The INTERNATIONAL CONFERENCE for HIGH PERFORMANCE COMPUTING NOV 15–20
CHICAGO, IL NETWORKING, STORAGE, & ANALYSIS

SPONSORED BY IEEE COMPUTER SOCIETY TCHPC acm Association for Computing Machinery sighpc

SC26
Chicago, IL | hpc unites.

Overview of the SCC26 Benchmarks

September 10th, 2026

Ryan DeRue, Purdue University

Jacob Verburgt, Purdue University

Miro Hodak, MLCommons/AMD

Shriya Rishab, MLCommons/Nvidia

Benchmarking Logistics

- Benchmarking portion will take place from Monday morning to Monday evening
  - More detailed logistics are available in the SCC overview webinar that was given earlier
- You will perform a series of three benchmarks
  - The artifacts of those benchmarks will be submitted to us and your cluster will be “certified”
- Certifying your cluster requires
  - Staying within power budget (10kW) while running all 3 benchmarks
  - All benchmarks must be run with the same hardware configuration
  - After certification
    - You cannot change your hardware configuration or reboot any node
    - You can re-submit a single benchmark within the benchmarking window
      - Normal penalty for going over power budget will apply
- Detailed submission instructions will be shared later
  - Follow submission instructions carefully
  - We plan to use scripts for automated grading
  - **Name your files and folders as instructed**
- Suggestions
  - Familiarize yourself with the benchmarks ahead of time
    - Write scripts and automate where possible
  - Vendor provided binaries are acceptable if these are publicly available

Benchmark Tasks

- There will be three benchmarking tasks
- Two Synthetic Benchmarks
  - High Performance Linpack (HPL)
  - HPL-Mixed Precision (HPL-MxP)
- One Application Benchmark
  - MLPerf Training
- We expect that the majority of your time will be spent on the application benchmark

Synthetic Benchmarks

Targets Specific Operations

HPL and HPL-MxP will be measuring the raw performance of your system against particular data types.

Operationally, this class of benchmarks are often used to assess the health of a node

Useful for making comparisons between hardware performance on a broader variety of applications

Application Benchmarks

Targets Holistic Performance

MLPerf will be measuring the performance of your system against a targeted workload.

This class of benchmarks is used for informing how a system will perform in a particular domain.

Useful for making very targeted comparisons with respect to the desired task

High Performance Linpack (HPL)

- One of the most popular benchmarks in the HPC world
  - TOP-500 listing of world's fastest supercomputers use HPL
    - Most recent list published June 2026
      - LineShine, Shenzhen Cloud Computing Co., 2.198 EFlops/s
- Solves a (random) dense linear system in double precision (FP64)
  - https://www.netlib.org/benchmark/hpl/index.html
- Used to measure "performance" of a computer or a cluster
- Input
  - Parameters for the benchmark run
- Output
  - No. of FLOP/s
- SCC25 best HPL score: National Taiwan University (561.6 Teraflops)
  - This was performed within the same 10kW power budget

Building and Running HPL

- Download and untar the HPL source tarball
  - `tar -xf hpl-2.3.tar.gz`
- Need to load the necessary libraries for BLAS routines
- Verify that you have all the dependencies (compiler, MPI, BLAS)
  - Internally uses BLAS libraries for LA subroutines
    - Intel MKL
    - OpenBLAS
  - Uses MPI for distributed memory parallelism
  - Uses OpenMP for shared memory parallelism
- Now compile HPL
  - `./configure --prefix= …`
  - `make -j16 && make install`
- Add the binary location to your PATH
  - `export PATH=/path/to/xhpl/binary:$PATH`
- Now run HPL
  - `mpirun -np 2 xhpl`
- NVIDIA and AMD both provide containers for launching HPL on their GPUs
  - NVIDIA Documentation: https://docs.nvidia.com/nvidia-hpc-benchmarks/HPL_benchmark.html
  - AMD Documentation: https://github.com/amd/InfinityHub-CI/tree/main/rochpl

HPL Input Files and Tuning

- Input for the benchmark is supplied through an input file named `HPL.dat`
  - Edit the input file to reflect your setup
- Important parameters
  - `N  #Array size`
  - `NB  #Block size for LA operations`
  - `P  #Factorization rows`
  - `Q  #Factorization columns. PxQ must equal your MPI processes`
  - Change `no. of algorithms to test`
- Typically, HPL on CPUs should occupy 80-90% of your memory for optimal performance
  - On GPUs, this is often closer to 90-100%
- `NB` (Block size) impacts performance
  - Try different `NB` sizes
  - Empirically find out which one is better
  - Optimal NB size depends on your architecture/GPU type
- Exact layout of MPI processes/OpenMP threads impact performance
  - Optimal layout depends on the processor architecture
- HPL tends to perform best when the processor grid approximates a square shape
  - P=4, Q=4

HPL Resources

- https://www.netlib.org/benchmark/hpl/faqs.html
- https://frobnitzem.github.io/hpl-hpcg/
- https://www.advancedclustering.com/act_kb/tune-hpl-dat-file/
- https://ulhpc-tutorials.readthedocs.io/en/latest/parallel/mpi/HPL/
- https://developer.amd.com/spack/hpl-benchmark/

HPL-MxP

- HPL with FP16 factorization
  - Solver has a “refinement” step to bring the solution back to 64-bit accuracy
  - Developed as a result of the shift towards reduced precision arithmetic
- A Top500 of HPL-MxP is also published
  - El Capitan, LLNL, 16.680 EFlops/s

November 2025

Rank	Site	Computer	Cores	HPL-MxP (Eflop/s)	TOP500 Rank	HPL Rmax (Eflop/s)	Speedup
1	DOE/SC/LLNL	El Capitan	11,340,000	16.680	1	1.8090	9.2
2	DOE/SC/ANL	Aurora	9,264,128	11.643	3	1.0120	11.5
3	DOE/SC/ORNL	Frontier	9,066,176	11.390	2	1.3530	8.4
4	EuroHPC/FZJ	JUPITER Booster	4,801,344	6.255	4	1.0000	6.3
5	SoftBank	CHIE-4	662,256	3.304	17	0.1354	24.4

HPL-MxP Input

- HPL-MxP input looks very similar to HPL but is typically passed through the command line
  - `N -> --n  <int>`
  - `NB -> --nb <int>`
  - `P -> --nprow <int>`
  - `Q -> --npcol <int>`
- A few new required arguments:
  - `--sloppy-type <setting> //used to specify the data type used in LU Factorization`
  - `--gpu-affinity <string> //colon delimited list of GPUs`
  - `--nporder <string> //"row" or "column"`
- There are also a few more optional arguments depending on the implementation used
  - Pay particular attention to options regarding GPU-aware MPI
- **We require that lower precision factorization happen in FP16**
  - This is a changed requirement since last year
  - This is also a requirement of an official submission to the list

HPL-MxP Implementations

- The authors of HPL-MxP created a reference implementation
  - You would need to build this implementation from source
- Most teams will want to use a vendor optimized implementation
  - NVIDIA's HPL-MxP Container
  - AMD's HPL-MxP Container
  - Intel's HPL-AI* Container
- Each of these implementations have slightly different optional arguments
  - Read the documentation for the one you are using
- You are allowed to use any *publicly available* vendor implementations

HPL-MxP Resources

- https://hpl-mxp.org/
- https://bitbucket.org/icl/hpl-ai/src/main/
- https://catalog.ngc.nvidia.com/orgs/nvidia/containers/hpc-benchmarks
- https://www.amd.com/en/developer/resources/infinity-hub/hpl-mxp.html
- https://docs.nvidia.com/nvidia-hpc-benchmarks/HPL_MxP_benchmark.html
- https://www.intel.com/content/www/us/en/docs/onemkl/developer-guide-linux/2024-1/overview-intel-distribution-for-hpl-ai-benchmark.html

MLPerf Benchmark for SCC'26

Llama 2 70b Training (LoRA Fine-tuning)

Miro Hodak, MLCommons/AMD
Shriya Rishab, MLCommons/Nvidia

MLPerf Training Benchmark Metrics

Primary Metric

Time-to-Train (TTT)

The leaderboard metric representing the overall time required to train the model to a fixed quality target.

TTT = run_stop − run_start

Reported in minutes. **Lower is better.**

Underlying Drivers

Throughput & Convergence

- **Throughput:** Determines how quickly the system processes training samples.
  - **Higher is always better**
- **Convergence:** Determines how many samples the model must process to meet quality.
  - **Must comply with RCP**

** Throughput is supporting info only and not used as the main leaderboard metric.*

A run succeeds only after crossing the benchmark-specific quality target: you need both fast throughput and convergence.

Scale Out

Core Concept & Goal

Distributed Paradigm
Training is naturally a scale-out task. Distributing workloads across cluster nodes is native to modern deep learning.

Target Objective
Use more compute nodes to achieve the absolute minimal Time-to-Train (TTT).

Scale Out Trade-offs

The Convergence Bottleneck
More nodes improve throughput, but more samples are required to converge. This is not a simple weak scaling problem.

RCP & Batch Size Constraints
Convergence is governed by RCPs and is highly model-dependent. Larger global batch sizes inherently require more samples to converge.

Optimization Key
To successfully minimize TTT, you must search for and find the optimal hyperparameters (HPs) and parallelism settings.

Llama2-70b Training Benchmark

LLaMA 2 70B

Llama 2 70B (70 billion parameters) was released by Meta in July 2023 and has been highly popular since its release.

huggingface.co/blog/llama2

SCROLLS Dataset

Standardized CompaRison Over Long Language Sequences.

Composed of question-summary pairs from official government research reports, including the Congressional Research Service and U.S. GAO.

LoRA Fine-Tuning

Adapts pre-trained models to specific tasks on smaller, task-specific datasets to enhance performance.

Low-Rank Adaptation is a PEFT technique that trains dense layers indirectly by optimizing rank decomposition matrices, keeping pre-trained weights frozen.

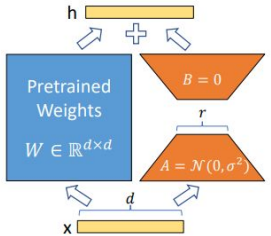


Figure 1: Our reparametriza-
tion. We only train A and B

Benchmark Status

MLPerf Training

One of the most popular benchmarks

Official Resources:

mlcommons.org/lora-fine-tuning

Convergence (RCPs)

A run converges at the first scheduled evaluation where the benchmark reaches its quality target.

Llama 2 70B LoRA Specs

Quality Metric & Target

Validation loss ≤ **0.925** (logged under MLLOG key `eval_accuracy`)

Convergence Value

Number of training samples consumed at successful evaluation. Fewer samples mean faster convergence.

Evaluation Frequency

Every 384 sequences:

- `CEIL(384/GBS) * GBS` steps if 384 not divisible by GBS
- Skip first `FLOOR(0.125*GBS + 2)` evals

Quantization & Example

Why is Convergence Quantized?

Because quality is checked only at specific evaluation points, actual convergence values are quantized to those intervals.

Example with Global Batch Size (GBS) = 8

Evaluation frequency is 384 samples and the first 3 evaluations are skipped. Legal evaluation points are:
1536, 1920, 2304, 2688, 3072, 3456, ...

The "Next Evaluation" Rule

A run that crosses the target shortly after 3,072 samples will still be recorded as converging at **3,456** because that is the next scheduled evaluation.

Convergence (RCPs)

What is an RCP?

A **Reference Convergence Point** describes the expected stochastic convergence behavior of a benchmark.

Your runs need to match convergence (in number of samples) given by RCP

Llama 2 70B LoRA Specs:

- **20 seed runs** to characterize typical convergence behavior for GBS 8, 16, 32 and 128.
- **Min Threshold:** Statistically derived limit for plausible convergence.

Data: rcps_llama2_70b_lora.json

Ten Runs Required

Training is highly stochastic due to random seeds, data order, dropout, and hardware nondeterminism.

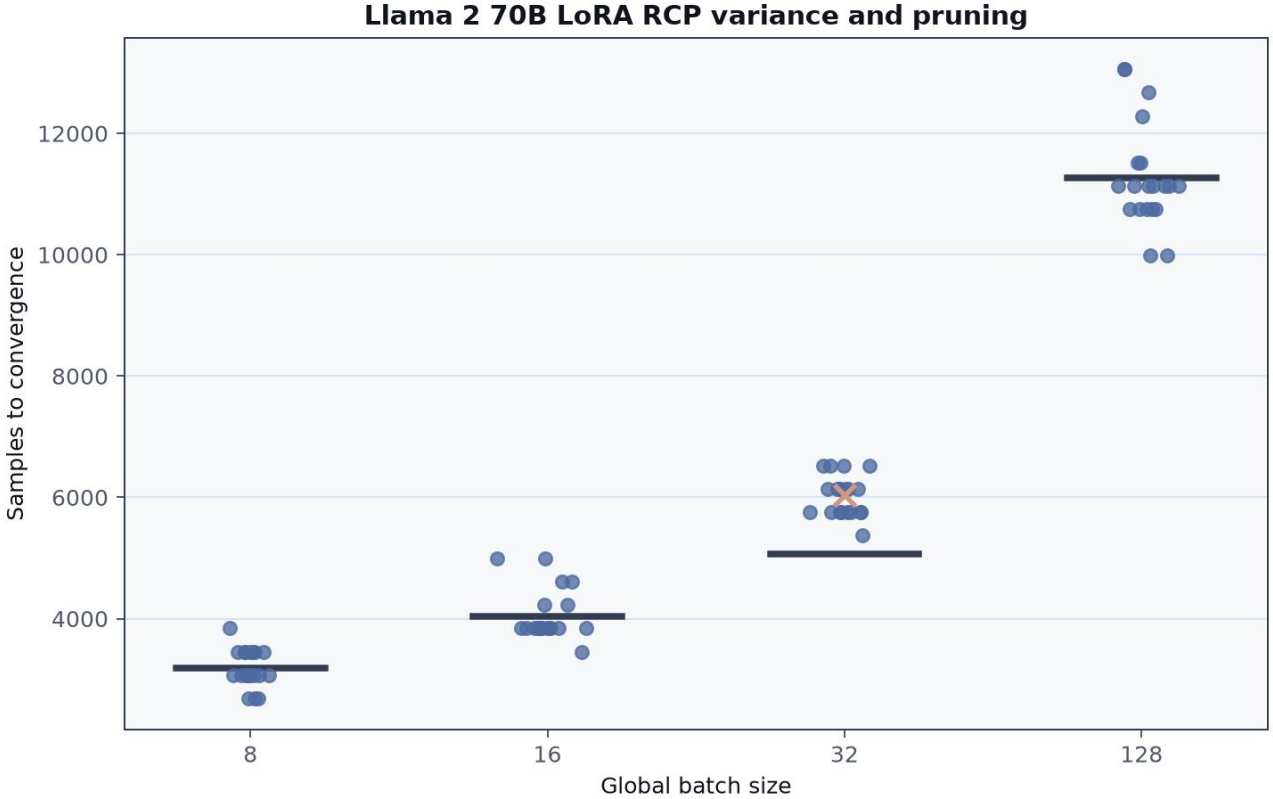
Olympic Trimming:

- **Score:** Calculated as the mean of the middle 8 convergence runs.
- **Outliers:** Fastest & slowest runs are discarded to prevent lucky/unlucky outliers.
- **Failed Runs:** Up to 1 failed run is allowed as it will be discarded.
- **Timing:** Results use the same trimmed 8-run standard.

Normalization Rules

- **Implausibly Fast** runs fail the benchmark entirely.
- **Credibly Fast** runs are accepted but scaled up.
- **One-Sided Rule:** Slow runs never receive downward adjustment.

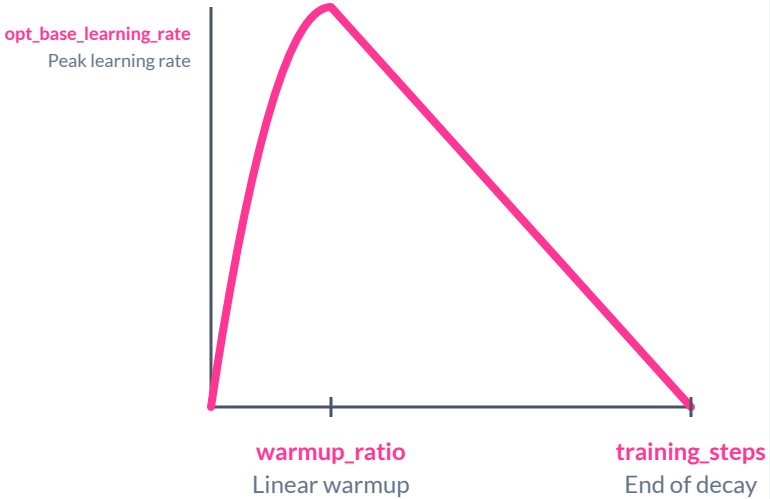
Convergence (RCPs)



Dots: 20 RCP runs · dark line: operational mean · ×: measured mean removed by RCP pruning

MLPerf Training llama 2 70B LoRA parameters

Cosine Learning Rate Schedule



Unconstrained HPs & Tuning

1. opt_base_learning_rate

The maximum peak learning rate. If you increase the global batch size, scale this value up (linearly or via square-root) to maintain optimal convergence.

2. opt_learning_rate_warmup_ratio

The fraction of total training steps dedicated to linear warmup. Increase this if early training is unstable or gradients explode.

3. opt_learning_rate_training_steps

Total decay steps. Can exceed the convergence samples budget so the training region uses a high learning rate for most of your training.

4. Batch Size & Accumulation Steps

Both global_batch_size and gradient_accumulation_steps are unconstrained. Scale accumulation steps to fit target batch sizes within GPU memory limits.

Hyperparameters Reference Link - Shows some constrained HPs as well which cannot be modified.

MLPerf Training Benchmark Rules

Core Requirements

1. MLLOGGER Integration
Integrate MLLOGGER into the implementation.

2. Accuracy Evaluation
Runs must do accuracy evaluation every **384 samples**.

Reference and optimized implementations already have these in place.

3. Running the benchmark
Follow README.md in the implementations on how to run training. Generally it is a bash script that runs with docker either locally or on a slurm-based cluster.

Run 1 run first and investigate it before generating a set of 10 submittable logs.

What You Can & Cannot Change

Hyperparameters (HPs)
Certain HPs can change (see previous slide).

Parallelism
You can use any types and combinations of parallelism
- Think Data Parallelism (DP), Fully sharded data parallelism (FSDP), Tensor Parallelism (TP), Pipeline Parallelism (PP), Context Parallelism (CP)...

Precision
Any type of precision is okay.
- Reference is in BFloat16 precision.
- Optimized implementations from Nvidia and AMD use FP4 in later rounds, FP8 was used in earlier rounds.

MLPerf Training Implementations

System	Approx TTT (mins)	Implementation
8x A100s	120 to 140	code
8x H100s	30	NVIDIA v4.0 code
8-16x H200s	10 to 40	Dell v6.0 code
8-16x MI300s	15 to 30	AMD v5.1 code
8-16x MI350s	8 to 15	AMD v6.0 code

Sources

Official Benchmarks

mlcommons.org/benchmarks/training/

GitHub Repository

https://github.com/mlcommons/training_results_v<ROUND>/tree/main/<ORGANIZATION>/benchmarks/llama2_70b_lora/implementations/

Supports rounds v4.0, v4.1, v5.0, v5.1, and v6.0 across verified organizations.

MLLOGGER

What is MLLogger?

The logging API from the **mlperf_logging** package. It emits machine-readable JSON events prefixed with:

`:::MLLOG`

Three Primary Methods

`event(...)`

A point-in-time value (e.g., GBS, seed, or evaluation loss).

`start(...)`

Starts a timed interval (e.g., `run_start`, `eval_start`).

`end(...)`

Ends an interval (e.g., `run_stop`, `eval_stop`).

What the Checker Determines

- Whether the run followed the required lifecycle.
- Which benchmark, system, GBS, seed, and hyperparameters were used.
- Whether the quality target was reached (eval loss <= 0.925).
- Samples to convergence & Time to train (TTT).
- Whether 10 runs have distinct seeds & credible RCP convergence.

Typical Log Event Format

```
:::MLLOG {"key": "init_start", "value": true, "time_ms": 17182901}
:::MLLOG {"key": "global_batch_size", "value": 16, "time_ms": 17182905}
:::MLLOG {"key": "eval_accuracy", "value": 0.912, "metadata": {"samples_count": 384}}
```

Ensure rank 0 alone writes these records. Use a distributed barrier around timing boundaries (e.g., `init_stop` , `run_start` , `run_stop`). If using the reference implementation or optimized AMD/NVIDIA implementations, this should already be there.

Submission Package & Folder Structure

```
TEAM_NAME/
├── systems/
│   └── student_system.json
├── benchmarks/
│   └── llama2_70b_lora/
│       ├── README.md
│       ├── run_and_time.sh
│       ├── config.sh
│       └── source-or-patches/
└── results/
    └── student_system/
        └── llama2_70b_lora/
            ├── result_0.txt
            ├── result_1.txt
            ├── result_2.txt
            ├── result_3.txt
            ├── result_4.txt
            ├── result_5.txt
            ├── result_6.txt
            ├── result_7.txt
            ├── result_8.txt
            ├── result_9.txt
            └── scaling.json
```

Submission Guidelines

- **System Match:** The results directory name student_system must exactly match the filename systems/student_system.json.
- **JSON Template:** View an official system JSON file example.
- **Benchmark Name:** The benchmark directory must be exactly named llama2_70b_lora.
- **10 Genuine Runs:** Exactly 10 result files (result_0.txt to result_9.txt) are required. These must represent separate, real runs with distinct seeds (no duplicated logs).
- **Reproducibility:** The benchmark folder must contain sufficient source code, patches, and configurations to fully reproduce your results.
- **Scaling Metrics:** The scaling.json file is managed and updated automatically by the package checker.

Checks to Run Before Submission

01. Run RCP Checker Directly

Check convergence credibility on the directory containing all 10 results.

```
python -m mlperf_logging.rcp_checker \
TEAM_NAME/results/student_system/llama2_70b_lora \
--rcp_usage training \
--rcp_version 6.0.0 \
--verbose
```

02. Run Package Checker

Validates counts, compliance, seeds, and system JSON structure.

```
python -m mlperf_logging.package_checker \
TEAM_NAME training 6.0.0 \
--werror \
--log_output /tmp/package_student_team.log
```

03. Complete Official Verifier

Recommended final script. Runs package, summarizer, and repo checkers.

```
# Run the comprehensive test suite in order:
bash sources/logging/scripts/verify_for_v6.0_training.sh \
TEAM_NAME
```

Scripts installation: https://github.com/mlcommons/logging/tree/master#installation
pip install "git+https://github.com/mlperf/logging.git@6.0.0-rc6"

Reading a Result Log


```
:::MLLOG {
"namespace": "",
"time_ms": 1778522669880,
"event_type": "POINT_IN_TIME",
"key": "eval_accuracy",
"value": 0.92318,
"metadata": { "samples_count": 3072 }
}
```

Log Field Interpretation

time_ms: Absolute timestamp in milliseconds

event_type: Point event, interval start/end

key / value: Event name & measured metric value

samples_count: Training samples consumed at event

What to Inspect Manually

1. Configurations: Confirm exactly one global batch size and seed per run.

2. Run Status: Verify presence of `run_start` / `run_stop`, and that final status is "success".

3. Compute Raw TTT: Calculate total duration via:

```
(run_stop.time_ms - run_start.time_ms) / 60000
```

3. Understanding TTT and throughput:

```
TTT ≈ samples_to_convergence / effective_training_throughput
        + evaluation_time
        + synchronization and other timed overhead
```

4. Accuracy Progression: Read eval records in order to ensure convergence aligns with target threshold (**0.925**).

5. Convergence: Read samples_count (**3072**) to ensure convergence matches expectation

Model and Dataset Downloads

- Model and Dataset to be available as a download from MLCommons storage
  - A special download page for SCC26 is being created, to be ready soon
  - We will announce the location in the next few days
- Download the model and dataset and bring it to the competition
  - There will be a local copy available, but having your own will make things easier

Scoring Principles

Scoring will account for:

Absolute Performance

Who can achieve the lowest TTT.

Relative Performance

Performance relative to your HW capabilities.

Innovations

Novel optimizations and creative technical solutions.

Tips to be Successful

Do Test Runs

Before the competition starts, run extensive tests to calibrate your setup.

- Experiment with GBS/LR, parallelism, and precision to optimize TTT
- Understand convergence requirements and RCPs

Get Assets Early

Do not wait until the last minute to secure your working environment.

- Download models and datasets well ahead of time
- Bring all necessary components ready to SCC'26

Ask Questions

Proactive preparation is key to preventing deployment blockers.

- Engage with support and ask questions as you prepare
- Resolve technical doubts before competition begins

Summary: Practical/Short Guide

- Have model/dataset ready before competition
- Start with optimized (or reference) implementations
  - Have MLLOGGER embedded, knobs for parallelism and HPs
- Execute exploratory runs: Scale to your entire cluster and minimize TTT
  - High throughput and appropriate number of samples
    - Learning Rate HPs control convergence - make sure that you are hitting RCP convergence
    - Maximize throughput via system settings
- Execute 10 runs once you have identified optimal settings
- Verify convergence by running RCP checker
- Create submission package:
  - 10 runs in appropriately named files
  - student_system.json describes your system - can be prepared ahead of time
  - Verify package with package checker and official verifier scripts
- If running into issues with packaging, focus on log files and submit those

Questions?