



TÉCNICO
LISBOA



Enabling Incremental Checkpointing for GPU Workloads

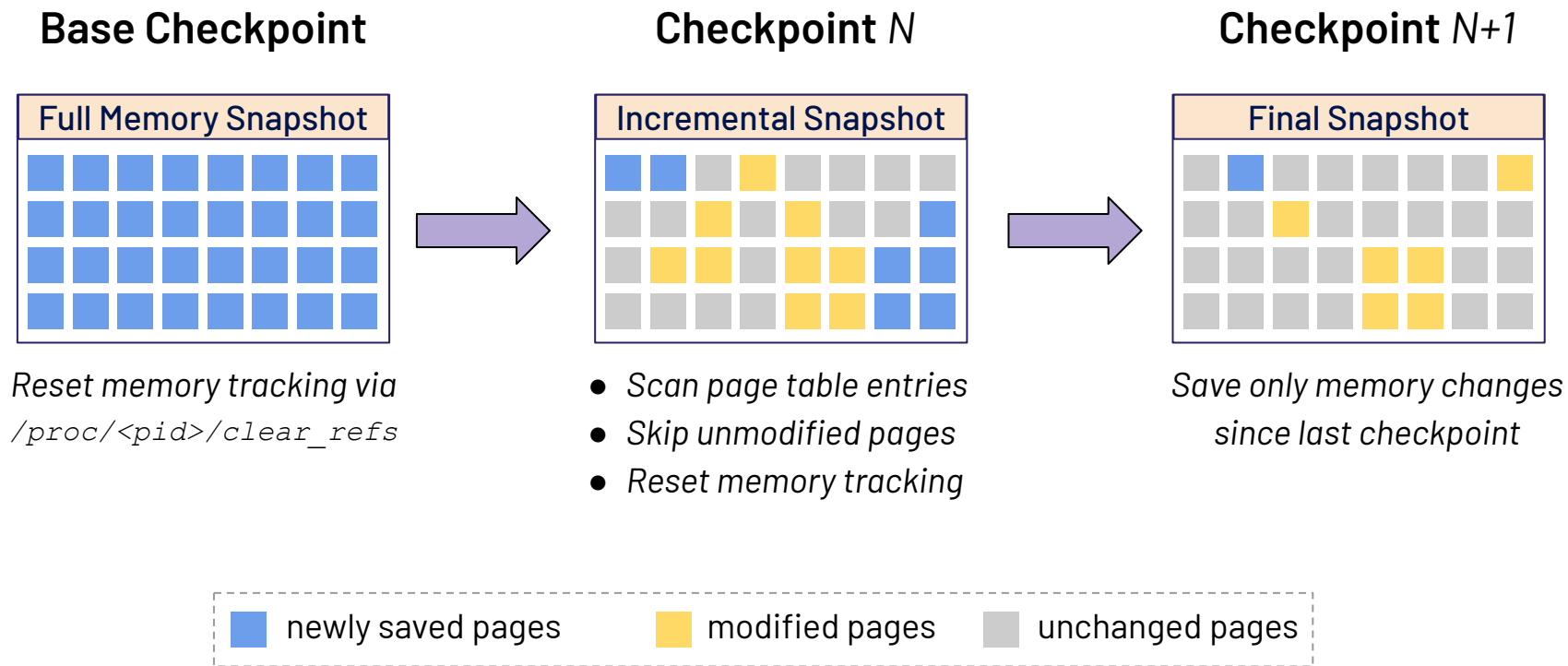


LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Radostin Stoyanov - PhD Student @ Scientific Computing
Supervisors: Prof. Rodrigo Bruno, Prof. Wes Armour

Incremental CRIU Snapshots



Why do we need incremental GPU snapshots?

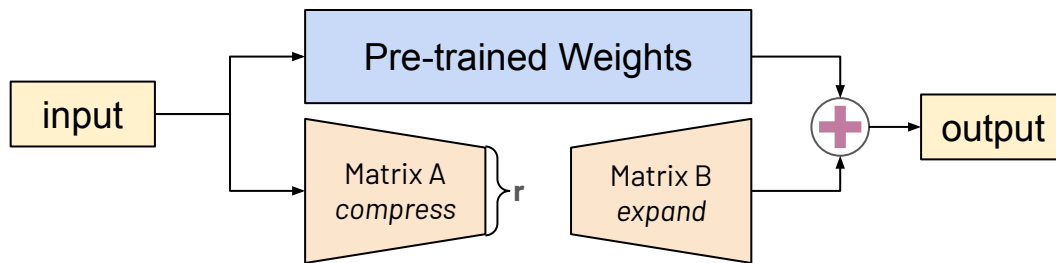
```
568 + /* Disable CUDA checkpointing with pre-dump */  
569 + if (stage == CR_PLUGIN_STAGE__PRE_DUMP) {  
570 +     plugin_disabled = true;  
571 +     return 0;  
572 + }  
573 +
```

[82b0342](#) (cuda: disable CUDA plugin for pre-dump)

Low-Rank Adaptation (LoRA)

Fine-tune a large model while keeping the pretrained model frozen

- Fine-tuning: A fully updated copy of the model for every use case (e.g., summarization, translation, question answering)
 - Storage and deployment are expensive
- LoRA: Learns a small set of additional parameters for each use case
 - Training optimizes only these parameters while retaining the original model

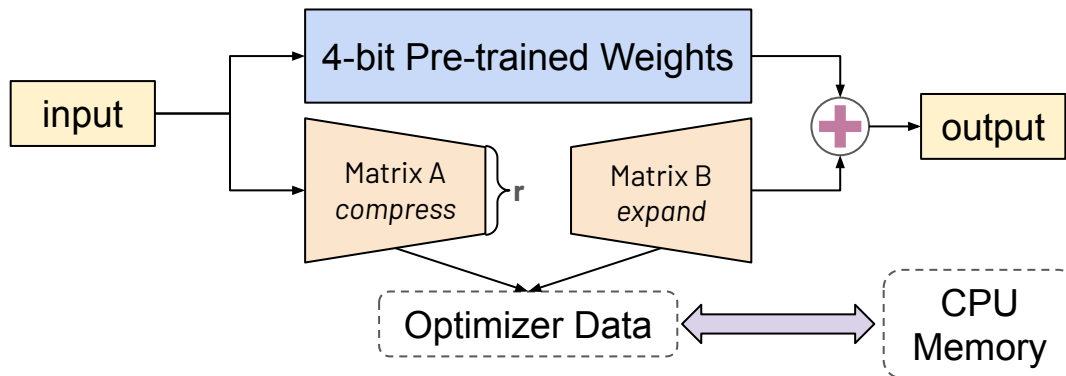


Only A and B contain trainable parameters

Quantized Low-Rank Adaptation (QLoRA)

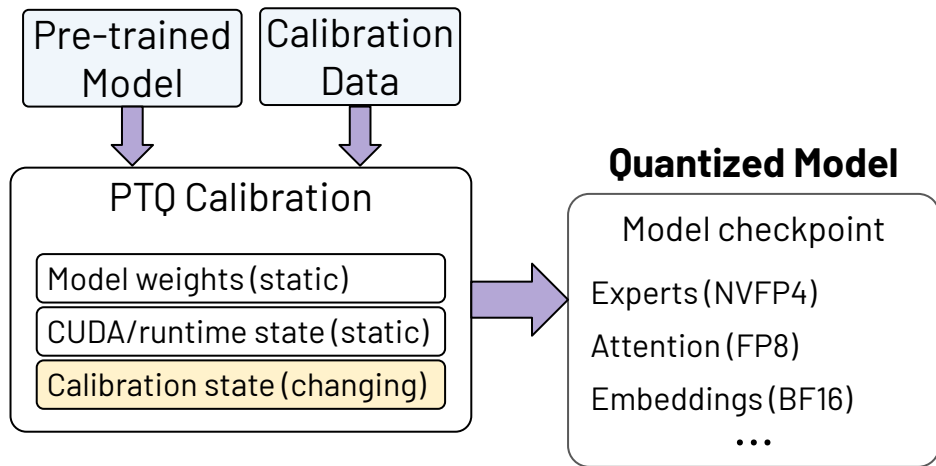
LoRA fine-tuning with a quantized frozen base model

- QLoRA: Stores the pretrained model in 4-bit precision (NF4)
 - Reducing GPU memory usage
- Paged optimizers improve memory management
 - Temporarily move optimizer data between GPU and CPU memory
 - Prevent GPU memory spikes from causing out-of-memory errors



Post-Training Quantization (PTQ)

Compress a trained model without full retraining



Incremental Snapshots

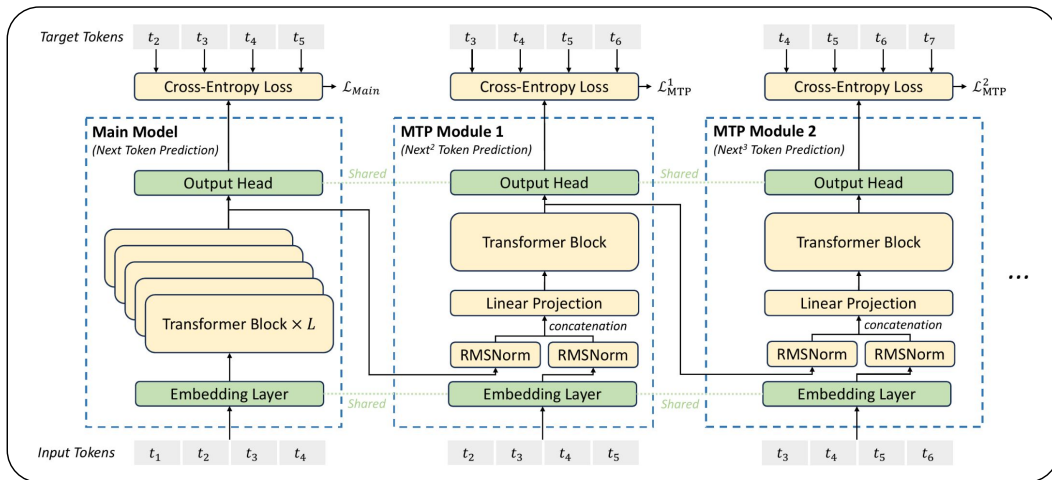
- Model state remains unchanged
- Calibration state evolves
- Activations change per batch

<https://github.com/NVIDIA/Model-Optimizer>

<https://github.com/vllm-project/llm-compressor>

Multi-Token Prediction (MTP)

Reducing checkpointing overhead during MTP training



DeepSeek-AI et al. (2024) "[DeepSeek-V3 Technical Report](#)"

Incremental Snapshots

- Main model checkpointed once
- Updated MTP states are captured incrementally

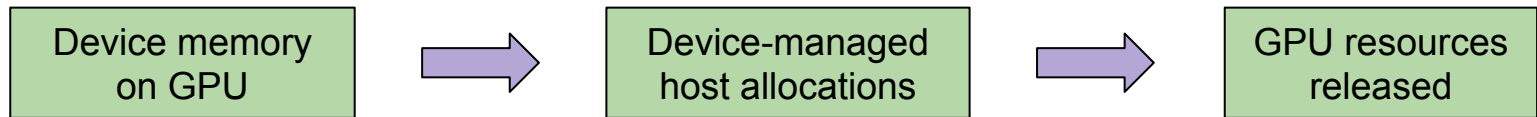
Challenges with incremental GPU checkpointing

How to efficiently identify unmodified GPU memory during checkpointing?

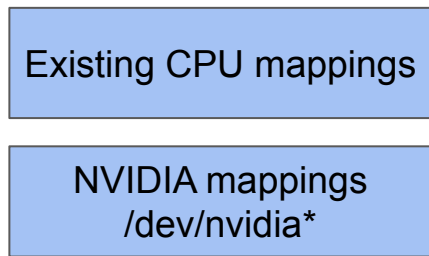
Identifying GPU Memory

Compare `/proc/<pid>/maps` before and after CUDA checkpointing

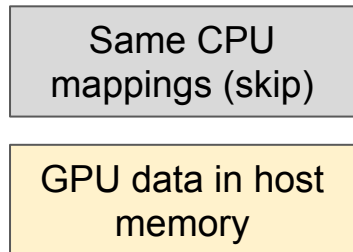
CUDA checkpoint



Scan pre-checkpoint

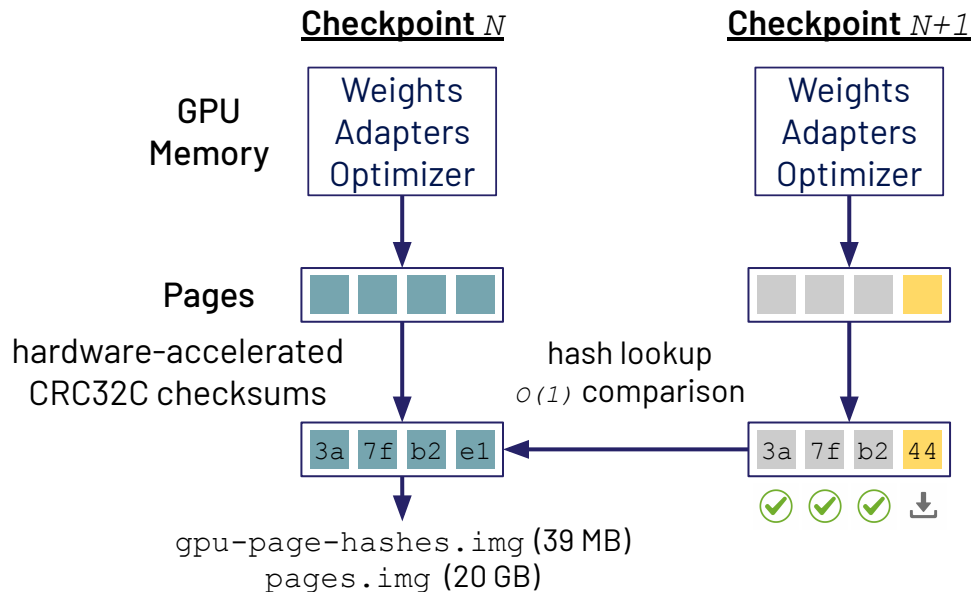


Identify GPU data



**Hash VMA
Pages**

Identifying Modified GPU Memory Pages



Llama 3 8B QLoRA: 89% pages skipped (20 GB $\rightarrow$ 2 GB)

Restoring GPU Memory from Parent Snapshots

Base Checkpoint CPU + GPU

A	B	C	D
E	F	G	H
I	J	K	L

12 pages in parent folder

Incremental Checkpoint

ref	ref	C'	ref
ref	ref	ref	H'
ref	ref	K'	M

3 modified pages + 1 new page

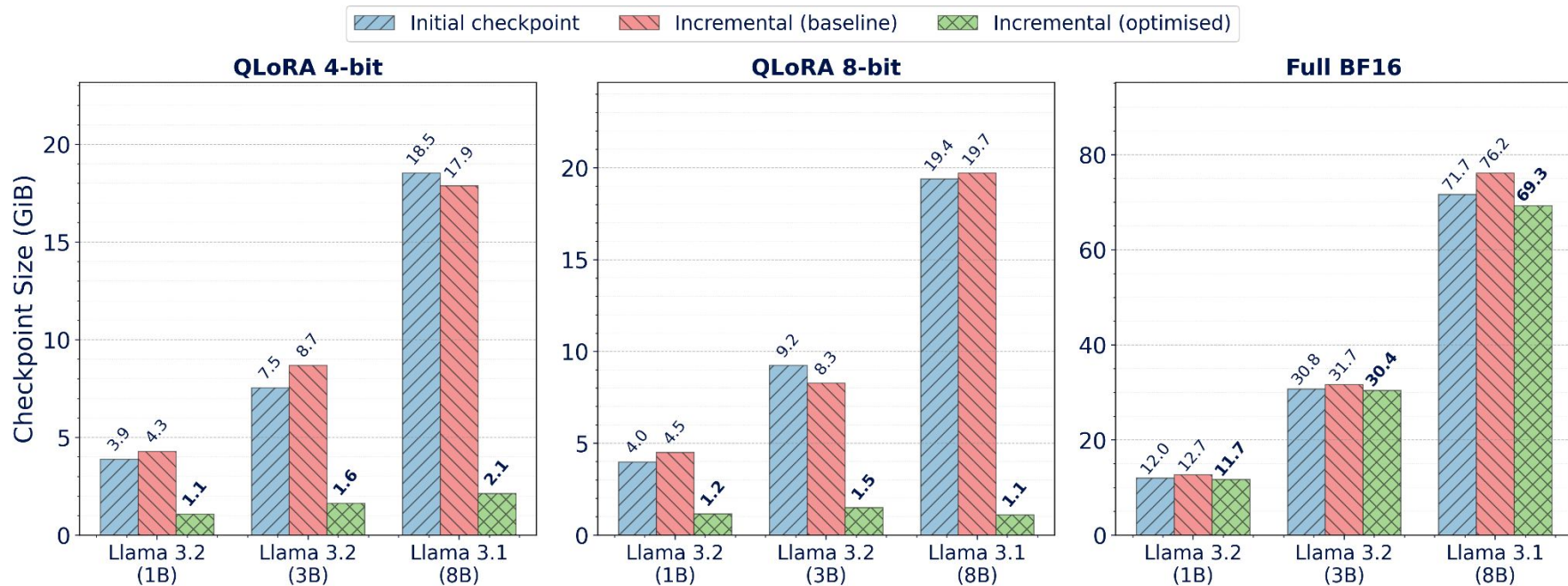
Restored Memory CPU + GPU

A	B	C'	D
E	F	G	H'
I	J	K'	M

8 parent + 4 current pages

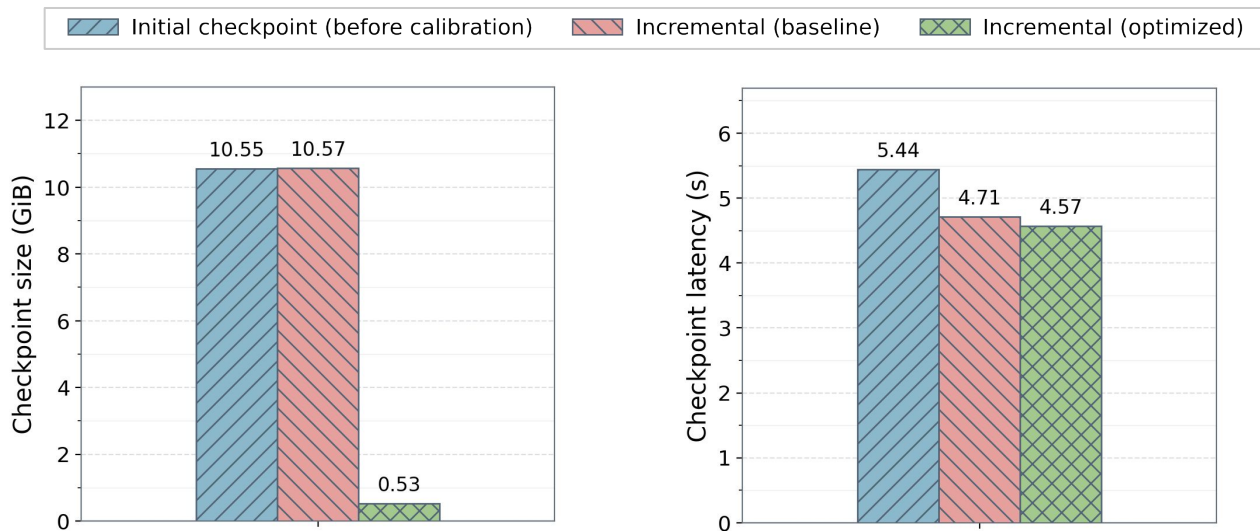
1. CRIU rebuilds host memory
2. CUDA restores complete device state

Preliminary Evaluation: Fine-Tuning



Baseline checkpoints have no optimisations
Optimised incremental checkpoints use LZ4 compression and deduplication
NVIDIA A100 SXM4 (80GB)

Preliminary Evaluation: Post-Training Quantization

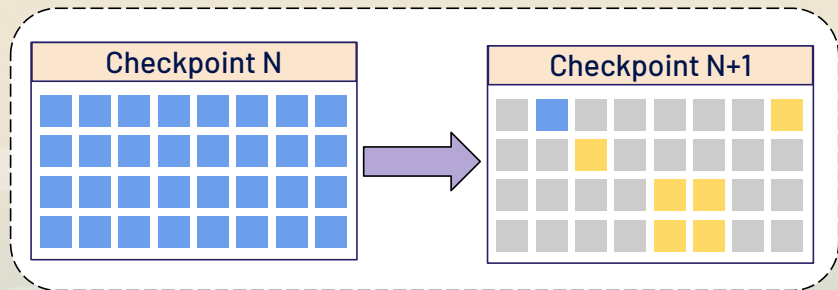


Checkpoint size & latency during Qwen3-4B calibration (quantize a BF16 model to FP8)

Incremental checkpoints reduce the snapshot from 10.57 GiB to 0.53 GiB

NVIDIA RTX 5090 (24GB)

Summary & Questions



Incremental GPU Checkpointing

- Avoids duplicated memory copies
- Page hashing skips unmodified memory
- Reduce checkpoint sizes by up to ~89%

