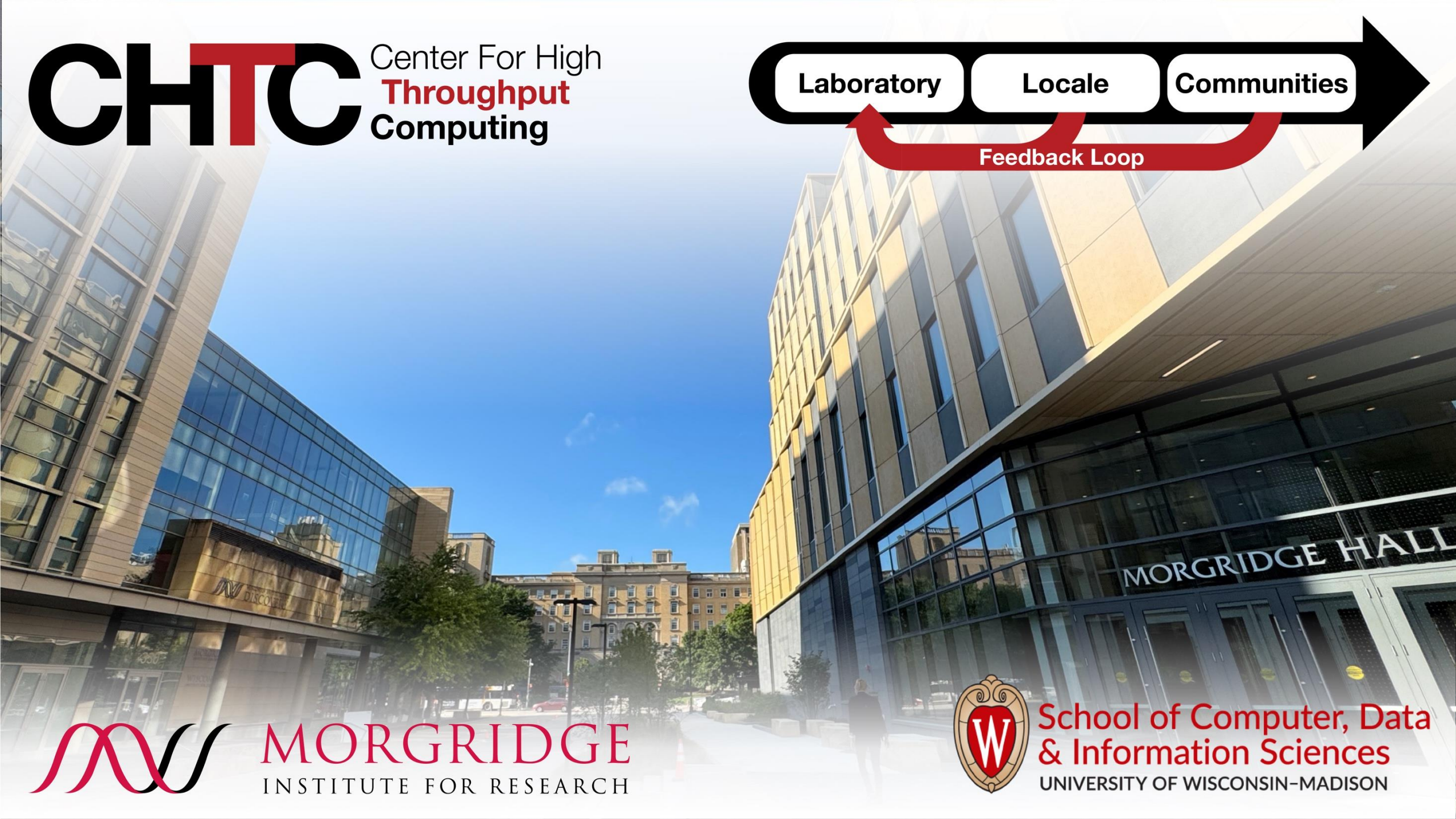
HTCondor GPU administration: Theory and (CHTC) Practice

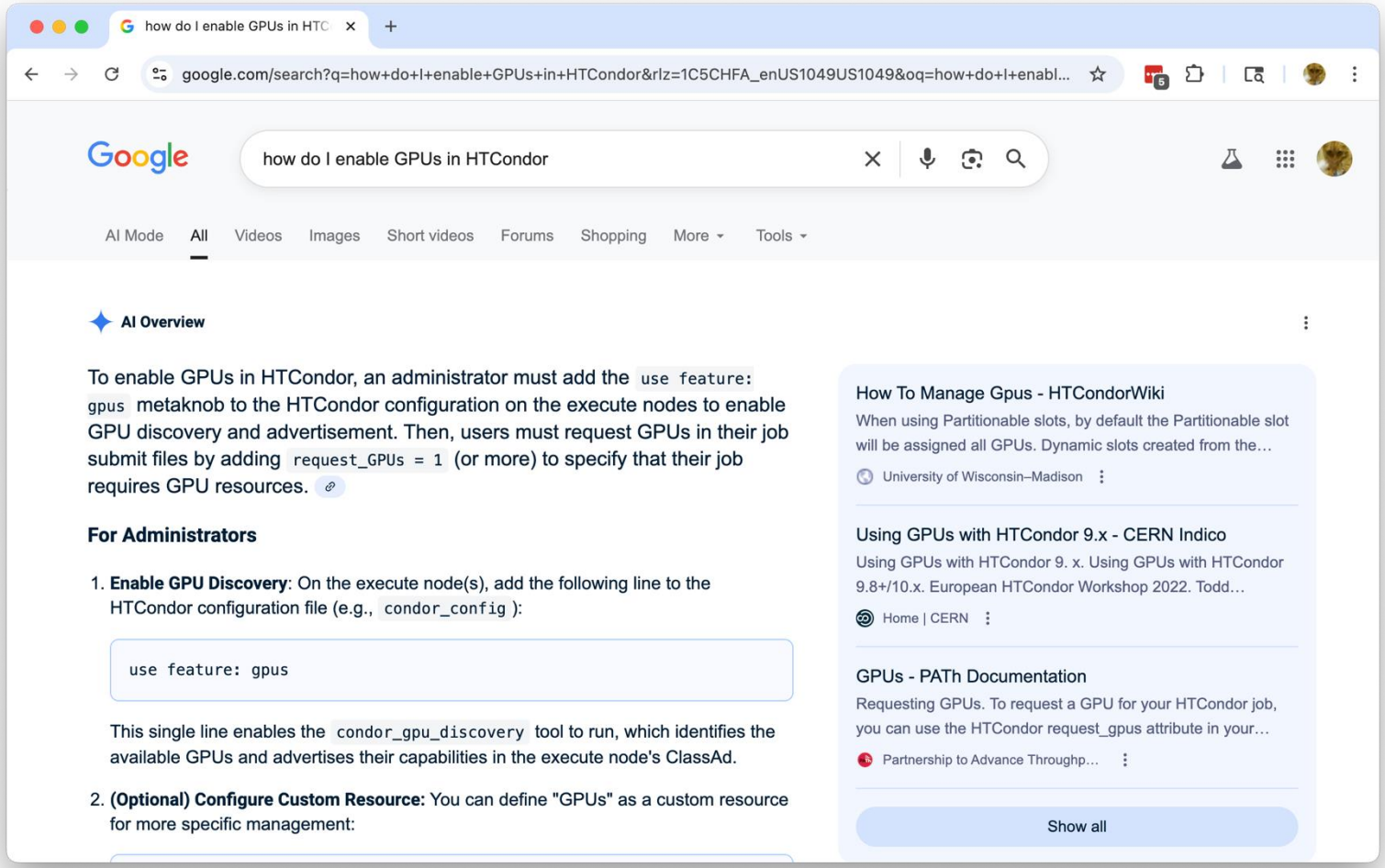
HTCondor European Workshop 2025

Brian Bockelman

MORGRIDGE
INSTITUTE FOR RESEARCH

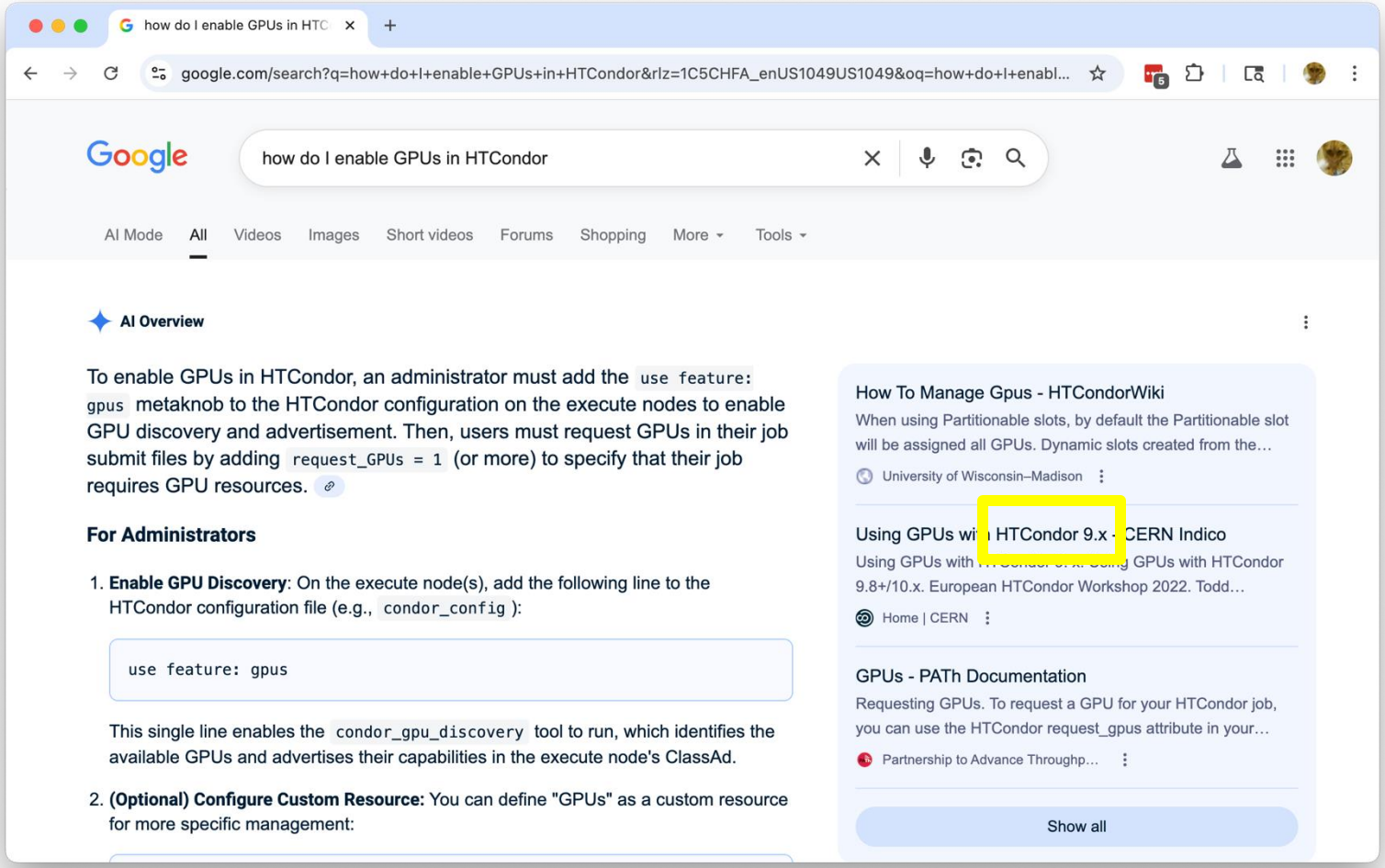

CHTC
Center For High
Throughput
Computing
Laboratory
Locale
Communities
Feedback Loop
MORGRIDGE HALL
MORGRIDGE
INSTITUTE FOR RESEARCH
School of Computer, Data
& Information Sciences
UNIVERSITY OF WISCONSIN-MADISON

GPU Basics in HTCondor

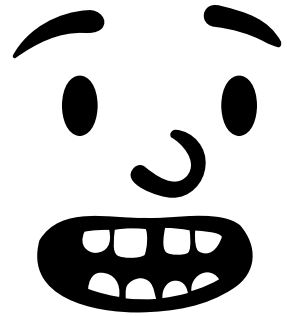


- Behold the glory of AI!

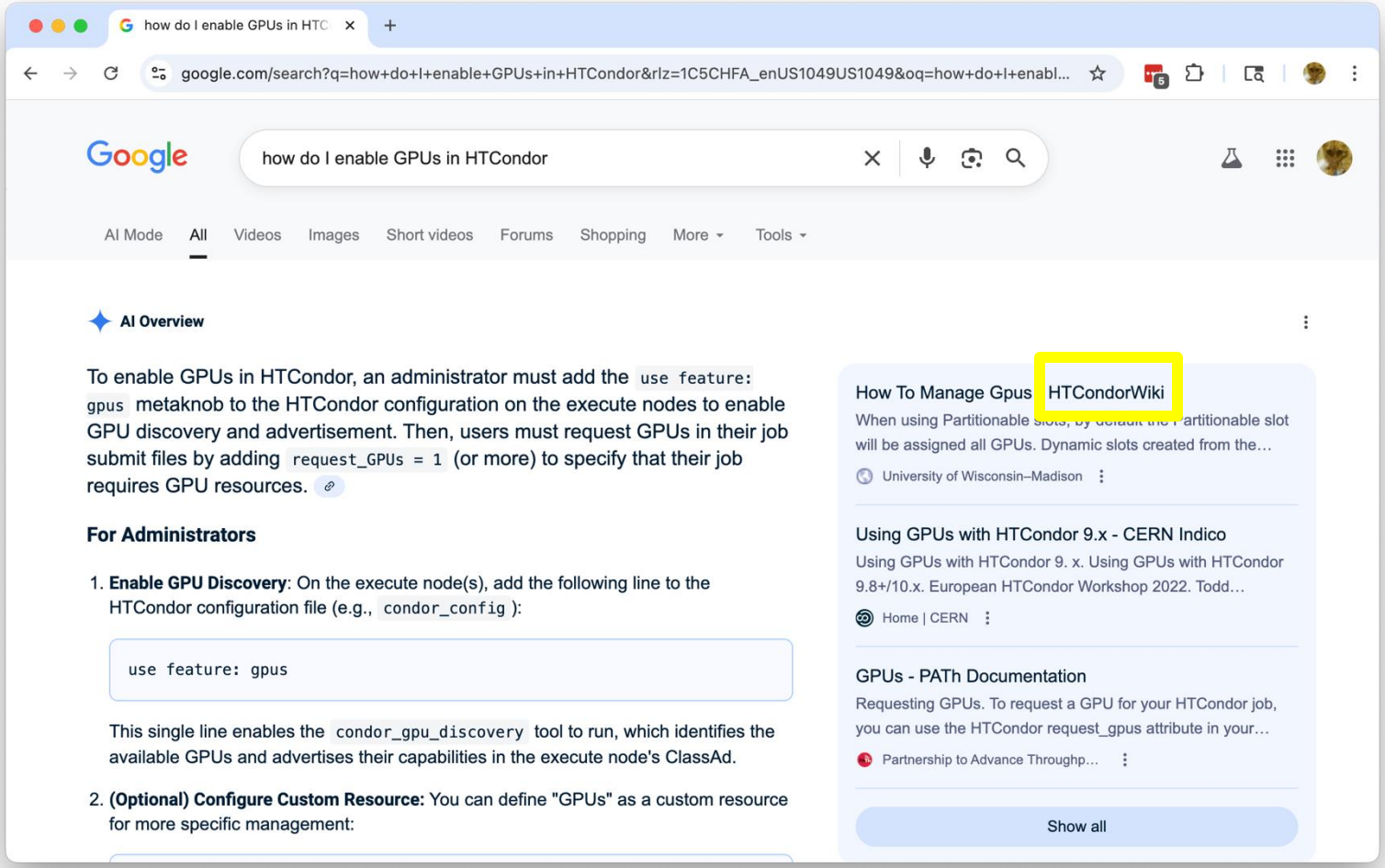
GPU Basics in HTCondor



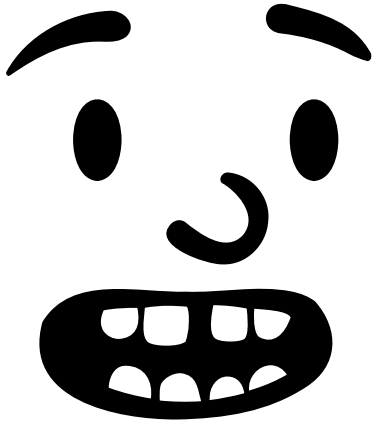
- Behold the glory of AI!
  - But HTCondor 9.x?



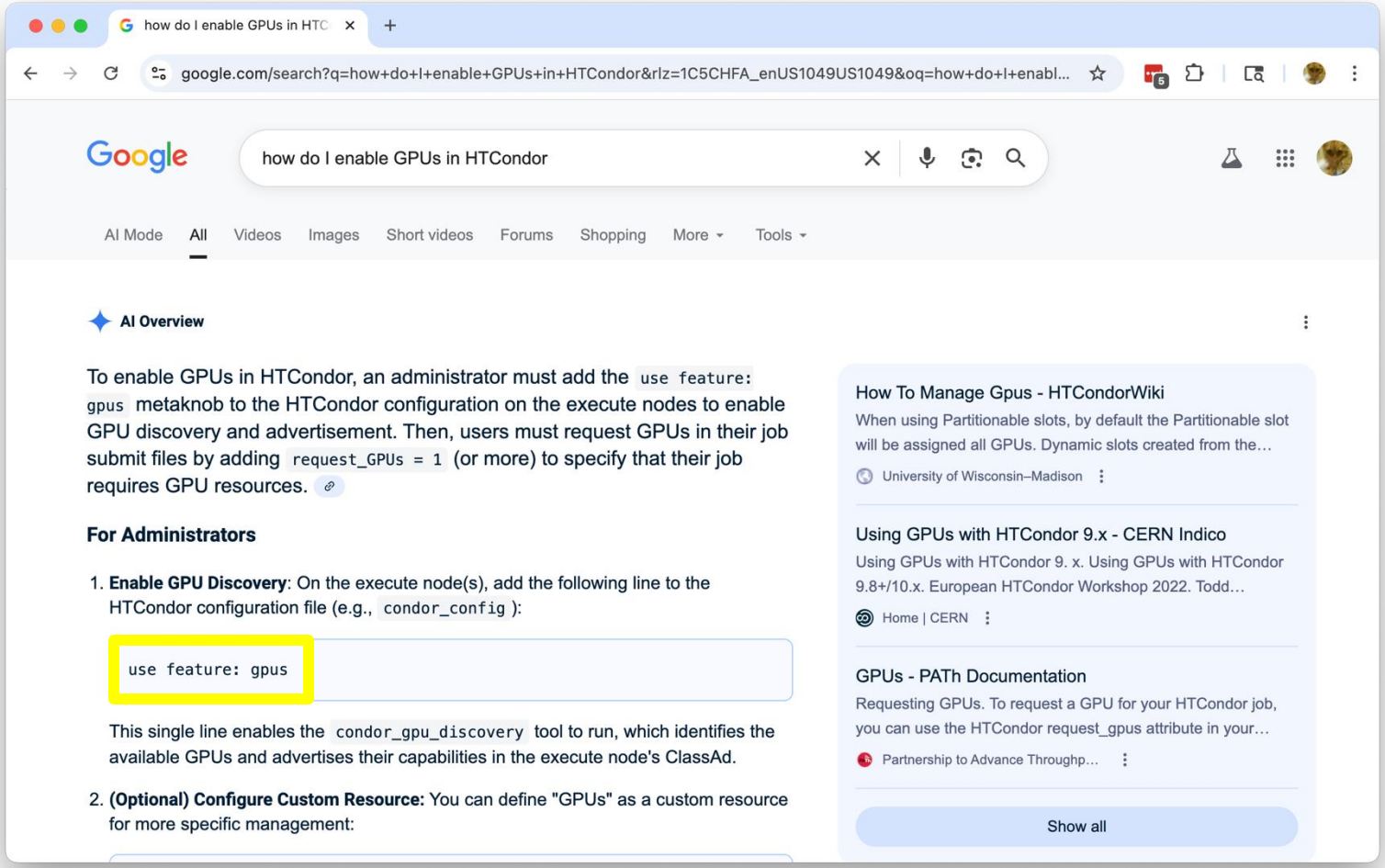
GPU Basics in HTCondor



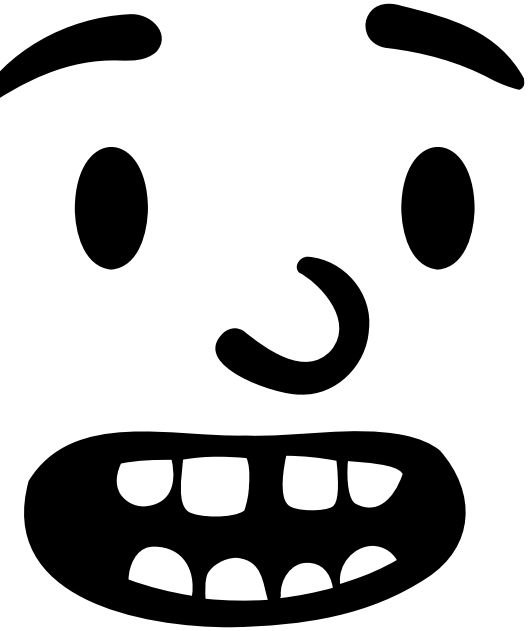
- Behold the glory of AI!
  - But HTCondor 9.x?
  - HTCondor Wiki? Don't use that!



GPU Basics in HTCondor



- Behold the glory of AI!
  - But HTCondor 9.x?
  - HTCondor Wiki? Don't use that!
  - Is this even true?!?!



To Detect GPUs in an EP

- AI didn't *completely* lie.
- Each EP has a "bag" of resources. By enabling the GPU feature, local GPUs are added to the bag.

use feature : GPU

- And similarly, you can request a GPU in the job description language:

request_GPUs = 1

- The job will startup with the environment variables – and, in a new enough HTCondor install, the per-GPU protections – for a basic CUDA job to run (inside or outside a container).

Question: What could possibly go wrong?

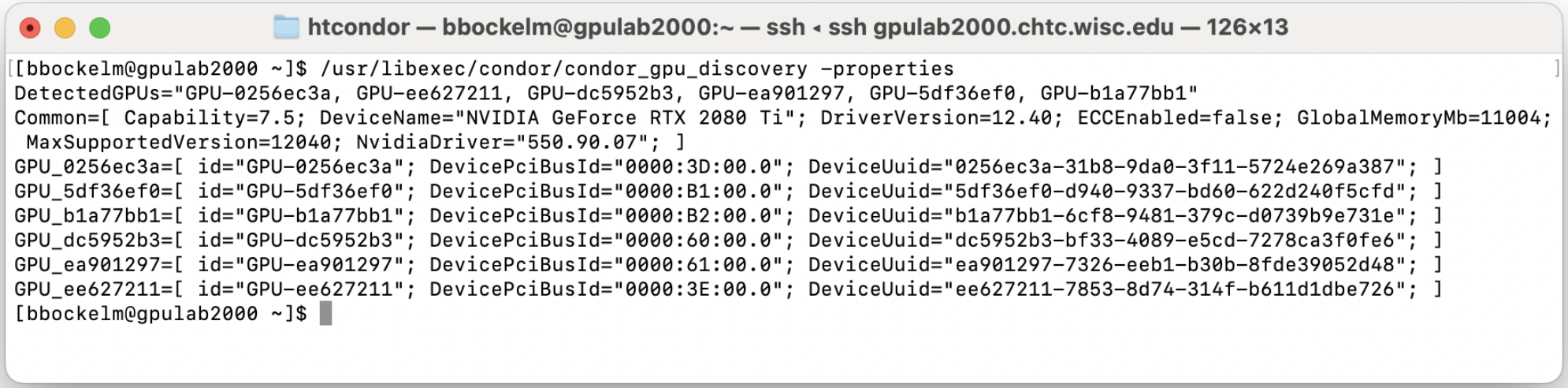
This talk is about all the considerations for GPU management *after* you get the "Hello World" job working!

The rest of this talk: What Could Possibly Go Wrong?

- Between running the first GPU-enabled job to building out a happy user base for your site, what do you need to do?
- This talk covers the real-world experience from CHTC in running a few hundred GPU devices locally across the past 8 years.
- GPUs are a whole new world! Things to consider:
  1. GPU cost != CPU cost
  2. GPUs are heterogeneous
  3. GPUs quantize poorly
  4. GPUs fall off the bus
- And that, my friends, is the rest of my talk!

First, a side-note

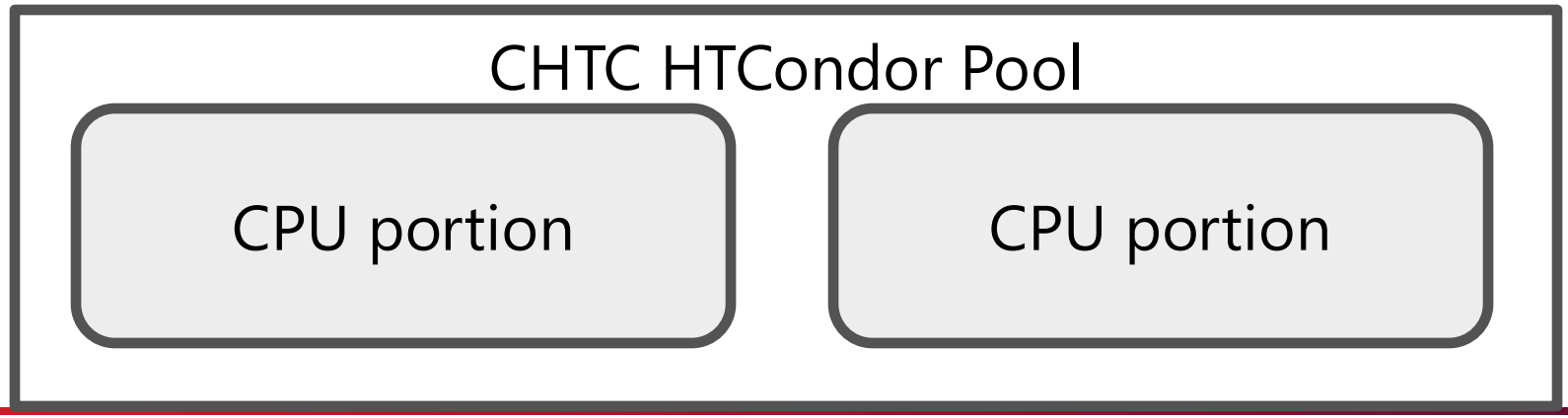
- Everything HTCondor knows about your GPUs comes from the output of condor_gpu_discovery.
- If in doubt, give it a try and make sure the results look sane!



How much does a GPU cost?

SLOT_WEIGHT = Cpus

- By default, the “cost” for a job is equal to cores (or SMT) requested.
  - If you set “request_cpus = 1” then the *cost* to your fairshare is 1.
  - A nice, new core on an EP may be worth $500 (capital). An old, smelly core may be worth $50.
  - A GPU? That new Blackwell device may run >>$25,000.
- Keeping the default “fairshare cost” of Cpus may undercharge by 100x!
- CHTC approach:
  - Partition the pool according to resource type, charge accordingly. Basically, separate universes!



CHTC setup

Does
SLOT_WEIGHT = GPUs
make any sense?

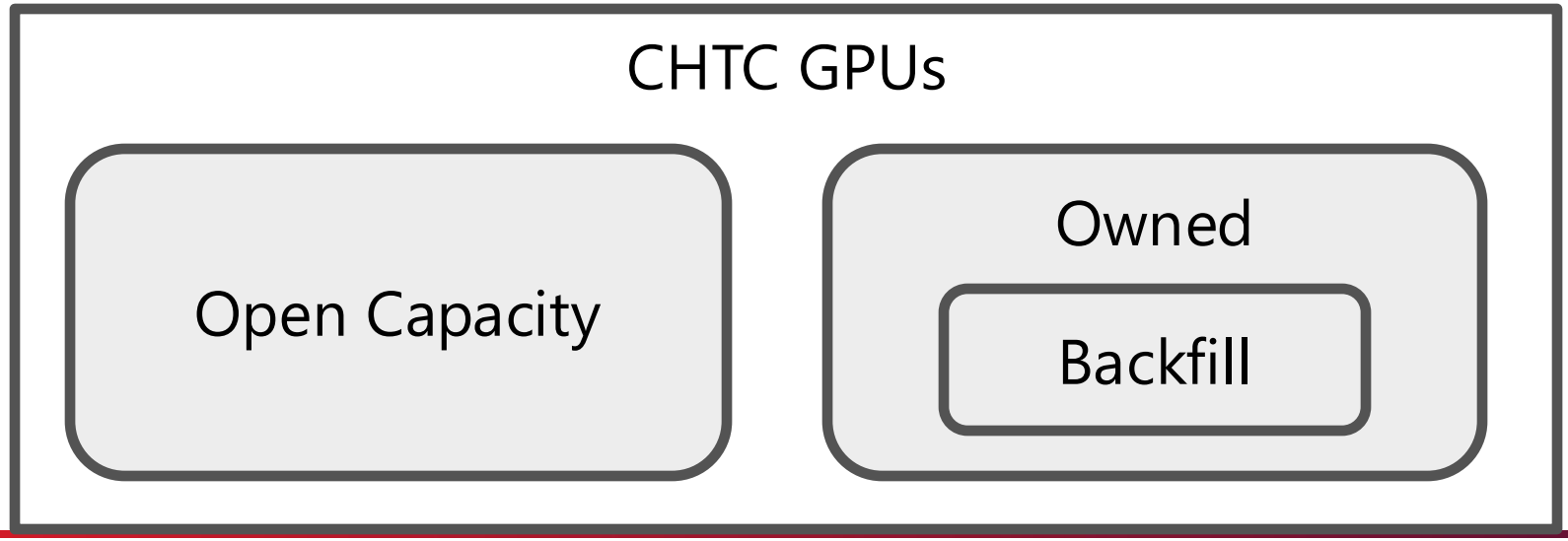
- We split out the GPU & CPU portions by running two negotiators:

```
NEGOTIATOR_SLOT_CONSTRAINT = IsGPULabSlot =!= true
DAEMON_LIST                          = $(DAEMON_LIST) GPULAB_NEGOTIATOR
GPULAB_NEGOTIATOR                    = $(NEGOTIATOR)
GPULAB_NEGOTIATOR_ARGS               = -local-name GPULAB_NEGOTIATOR
GPULAB_NEGOTIATOR.NEGOTIATOR_NAME    = gpulab_negotiator
GPULAB_NEGOTIATOR.NEGOTIATOR_JOB_CONSTRAINT  = WantGPULab ?: false
GPULAB_NEGOTIATOR.NEGOTIATOR_SLOT_CONSTRAINT = IsGPULabSlot ?: false
GPULAB_NEGOTIATOR.SPOOL              = /var/lib/condor/gpulab_spool
```

- Conceptual question: Do you have “old stinky” CPU-only jobs block the “shiny fancy” GPUs from being used?

Next layer of detail: “open capacity” vs “owned”

- The GPU policy comes in two flavors:
  - “open capacity”: Purchased by the university, fair-shared across all the groups. No preemption.
  - “owned”: Purchased by individual research groups; their jobs should be scheduled *quickly* on their hardware.
    - This brings up a third capacity type: “backfill”. Created from idle owned GPUs, jobs running on backfill will be preempted when the owner returns.
    - N.b. – the concept of “ownership” dates way back to HTCondor’s scavenging days: idle desktops had owners.

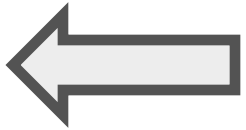


Creating Backfill Slots

- HTCondor has *long* supported the idea of multiple types of slots.
  - Newer is the idea of “backfill” slots that can overlap with existing resources.
  - **Goal**: If a slot is activated that needs the resources in a backfill slot, the EP will preempt the backfill slot.

use FEATURE : GPUs

use FEATURE : PartitionableSlot(1, 100%)



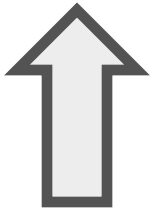
Standard Partitionable Slot w/ GPUs for owner

Creating Backfill Slots

- HTCondor has *long* supported the idea of multiple types of slots.
  - Newer is the idea of “backfill” slots that can overlap with existing resources.
  - **Goal**: If a slot is activated that needs the resources in a backfill slot, the EP will preempt the backfill slot.

```
use FEATURE : GPUs

use FEATURE : PartitionableSlot(1, 100%)
SLOT_TYPE_1_START = $(START) && \
            stringListsIntersect("Chemistry_PI", (TARGET.ChtcProjects ?: "")
```



Only allow the chemistry group access the “owned” slot

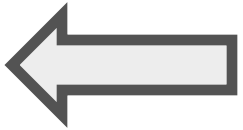
Creating Backfill Slots

- HTCondor has *long* supported the idea of multiple types of slots.
  - Newer is the idea of “backfill” slots that can overlap with existing resources.
  - **Goal**: If a slot is activated that needs the resources in a backfill slot, the EP will preempt the backfill slot.

```
use FEATURE : GPUs

use FEATURE : PartitionableSlot(1, 100%)
SLOT_TYPE_1_START = $(START) && \
            stringListsIntersect("Chemistry_PI", (TARGET.ChtcProjects ?: ""))

use FEATURE : PartitionableSlot(2, 100%)
SLOT_TYPE_2_BACKFILL = true
SLOT_TYPE_2_NAME_PREFIX = backfill
SLOT_TYPE_2_START = $(START) && (TARGET.RequestGPUs ?: 0) > 0
```



Backfill slot created as second p-slot

Creating Backfill Slots

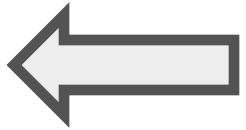
- HTCondor has *long* supported the idea of multiple types of slots.
  - Newer is the idea of “backfill” slots that can overlap with existing resources.
  - **Goal**: If a slot is activated that needs the resources in a backfill slot, the EP will preempt the backfill slot.

```
use FEATURE : GPUs

use FEATURE : PartitionableSlot(1, 100%)
SLOT_TYPE_1_START = $(START) && \
            stringListsIntersect("Chemistry_PI", (TARGET.ChtcProjects ?: "")

use FEATURE : PartitionableSlot(2, 100%)
SLOT_TYPE_2_BACKFILL = true
SLOT_TYPE_2_NAME_PREFIX = backfill
SLOT_TYPE_2_START = $(START) && (TARGET.RequestGPUs ?: 0) > 0

SLOT_TYPE_2_PREEMPT = size(ResourceConflict ?: "") > 0
```



Here’s the magic!

GPUs are heterogeneous

request_GPUs = 1

- By default, we pretend a “CPU is a CPU is a CPU”
- For the most part, this is correct:
  - By default, a compiled C++ program will work on a CPU made in 2025…
    - … or 2015
    - … or 2005.
  - Any CPU-based job can run almost* anywhere.
- This is *not* true for a GPU:
  - Each GPU family requires a completely different runtime environment (ROCm vs CUDA vs OpenCL).
  - Each executable has a minimum supported runtime version (CUDA capability).
  - As GPUs evolve, there can be a 10x performance difference between the oldest and newest viable GPUs.

GPUs are heterogeneous

- A given job may only run on a subset of available GPUs.
  - Worse: most users are ‘trained’ to only run on the ‘fastest’ GPUs and not think about throughput.
  - Note: Long queueing time affects throughput!
- **Approach**: Train users to express requirements, not model.

Good:

```
request_GPUs = 1
gpus_minimum_capability = 7.0
gpus_minimum_memory = 45GB
```

… Not so Good:

```
request_GPUs = 1
Requirements = \
  GPUs_DeviceName = \
    "NVIDIA A100-SXM4-80GB"
```

Note: this doesn’t work well if researcher is running one job! Why?

GPUs are not heterogeneous

- Approach 2: Build a flexible container image

Dockerfile

```
# Start from a standard CUDA image
FROM nvidia/cuda:12.4.1-runtime-ubuntu22.04

# Update software repositories and install dependencies
RUN apt-get update
RUN apt-get install software-properties-common -y
RUN add-apt-repository ppa:deadsnakes/ppa -y
ARG DEBIAN_FRONTEND="noninteractive"
ENV TZ=America/Chicago
RUN apt-get update

# Install python and pip
RUN apt-get install -y \
    python3.12 python3.12-dev python3.12-venv zip \
    && update-alternatives --install /usr/bin/python python
/usr/bin/python3.12 1 \
    && python -m ensurepip --upgrade \
    && rm -rf /var/lib/apt/lists/*

# Set working directory
WORKDIR /app

# Copy only the requirements file first and install dependencies
COPY requirements.txt .
RUN pip3.12 install -r requirements.txt
```

We want to use GPUs and this is a generally compatible CUDA image to start from

We want to install python3.12 and make sure that pip is available

Install our python dependencies

requirements.txt

```
pandas
typer
torch
matplotlib
torchvision
```

GPU container hints

- Some observations from the trenches:
  - Backward compatibility tends to be quite good: newer versions of CUDA provide a superset of functionality.
  - CUDA licensing is restrictive: you *cannot* distribute the resulting container publicly.
  - Worthwhile to keep driver fairly recent – never know what CUDA version will be inside the user's containers.
  - CUDA can even provide *forward* compatibility!
    - CUDA compiler can embed an intermediate representation (PTX) into the binary.
    - If the compiled code is run on newer GPUs incompatible with the machine code, then the runtime will JIT compile the PTX IR into machine code.
    - Provide the following flags to nvcc:

nvcc -arch=all

GPUs quantize poorly

Important metric: “time to first crash”!

- CHTC has ~35k CPU cores (hyperthreads).
  - There’s constant turnover as jobs finish. Someone submitting a new workload will see their first job start in minutes.
- CHTC has ~150 “open capacity” and ~150 “owned” GPUs.
  - A new workload may wait hours – a whole workday – before the first job starts.
- **Forget fairshare, we need to create more turnover!**
- Strategies:
  - Encourage shorter jobs.
  - Encourage smaller jobs (fewer GPU devices)
  - Use preemption to “take back” assigned slots.
  - Have dedicated short-running slots.

(i.e., how long does it take to find out your student has a typo on line 2?

Discussion

- **Question**: What mechanisms are interesting to this group for implementing policy?
- CHTC example: Negotiator concurrency limits (mechanism) to encourage short-running jobs (policy).
  - Up to 4 GPUs can be used for long-running jobs.
  - 2/3 of the open capacity can be used for medium-length jobs
  - All open capacity can be used for short-running.
- Plausibly could use SLOT_WEIGHT to:
  - Decrease cost for single-GPU jobs.
  - Reward using older models.
- What else? **How to encourage checkpointing**?
  - **How to encourage backfill usage**?
  - Start all checkpointable jobs in the “debug queue” and continue them elsewhere once they’re known to work?
    - Today, such a policy would be “death by ClassAd expressions”, no?

Example – concurrency limits for GPUs

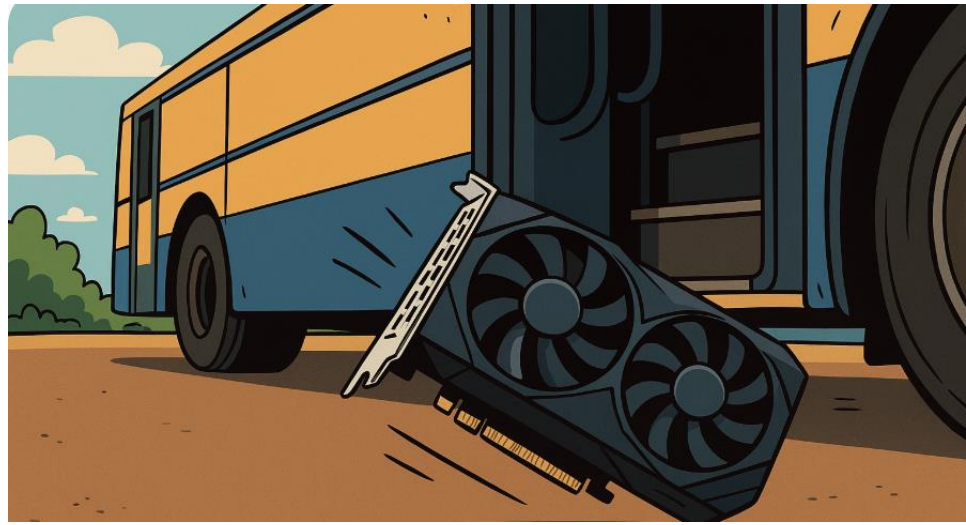
```
# Configuration on the central manager
GPULAB_NEGOTIATOR.CONCURRENCY_LIMIT_DEFAULT                =
GPULAB_NEGOTIATOR.CONCURRENCY_LIMIT_DEFAULT_GPULAB_MEDIUM      = 30
GPULAB_NEGOTIATOR.CONCURRENCY_LIMIT_DEFAULT_GPULAB_MEDIUMSHORT  = 60
GPULAB_NEGOTIATOR.CONCURRENCY_LIMIT_DEFAULT_GPULAB_MEDIUMLONG   = 12
GPULAB_NEGOTIATOR.CONCURRENCY_LIMIT_DEFAULT_GPULAB_INTERACTIVE  = 4
GPULAB_NEGOTIATOR.CONCURRENCY_LIMIT_DEFAULT_GPULAB_LONG        = 4

# Configuration on the AP / schedd – only showing medium-length jobs.
JOB_TRANSFORM_GPULABMEDIUM @= end
  REQUIREMENTS WantGPULab && (GPUJobLength =?= "medium")
  if defined MY.ConcurrencyLimits
     SET ConcurrencyLimits "$(MY.ConcurrencyLimits),GPULAB_MEDIUM.$(MY.Owner):$(MY.RequestGPUs)"
  else
     SET ConcurrencyLimits "GPULAB_MEDIUM.$(MY.Owner):$(MY.RequestGPUs)"
  endif
@end
JOB_TRANSFORM_NAMES = $(JOB_TRANSFORM_NAMES) GPULABMEDIUM
```

GPUs fall off the bus

At CHTC, we dread the following kernel message:

kernel: NVRM: GPU 0000:01:00.0: GPU has fallen off the bus.



- GPUs fail often:
  - Sometimes it's obvious: GPU disconnects from the PCIe bus (electrical issues? Heat?)
  - Sometimes it's not: NVIDIA driver crash causes all GPU invocations to hang indefinitely.
- Unlike CPUs, this isn't fatal for the host. The OS stays up, the EP continues to accept new jobs / activations.
  - Unfortunately, it means the EP will quickly fail all matching jobs.
  - Unlike for Singularity, the EP does *not* test the job environment works prior to launching the executable.

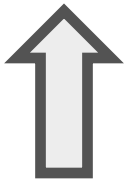
What to do?

Offline GPUs @ CHTC

- Every 5 minutes, a systemd timer fires that:
  - Iterates through all GPUs found in /proc/driver/nvidia/gpus/*/information
  - Runs nvidia-smi to see if the GPU is minimally functional.
  - If not, then it will drain the device (in the GPU driver), write the offline GPUs into a config file, and run condor_reconfig.
    - The startd will stop accepting claims for that particular GPU – no jobs will start on it!
- Example:

```
$ cat /var/run/condor/condor_config.startd_offline_gpus
OFFLINE_MACHINE_RESOURCE_GPUS = GPU-86f9627b GPU-35e4464a
```

```
$ cat /proc/driver/nvidia/gpus/0000\:b2\:00.0/information
Model:  NVIDIA GeForce RTX 2080 Ti
IRQ:    227
GPU UUID:  GPU-b1a77bb1-6cf8-9481-379c-d0739b9e731e
Video BIOS:  90.02.30.00.81
Bus Type:  PCIe
DMA Size:  47 bits
DMA Mask:  0x7fffffffffff
Bus Location:  0000:b2:00.0
Device Minor:  5
GPU Excluded: No
```



Note we use the “long-form”/UUID of the GPU device name. Can anyone guess why?

Conclusions

- In some cases, GPUs are a new beast:
  - GPUs (or their drivers) can go drastically wrong without crashing the node.
  - Compatibility matrix between jobs and GPU devices is more complicated.
- In other cases, GPUs make existing problems more acute:
  - Small numbers of devices make turnover more important. Back to preemption!
  - What's the "cost" of using the resource?
- CHTC itself is in this new(?) world, balancing CPU & GPU resources.
  - We've tried to solve some problems; left others unsolved.

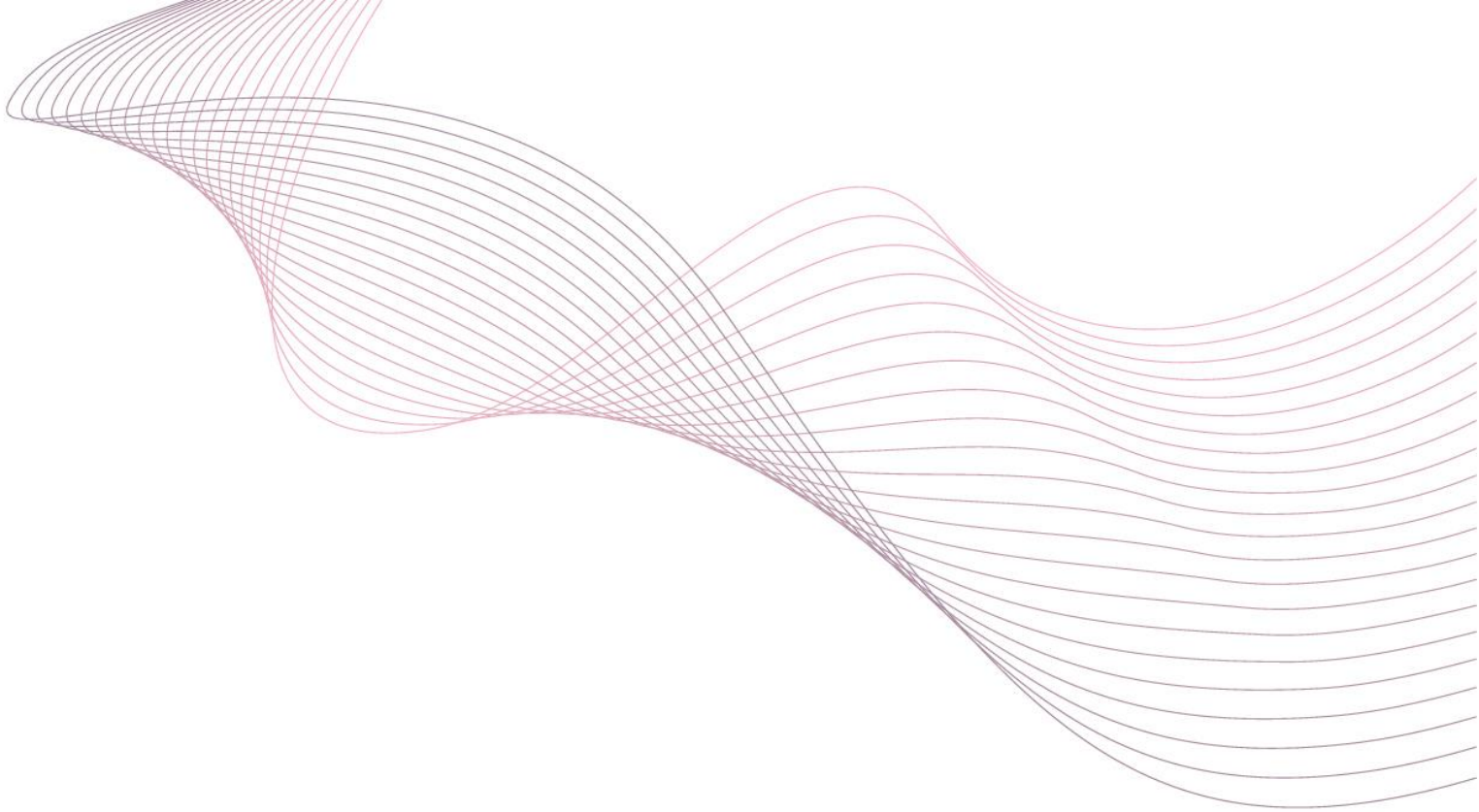
Interested in hearing what the community wants!

Questions?

This project is supported by the National Science Foundation under Cooperative Agreements OAC-2030508. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

MORGRIDGE
INSTITUTE FOR RESEARCH

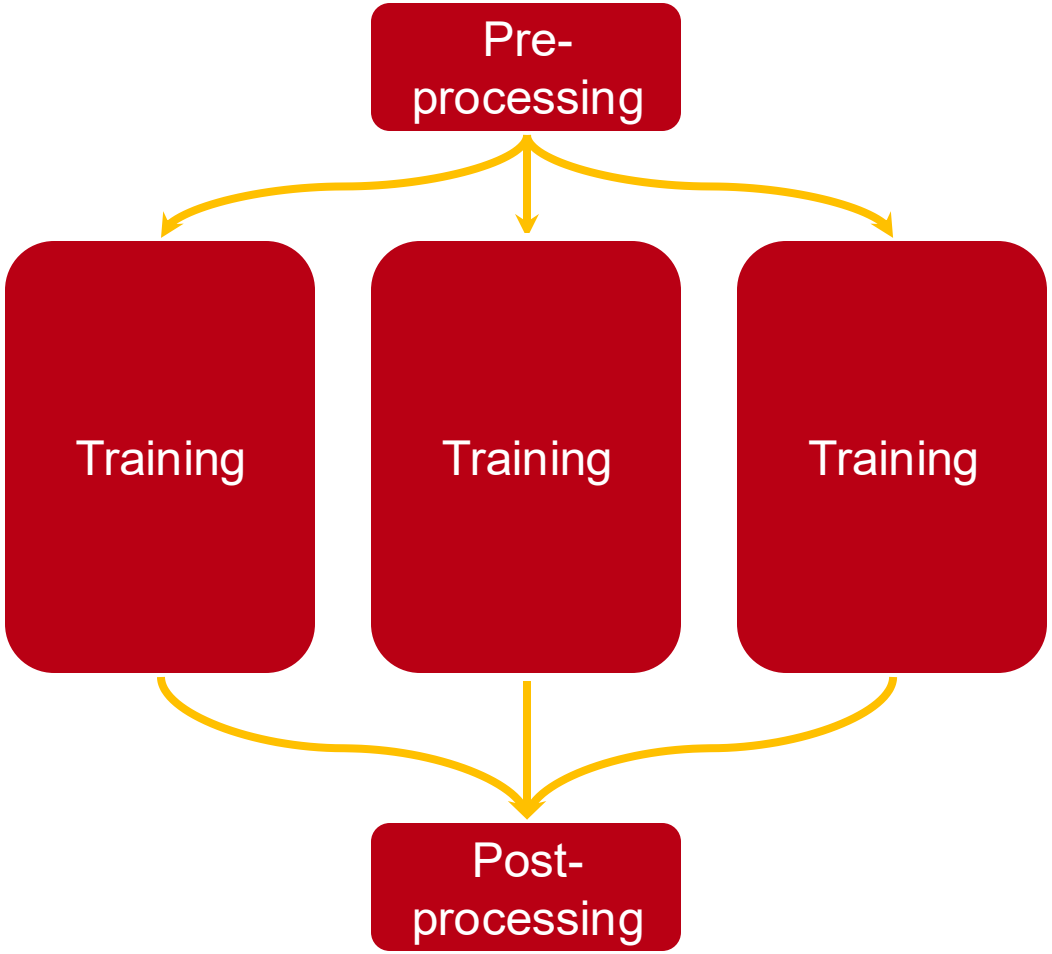
Backup



MORGRIDGE
INSTITUTE FOR RESEARCH

How can you improve your workflow?

- Automated workflows (e.g. using DAGMan)
- Hyperparameter and ensembles – Don't think of training one model, think of training many and finding (or combining) the best
- Weights and Biases (or similar tools) to monitor training runs
  - Works as expected, but be aware of API key leakage



Checkpointing

Update the logic of the training script to enable loading+resuming from a checkpoint.

The job doesn’t need to run to full completion within one job cycle. This means:

- Resilience against job eviction and machine issues
- Ability to request “short” jobs (more slots available!)

Checkpointing, required changes

train.py – add logic for loading and resuming

```
def main(
    data_dir: Path = typer.Option("./data", "--data-dir", "-d", help="Directory
containing the training data"),
    checkpoint_dir: Path = typer.Option("./checkpoints", "--checkpoint-dir", "-c",
help="Directory to save model checkpoints"),
    epochs: int = typer.Option(10, "--epochs", "-e", help="Number of training epochs"),
    checkpoint_every: int = typer.Option(5, "--checkpoint-every", "-c", help="Number of
epochs between checkpoints"),
):
   …
    if os.path.exists(model_path):
        print(f"Loading checkpoint from {model_path}")
        checkpoint = torch.load(model_path)
        model.load_state_dict(checkpoint['model_state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
        start_epoch = checkpoint['epoch'] + 1
        losses = checkpoint['losses']
        accuracies = checkpoint['accuracies']
        print(f"Resuming training from epoch {start_epoch}")

    for epoch in range(start_epoch, epochs):
        … # same internals as the baseline example
        if (epoch+1) % checkpoint_every == 0 or (epoch+1) == epochs:
           # Save checkpoint after each epoch
           checkpoint = {
               "epoch": epoch,
               "model_state_dict": model.state_dict(),
               "optimizer_state_dict": optimizer.state_dict(),
               "losses": losses
           }
           torch.save(checkpoint, model_path)
           print(f"Saving checkpoint to {model_path}. Epochs trained   xx`: {epoch + 1}
of {epochs}")
           if epoch+1 == epochs:
               sys.exit(0)
           else:
               sys.exit(85)
```

Add a command line option defining how often to save (nice addition)

Store some additional required information for resuming from a checkpoint

When to save checkpoints

The HTCondor bits:

With submit file changes, HTCondor will detect this exit code and handle checkpoint files gracefully

https://portal.osg-htc.org/documentation/htc_workloads/submitting_workloads/checkpointing-on-OSPool/

Checkpointing, submit file changes

train.sub

```
container_image = osdf:///ospool/ap40/data/iaross/catdog.sif
# see our guide on converting

checkpoint_exit_code         = 85
transfer_checkpoint_files    = output/model.pth

request_disk = 500MB
request_memory = 6GB
request_cpus = 1

# See our CHTC guide on GPU usage / OSPool guide
request_gpus = 1
gpus_minimum_capability = 7.5
gpus_minimum_memory = 4096

... # rest of submit file
```

train.sh

```
#!/bin/bash

wd=$(pwd)
mkdir data
mkdir $wd/output

unzip train.zip -d data/
rm train.zip

cd /app/

python train.py \
  --data-dir $wd/data/ \
  --checkpoint-dir $wd/output/
  --checkpoint-every 5
  --epochs 20
exit $?
```