Optimizing Resource Utilization for Interactive GPU Workloads with Container Checkpointing

Viktória Spišaková, Radostin Stoyanov

Supervisor: Prof. Rodrigo Bruno, Prof. Wes Armour

Role of GPUs in Today's Computing

- GPU clusters have become the standard
- Competitive edge, accelerated time-to-results, and expanded computational capabilities
- Expensive and potentially more scarce

US tightens its grip on AI chip flows across the globe

By **Karen Freifeld**

January 13, 2025 6:29 PM GMT · Updated 15 days ago

https://www.reuters.com/technology/artificial-intelligence/us-tightens-its-grip-ai-chip-flows-across-globe-2025-01-13/

https://www.technologyreview.com/2025/01/24/1110526/china-deepseek-top-ai-despite-sanctions/

DeepSeek's success is even more remarkable given the constraints facing Chinese AI companies in the form of increasing US export controls on cutting-edge chips. But early evidence shows that these measures are not working as intended. Rather than weakening China's AI capabilities, the sanctions appear to be driving startups like DeepSeek to innovate in ways that prioritize efficiency, resource-pooling, and collaboration.

GPU Workload Types

- HPC workloads: Computational physics, chemistry, fluid dynamics, etc.
  - Characteristics
    - Finish and release resources
    - Classic way of using GPU to accelerate computations

- Interactive workloads: JupyterHubs (web UI), AI inference (chatbots)
  - Characteristics
    - Running (potentially) indefinite: idle GPUs for extended periods of time
    - New class of GPUs workloads leading to different challenges

Challenges with GPU Workloads

DEPARTMENT OF ENGINEERING SCIENCE

UNIVERSITY OF OXFORD

- HPC workloads
  - Fault-tolerance - In large data centers (or large computations) failures/errors happen all the time
- Interactive workloads
  - Demand for low-latency responses to user inputs
    - e.g. sub-millisecond latency with large models, human-computer interaction with interactive systems under *X* ms
  - Effective utilization of resources

Example from Practice: e-INFRA CZ

DEPARTMENT OF ENGINEERING SCIENCE

UNIVERSITY OF OXFORD

Multi-tenant, multi-purpose Kubernetes clusters for academic users in Czechia:

- 3072 CPU cores, 21 TiB memory, 41 GPU (A10, A40, A100, H100, L4)
- 450 users
- Containers provide reproducibility – crucial for research

Problem:

- Limited number of GPUs for growing number of users and workloads

HPC Applications

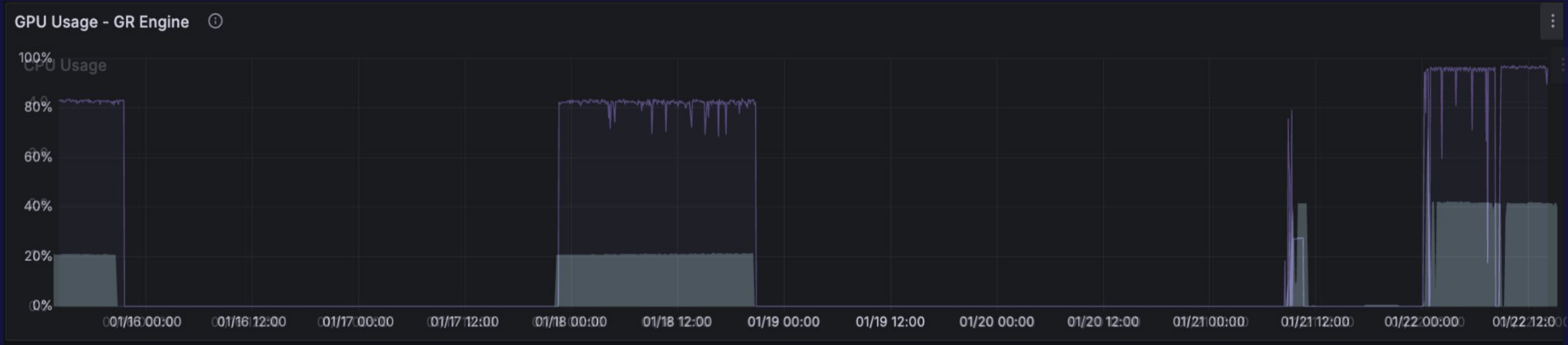
			Avg Util Last Hour	Avg Util Last Day	Running
receptorvshumanmaca-qkgwyakzvj-m259x	[redacted]	Whole GPU	100.00%	90.34%	1.40 weeks
foldifyobviousjuniper-dxqaaxefor-7pqxk	[redacted]	Whole GPU	100.00%	91.22%	1.72 weeks
foldifylegacyantagonist-nzaixhidxx-nkzdv	[redacted]	Whole GPU	90.16%	88.66%	1.82 weeks

Interactive workloads

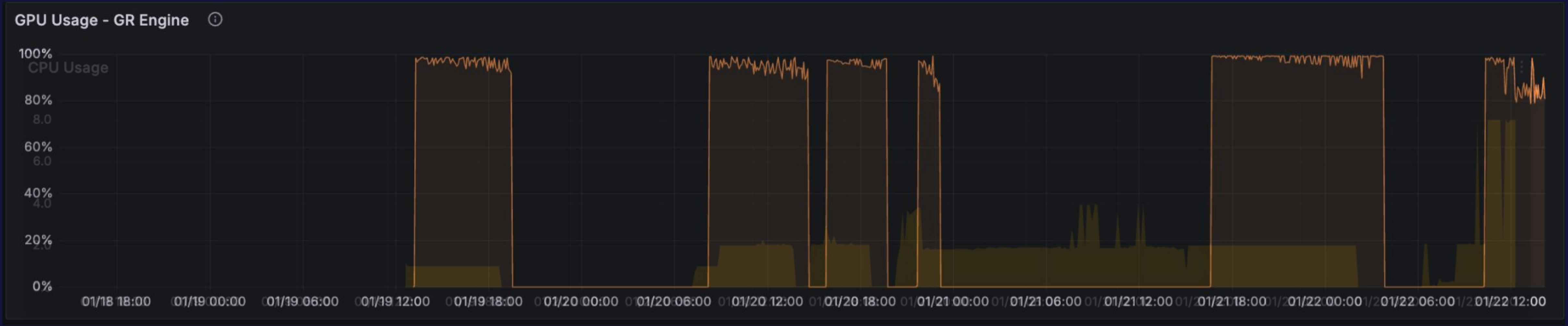
jupyter-[redacted]	jupyterhub-[redacted]	Whole GPU	97.29%	94.84%	1.95 weeks
jupyter-[redacted]--test	jupyterhub-[redacted]	Whole GPU	0.00%	10.97%	18.25 hours
jupyter-[redacted]--new-5fvillin	jupyterhub-[redacted]	MIG 1g_10gb	0.00%	0.00%	3.03 days
jupyter-[redacted]	jupyterhub-[redacted]	Whole GPU	0.00%	0.00%	1.83 days
jupyter-[redacted]-second	jupyterhub-[redacted]	Whole GPU	59.38%	33.32%	2.28 weeks
jupyter-[redacted]---47-50-55-2djupyter	jupyterhub-[redacted]	Whole GPU	31.62%	31.55%	1.28 weeks
jupyter-[redacted]	jupyterhub-[redacted]	Whole GPU	0.00%	0.21%	6.78 days

ollama-84ffc46bc7-7gj99	[redacted]	Whole GPU	1.31%	0.76%	3.08 days
ollama2-665bc69fb5-j5dr5	[redacted]	Whole GPU	0.93%	1.66%	1.60 days

JupyterHub User 1 - GPU utilization (purple) and CPU (grey)



JupyterHub User 2 - GPU utilization (orange) and CPU (yellow)



Low GPU Utilization – Optimization Techniques

Our observation:

- Low resource utilization
- Need for improving GPU utilization while preserving active sessions

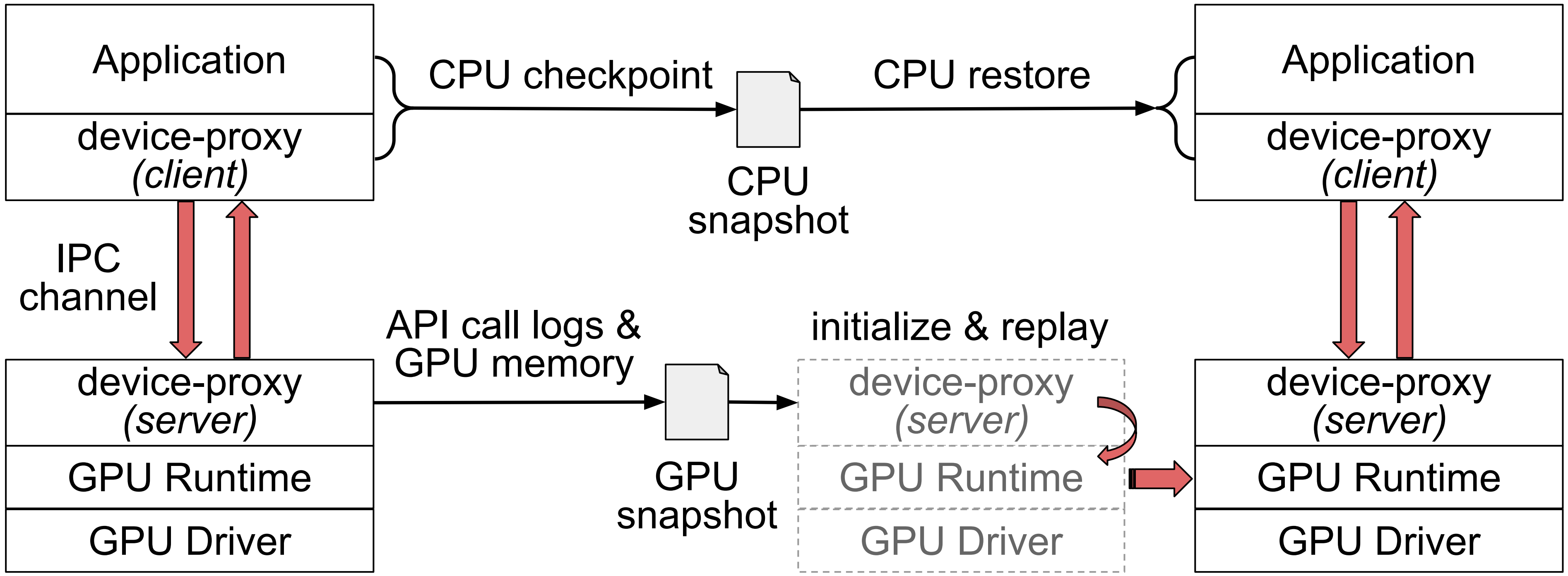
Existing solutions:

	Resource sharing (increased effectivity)	Resource reclaim (preemption)
Time slicing	✅	❌
Resource sharing (MIG)	✅	❌
GPU checkpoint/restore	✅	✅

Transparent GPU Checkpointing

Overview of GPU Checkpointing Methods

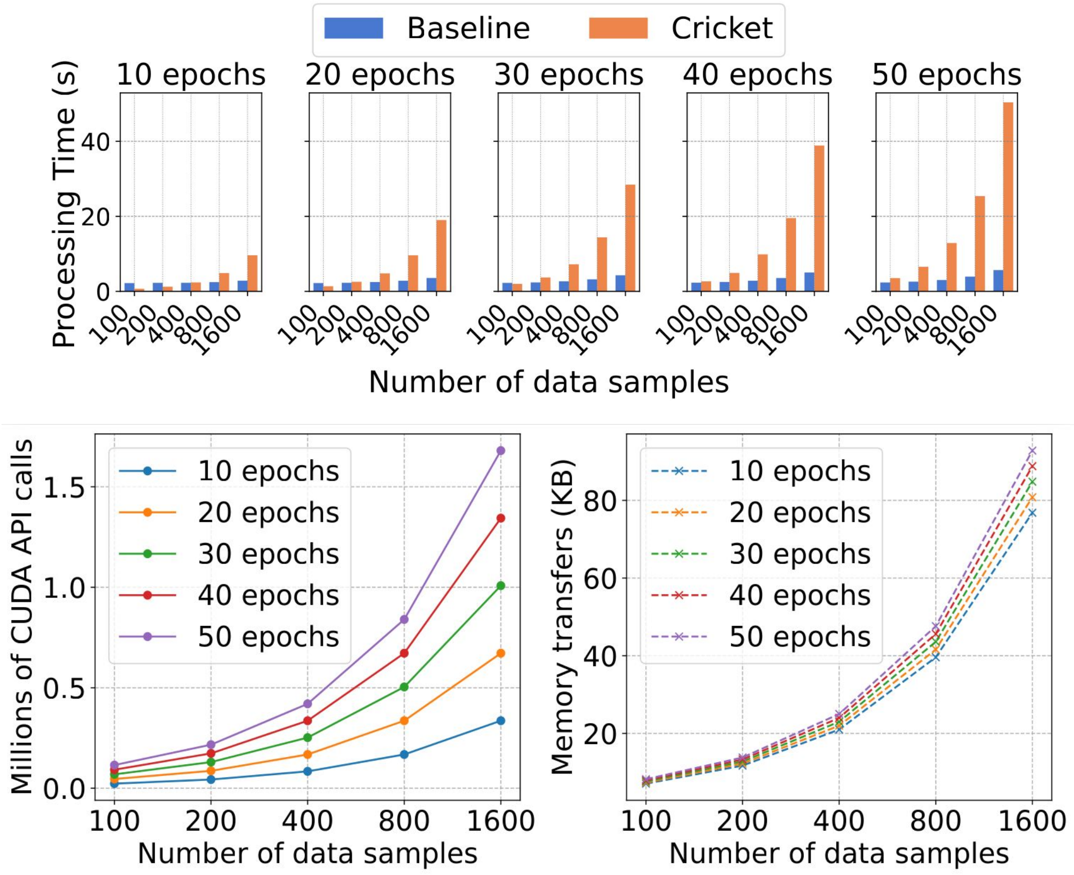
Checkpointing via API Interception



Just-In-Time Checkpointing: Low Cost Error Recovery from Deep Learning Training Failures, Tanmaey Gupta, et. al., 2024
Checkpoint/Restart for CUDA Kernels, Niklas Eiling, et. al., 2023
Singularity: Planet-Scale, Preemptive and Elastic Scheduling of AI Workloads, Dharma Shukla, et. al., 2022
Cricket: A virtualization layer for distributed execution of CUDA applications with checkpoint/restart support, Niklas Eiling, et. al., 2021

Challenges with API Interception

- Performance overhead for each API call
- Logs host-to-device (H2D) memory transfers
- GPU model-specific implementation
- Works only with dynamic linking
    - Requires building PyTorch from source



Overhead of API calls interception during neural network training with PyTorch

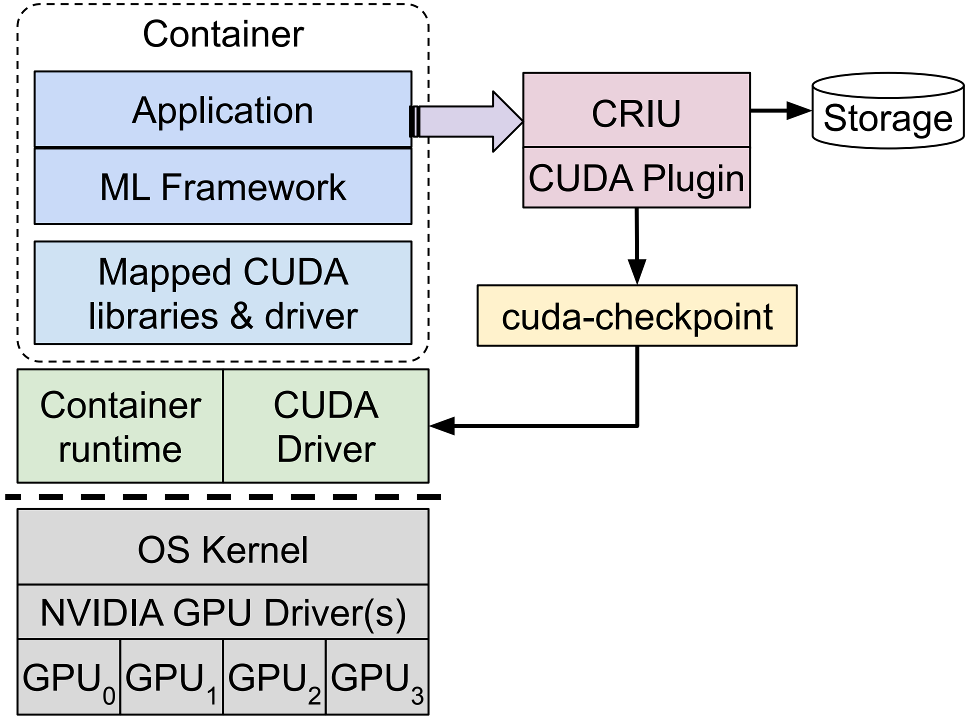
GPU Checkpointing with CRIU

Transparent and Unified GPU Snapshots

Checkpoint/Restore in Userspace

- Transparent checkpointing for Linux containers
- Implemented entirely in userspace
- Integrated with Docker, Podman and Kubernetes
- GPU support via **AMD** and **CUDA plugins**

CUDA Checkpoint/Restore



- Fully-transparent checkpoint/restore
- No API call interception
- No memory transfer logs
- Works for static & dynamic linking
- All GPU models are supported

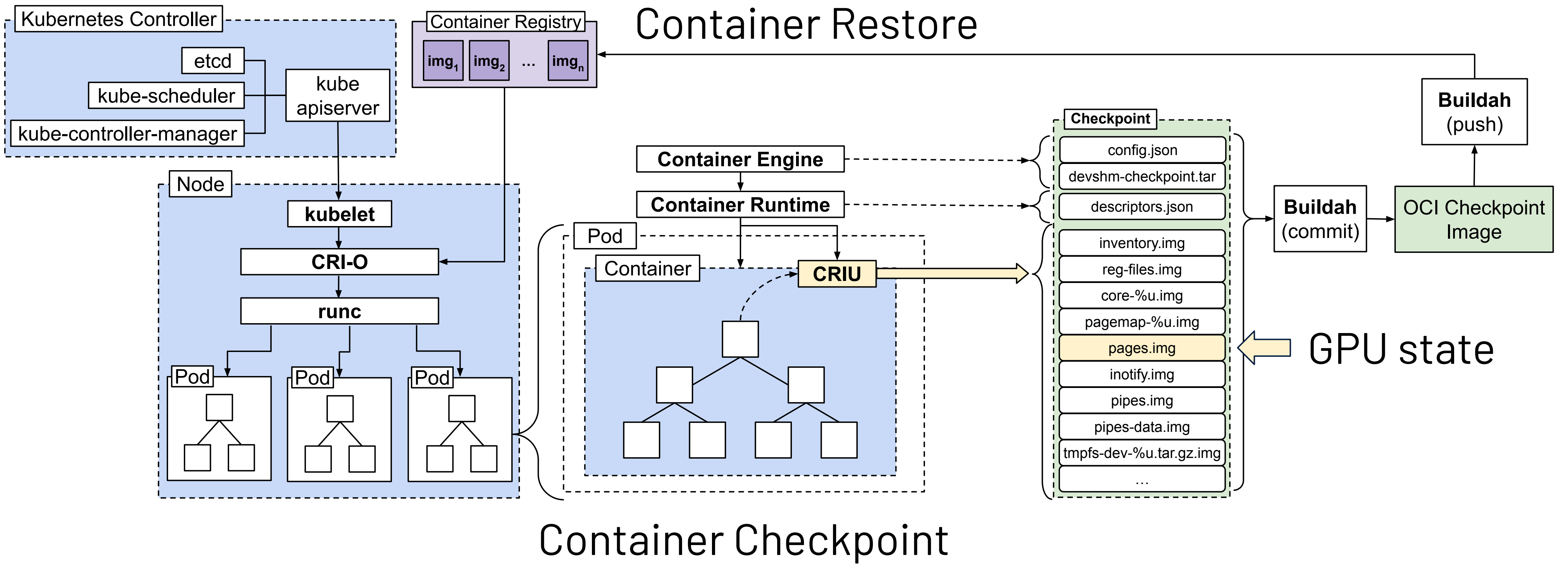
github.com/nvidia/cuda-checkpoint

github.com/checkpoint-restore/criu/tree/criu-dev/plugins/cuda

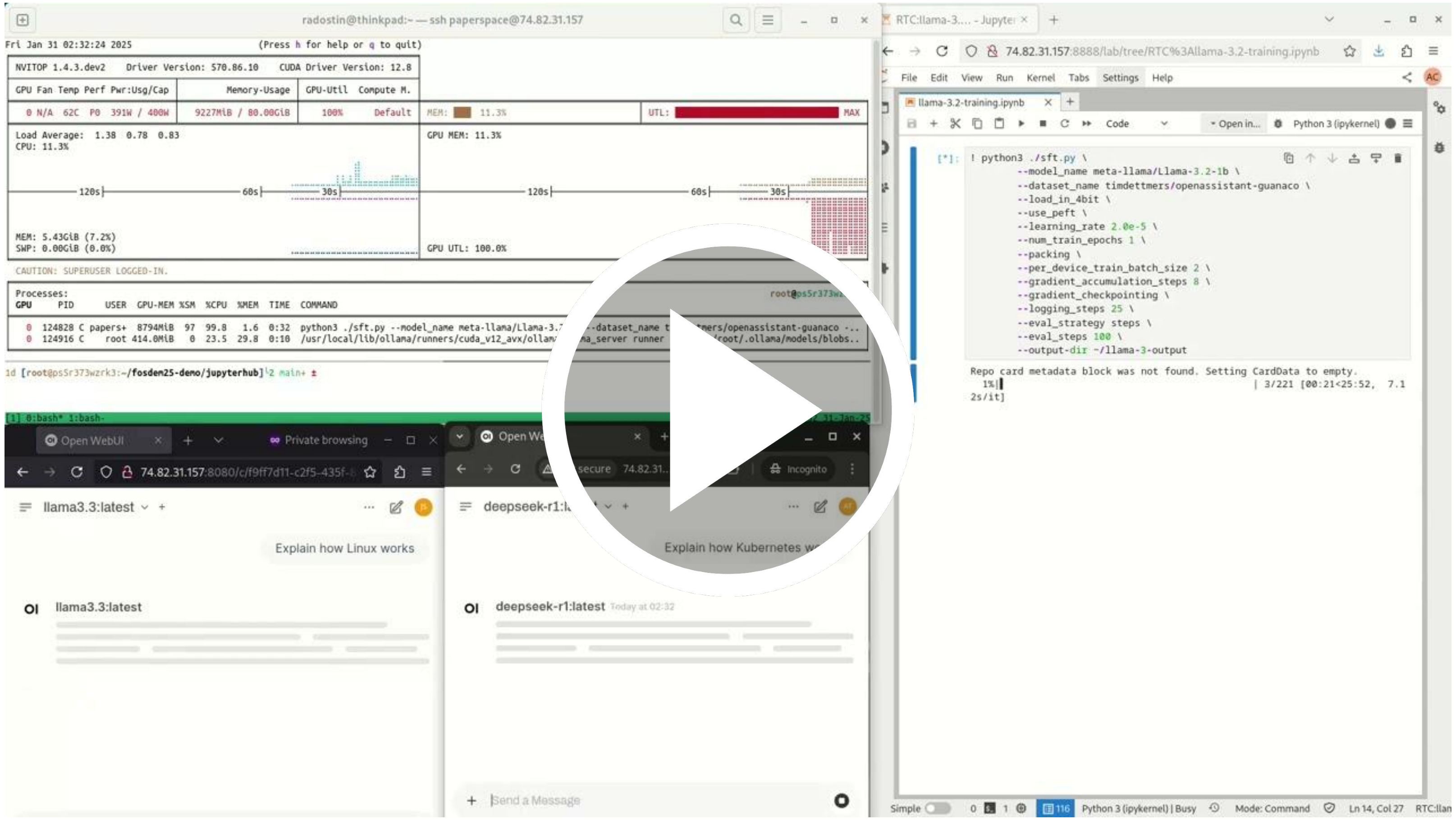
GPU Container Checkpointing

Optimizing Resource Utilization

Kubernetes Container Snapshots



kubernetes.io/blog/2022/12/05/forensic-container-checkpointing-alpha
developer.nvidia.com/blog/checkpointing-cuda-applications-with-criu


radostin@thinkpad:~ — ssh paperspace@74.82.31.157
Fri Jan 31 02:32:24 2025
(Press h for help or q to quit)
NVITOP 1.4.3.dev2 Driver Version: 570.86.10 CUDA Driver Version: 12.8
GPU Fan Temp Perf Pwr:Usg/Cap Memory-Usage GPU-Util Compute M.
0 N/A 62C P0 391W / 400W 9227MiB / 80.00GiB 100% Default
Load Average: 1.38 0.78 0.83
CPU: 11.3%
MEM: 5.43GiB (7.2%)
SWP: 0.00GiB (0.0%)
GPU MEM: 11.3%
GPU UTL: 100.0%
CAUTION: SUPERUSER LOGGED-IN.
Processes:
GPU PID USER GPU-MEM %SM %CPU %MEM TIME COMMAND
Open WebUI
Private browsing
74.82.31.157:8080
llama3.3:latest
Explain how Linux works
deepseek-r1:latest
Explain how Kubernetes w
Send a Message
llama-3.2-training.ipynb
74.82.31.157:8888/lab/tree/RTC%3Allama-3.2-training.ipynb
File Edit View Run Kernel Tabs Settings Help
Python 3 (ipykernel)
! python3 ./sft.py \
--model_name meta-llama/Llama-3.2-1b \
--dataset_name timdettmers/openassistant-guanaco \
--load_in_4bit \
--use_peft \
--learning_rate 2.0e-5 \
--num_train_epochs 1 \
--packing \
--per_device_train_batch_size 2 \
--gradient_accumulation_steps 8 \
--gradient_checkpointing \
--logging_steps 25 \
--eval_strategy steps \
--eval_steps 100 \
--output-dir ~/llama-3-output
Repo card metadata block was not found. Setting CardData to empty.
Python 3 (ipykernel) | Busy
Mode: Command
Ln 14, Col 27

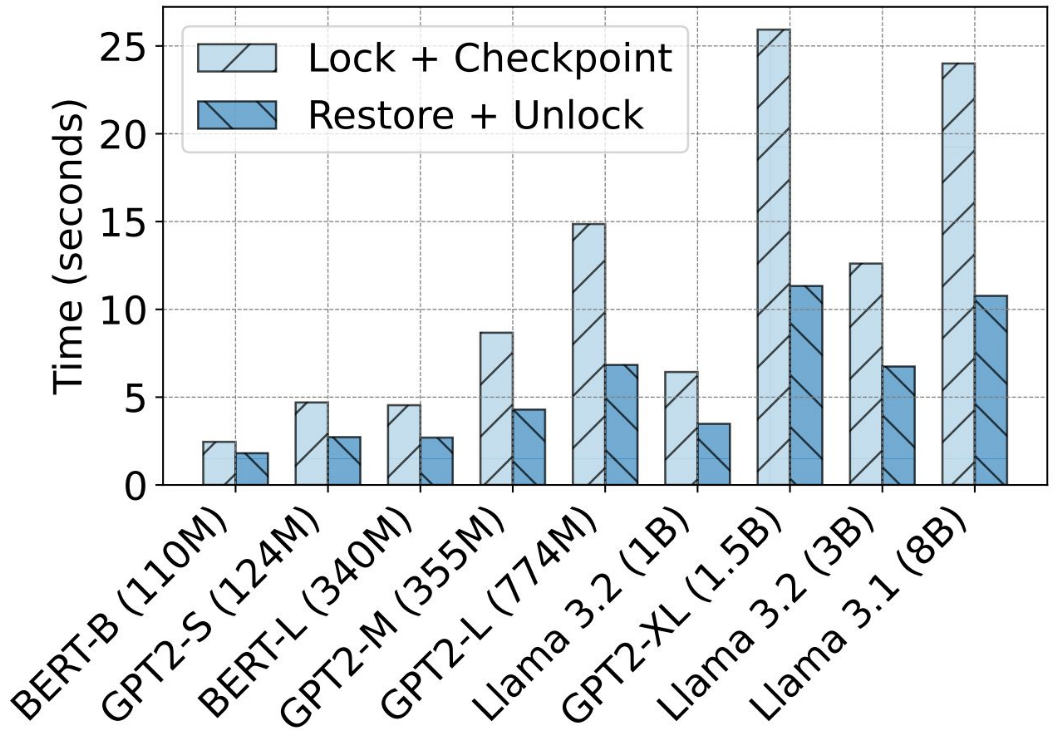
Evaluation Results

What factors determine checkpoint & restore latencies?

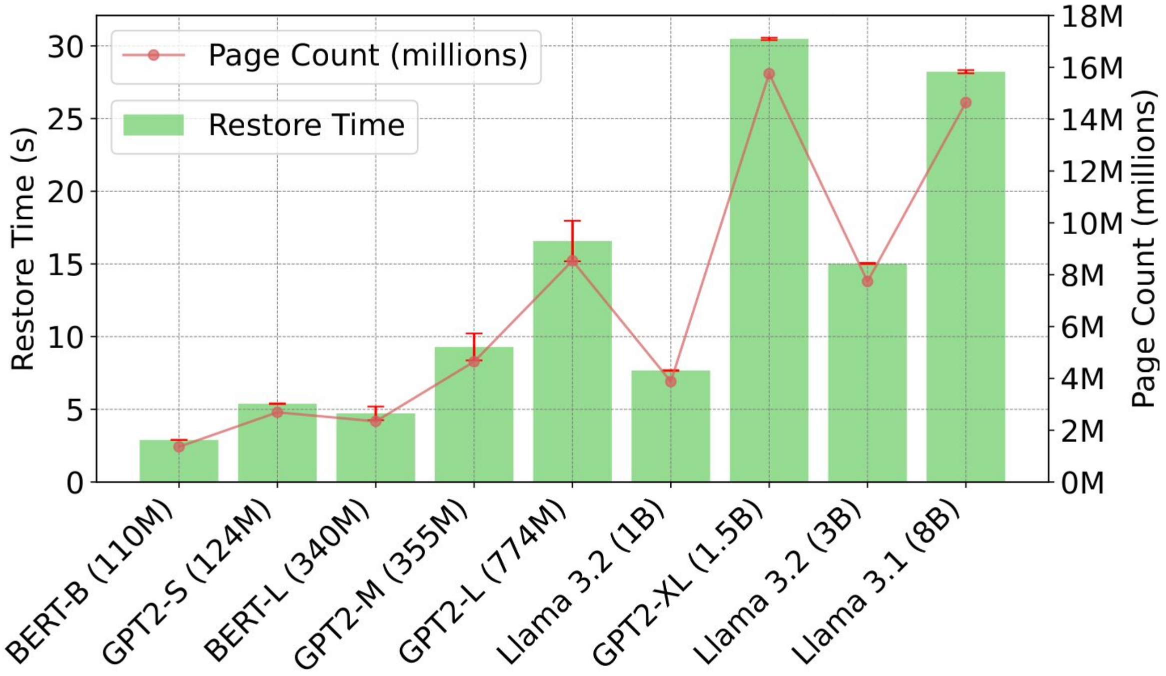
What are the scalability implications of using multiple devices?

Performance Evaluation

In-memory checkpoint/restore (A100 SXM4 80GB)

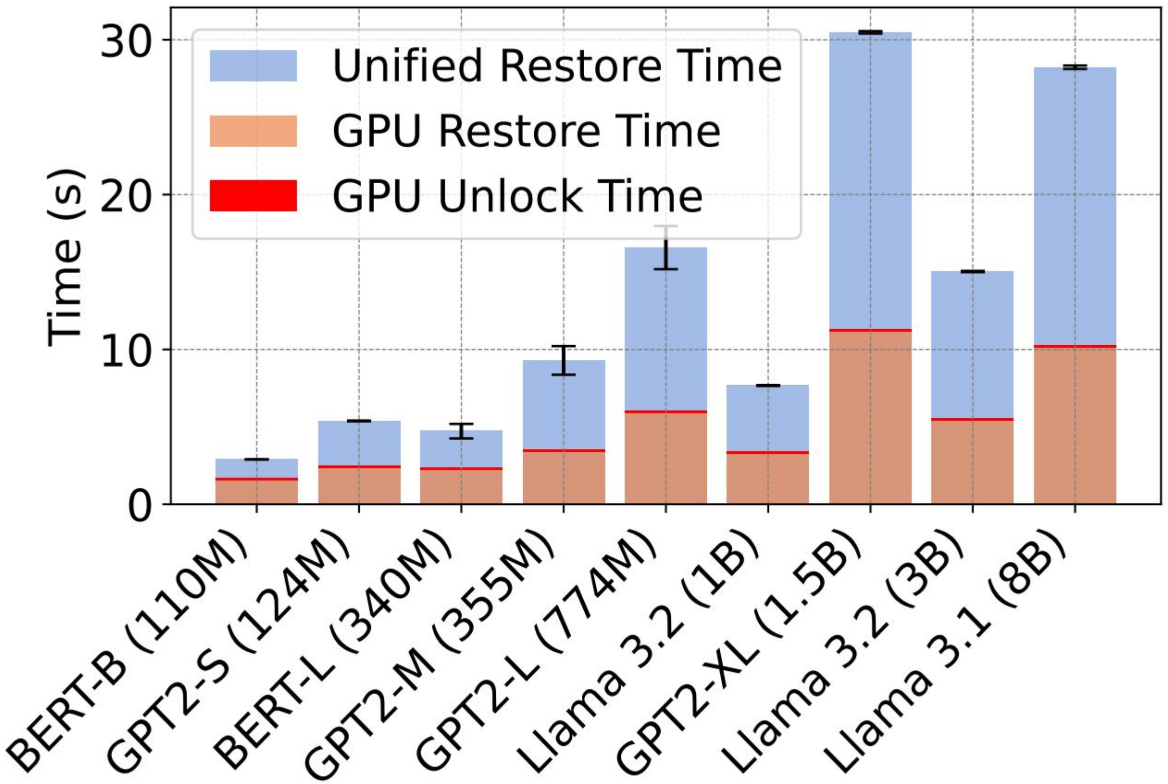


Restore from disk (H100 PCIe 5.0 80GB HBM3)



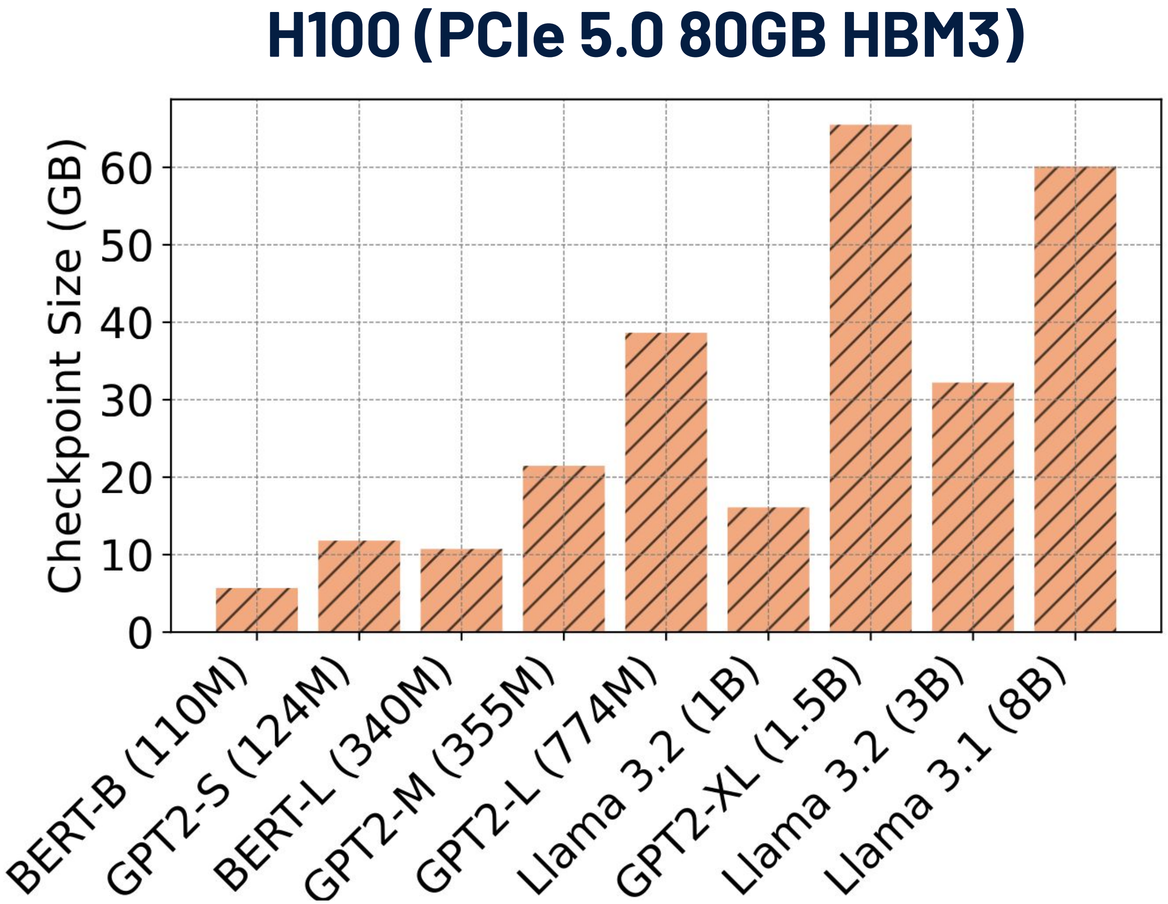
Restore Times

H100 (PCIe 5.0 80GB HBM3)



- Data-parallel training increases checkpoint size & restore times linearly
- Loading checkpoint data from disk to host memory can be expensive
- Lock/Unlock times are negligible

Checkpoint Size



Checkpoint size is determined by:

- Number of Parameters (weights)
- Parameter Precision (FP32, FP16, FP8)

GPU vs host memory:

- 90% for GPT 2 Small (124M)
- 97% for Llama 3.1 (8B)

Acknowledgements

- Andrei Vagin (Google)
- Jesus Ramos, Steven Gurfinkel (NVIDIA)
- Felix Kuehling, Rajneesh Bhardwaj, David Yat Sin, Ramesh Errabolu (AMD)
- Adrian Reber (Red Hat)
- Lukáš Hejtmánek (Masaryk University)

Summary & Conclusion

- Fully-transparent GPU snapshots
- CUDA & AMD plugins for CRIU
- Integrated with Kubernetes

github.com/nvidia/cuda-checkpoint

github.com/checkpoint-restore/criu