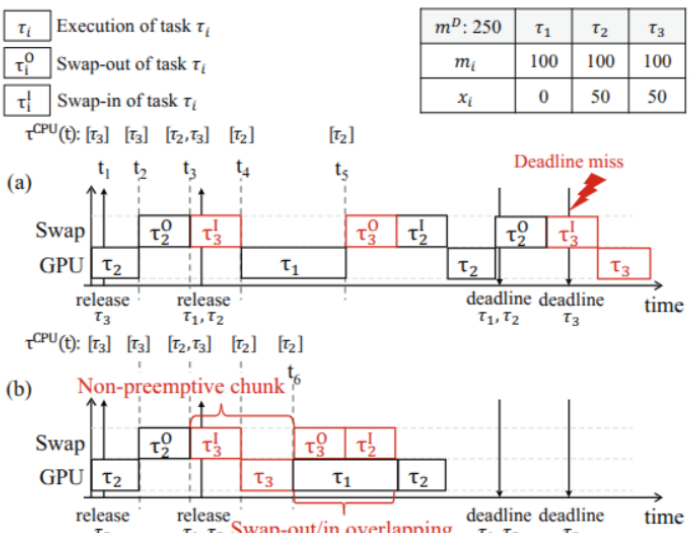


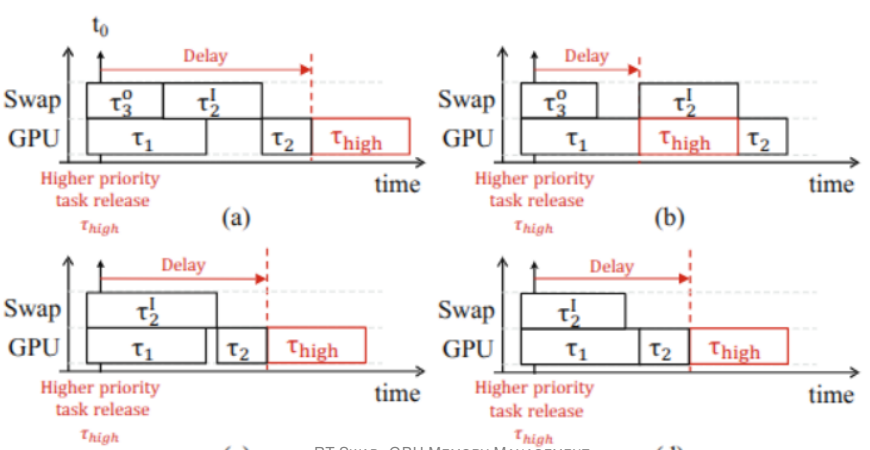
Fig. 5: Schedules for swap operations and normal executions: (a) unnecessary swap operations performed, (b) which is addressed by the RT-Swap Scheduler

MAXIMIZING OVERLAP (G2)

Overlap Strategy

$Q_c(t)$: set of jobs in the computation queue at time t
If highest priority job J_x in $Q_c(t)$ does not belong to $\tau^{CPU}(t)$,
RT-Swap scheduler schedules the swap in operation of the highest priority job J_y in $Q_c(t) \cap \tau^{CPU}(t)$ to overlap it with execution of J_x

- Avoid High Blocking: Start highest-priority swap-in only if its corresponding execution is also the highest priority.



Swap Volume Assignment (P2)

Optimization Problem: Minimize total swap volume.

$$\min_{x_i, \delta} \quad \sum_{\tau_i \in \tau} x_i$$

Constraints

- **Swap Limit**: $0 \leq x_i \leq m_i^S$.
- **Memory Capacity**: Ensure non-swapping volume + current m_i does not exceed m^D.

$$\sum_{\tau_i \in \tau} m_i - \sum_{\tau_j \neq \tau_i} x_j \leq m^D, \quad \forall \tau_i \in \tau$$

- **Timing**: Must satisfy Schedulability Analysis.

UB of total time for a job

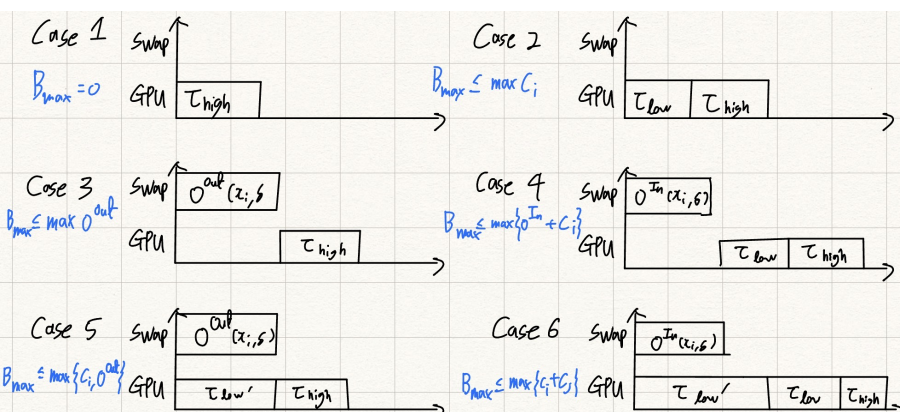
- $O^{Out}(x_i, \delta) + O^{In}(x_i, \delta) + C_i$

LEMMA 1: MAXIMUM BLOCKING TIME (B_{max})

The maximum blocking time B_{max} for a high-priority job is:

$$B_{max} = \max_{\tau_i \in \tau}(O^{Out}, O^{In} + C_i, C_{max,1} + C_{max,2})$$

6 Case Proof Analysis



Theorem 1: Schedulability Condition

Theorem 1

A task set τ is schedulable by the RT-Swap scheduling algorithm if the following condition holds for a given swap chunk size δ and swap volumes $\{x_i\}$:

$$\frac{B_{\max}}{\min_{\tau_i \in \tau} T_i} + \sum_{\tau_i \in \tau} \frac{O^{\text{out}}(x_i, \delta) + O^{\text{in}}(x_i, \delta) + C_i}{T_i} \leq 1.0$$

Proof Sketch

- The theorem applies utilization-based non-preemptive EDF analysis to these bounds.
- The total execution of each task τ_i (including swap overhead) is upper-bounded by $O^{\text{out}}(x_i, \delta) + O^{\text{in}}(x_i, \delta) + C_i$.
- The maximum delay caused by lower-priority tasks is captured by $B_{\max}$.

Intuition and Mathematical Idea

Schedulability on Range $[t, t + L]$

- To be schedulable, the following must hold:

$$B_{\max} + \sum_i \frac{C_i}{T_i} \cdot L \leq L$$

- **Interpretation**:
  - $B_{\max}$: Maximum blocking that started before the interval t.
  - $\sum \frac{C_i}{T_i} \cdot L$: Execution and blocking time occurring within the interval L.

Derivation

- Rearranging the terms: $B_{\max} \leq L \left(1 - \sum_i \frac{C_i}{T_i}\right)$
- The worst-case occurs at the smallest possible interval, $L = \min_i T_i$.
- Dividing by $\min T_i$ yields the final condition:

$$\frac{B_{\max}}{\min_i T_i} + \sum_i \frac{C_i}{T_i} \leq 1$$

Implementation Details

Prototype and Compatibility

- Implemented on Darknet and PyTorch frameworks.
- Compatible with any NVIDIA GPU supporting CUDA 10.2+.

RT-Swap Library

- **Deployment**: Simply load via LD_PRELOAD which is an environment variable.
- **Swap Handler**: Runs in a separate thread to prevent blocking inference.
- **Signals**: Uses POSIX signals to trigger swap operations.

RT-Swap Scheduler & Communication

- **Process**: Independent standalone process managing system-wide GPU memory.
- **IPC**: Uses named pipes for blocking I/O communication.
- **Integration**: Requires minimal code changes (e.g., $<$20 lines for Darknet).

Part VI

Experimental Evaluation

Experimental Setup

Hardware and Framework

- **Framework**: Darknet
- **Models**: YOLOv3, ResNet, ResNext, DenseNet.
- **Input Resolutions**: 256, 416, 608.
- **Metric**: Computation time measured by WCET over 1000 runs.

Comparison Methods

- **Base**: CUDA Unified Memory (UM) with LRU on-demand paging.
- **MO-VMM**: Standard object-level VMM API (Host-pinned memory).
- **MIN**: RT-Swap variant with fixed 2MB swap chunk size.
- **Pageable**: RT-Swap variant using pageable CPU memory.

SCHEDULABILITY RATIO: MULTI-DNN SCENARIOS

Evaluated 100 DNN task sets (up to 13 tasks) exceeding GPU memory capacity.

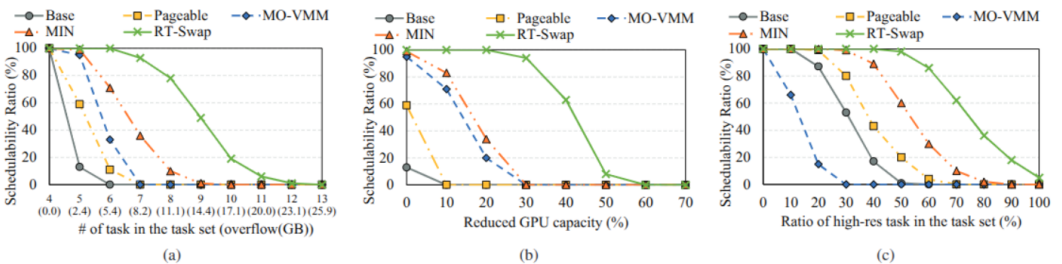


Fig. 7: Schedulability ratio comparison according to (a) the number of tasks in the task set, (b) reduced GPU capacity (%), and (c) ratio between low-resolution task and high-resolution task

Key Observations

- **Capacity Expansion**: RT-Swap remains schedulable even when demand exceeds GPU capacity by 96.2% (up to 25.9GB overflow).
- **Robustness**: Maintains high performance despite reduced GPU capacity and increased model sizes (resolutions).

Inference Latency: Predictability vs. Fluctuation

Comparison of observed latency distribution between Base and RT-Swap.

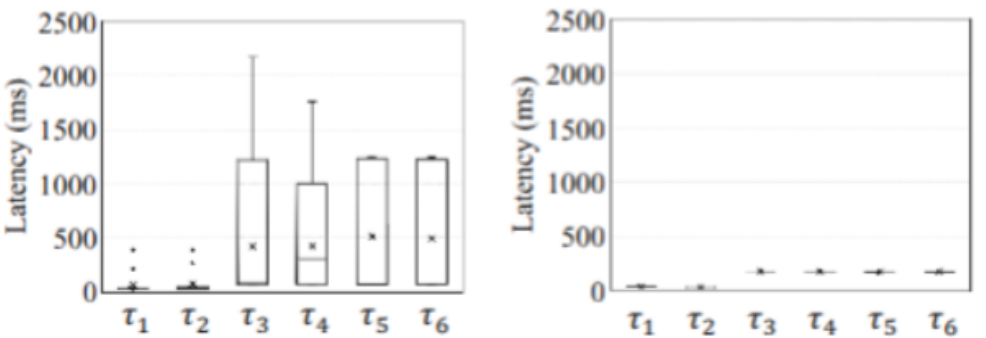


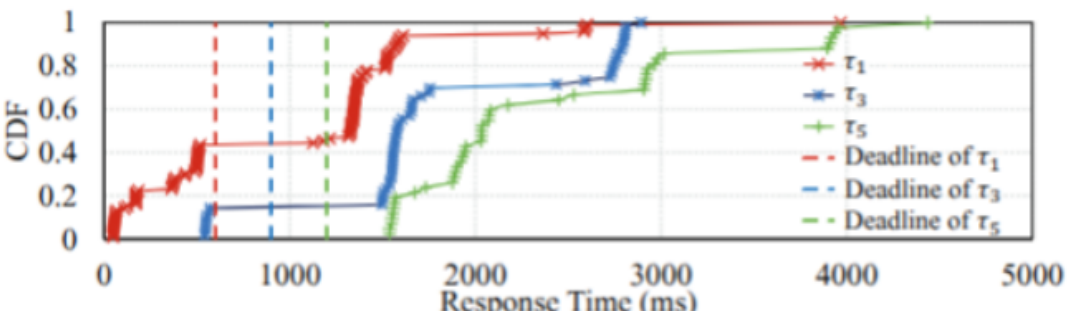
Fig. 8: Observed latency distribution using Base and RT-Swap

Performance Gap

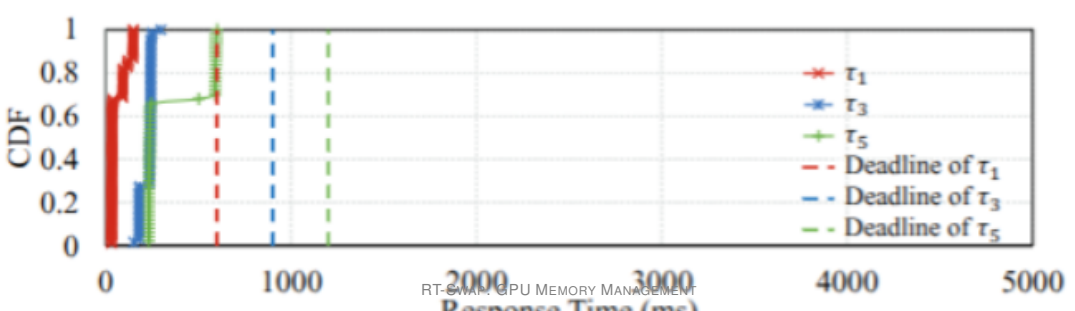
- **Base (UM)**: Latency fluctuates on the order of seconds due to unpredictable LRU page faults.
- **RT-Swap**: Guarantees consistent latency in milliseconds by limiting maximum swap volume.

Schedulability and Response Time

End-to-end response time CDF for the case study.



(a) Base



Runtime Overhead Analysis

RT-Swap maintains minimal overhead relative to DNN computation times.

TABLE V: Runtime overhead analysis for RT-Swap

Runtime overhead	Average	Min	Max
Initialization (one time) (μs)	120.5	80.1	194.2
Wrapper function calls (μs)	3.5	3.0	8.0
Scheduling (μs)	22.7	12.8	55.6
Extra CPU memory usage (MB)	14.9	14	15.7
Fragmentation (MB)	6.3	0	9.5

Efficiency Metrics

- **Scheduling**: Average overhead of $11.6\mu s$ for IPC and decisions.
- **VMM Wrapper**: Negligible delay of $3.5\mu s$ per allocation.
- **Memory Usage**: Minor GPU memory increase (avg. 6.3MB) due to bounded internal fragmentation.

Part VII

CONCLUSION

Conclusion

Summary

- Presented a new real-time memory management framework addressing the GPU memory wall .
- Concurrently ensures timely inference for multiple DNN tasks.
- Guarantees deterministic execution via swap-aware scheduling.
- Mitigates swapping overhead through task-level VMM and host-pinned memory.

Future Enhancement

- Develop layer-level GPU memory management for memory efficient scheduling.