LLMs on Aurora: Hands-On

🍋 ezpz

Sam Foreman
foremans@anl.gov
ALCF

2025-05-07

samforeman.me/talks/incite-hackathon-2025/ezpz/slides

ALCF INCITE GPU HACKATHON
May 20–22, 2025

LLMs on Aurora: 🍋 ezpz

Sam Foreman

2025-05-07

samforeman.me/talks/incite-hackathon-2025/ezpz/slides

📍 Currently

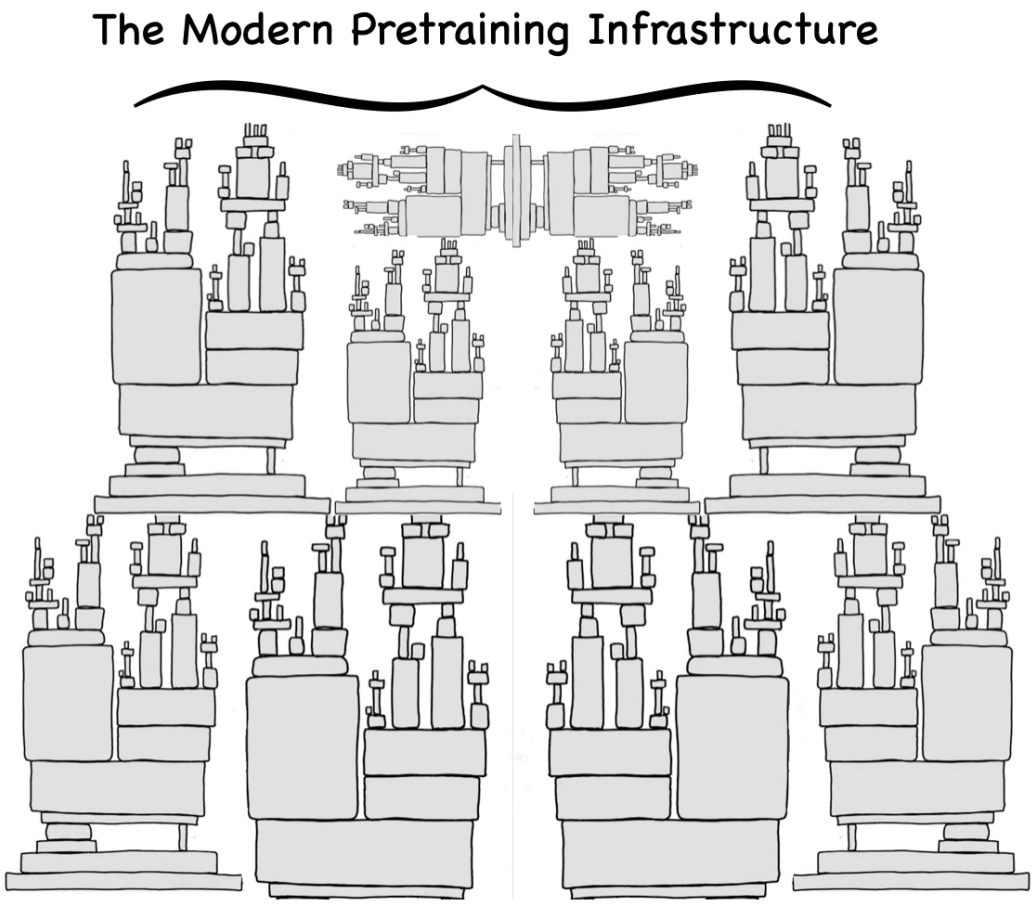

Figure 1: Current state of LLM Pretraining. [Source]

💬 LLMs on Aurora

- 🍋 ezpz
- 🤗 transformers
- 🏎️ Megatron-DeepSpeed

🍋 ezpz

> Write once, run anywhere

samforeman.me/talks/incite-hackathon-2025/ezpz/slides

🐣 Getting Started

1. Submit interactive job:

```
l select=2 -l walltime=01:00:00 \
lesystems=home:flare \
u_hack \
u_hack_prio
```

2. Source[1] the ezpz/bin/utils.sh script (using `curl` to download it[2]):

```
curl -L https://bit.ly/ezpz-utils)
```

1. In *general*, you should be wary of running random scripts from the internet.
2. https://bit.ly/ezpz-utils, since https://raw.githubusercontent.com/saforem2/ezpz/main/bin/utils.sh is a bit of a pain

🏝️ Shell Environment

1. Setup environment:

```
p_env
```

🔍 Environment Setup with `ezpz_setup_env`

- Wrapper around `ezpz_setup_job && ezpz_setup_python`

1. `ezpz_setup_job`: Determine the specifics of our active (PBS, SLURM) job[1]
2. `ezpz_setup_python`:
   - **if @ ALCF**:
     - Load the appropriate modules and activate base `conda` env
   - **else**:
     - Look for an active `conda` environment
       - If found, use it to build a new virtual environment
   - Activate the newly created `venvs/$(basename ${CONDA_PREFIX})` environment

1. e.g. `${NHOSTS}`, `${NGPU_PER_HOST}`, `${NGPUS}`, ...

⏱️ Working with Job Scheduler(s)

- `ezpz` integrates directly with the ALCF job scheduler[1]
  - has mechanisms for getting information about our currently running jobs
- 🪄 *Automagically*:
  - Determine the specifics of our active (PBS, SLURM) job
    (e.g. `${NHOSTS}`, `${NGPU_PER_HOST}`, `${NGPUS}`, ...)
  - Load the appropriate modules[2]
  - Create (or activate) a virtual environment *on top* of a base conda environment

1. *Should also work with SLURM* (needs further testing)
2. On any of the ALCF systems, including: Aurora, Polaris, ..., etc.

🔄 Use Custom Node Lists

- Experiment[1] with custom `hostfile`(s), e.g.:

```
curl -L https://bit.ly/ezpz-utils)
o `hostfile` specified, find and use `$PBS_NODEFILE`
p_job
 a subset of nodes:
 $PBS_NODEFILE > nodefile-0-1
 custom `nodefile-0-1`:
p_job nodefile-0-1  # will use `nodefile-0-1`
```

1. Or, for example, if you would like to exclude a node you suspect is having issues

🐍 Python Environments

- **ALWAYS** work inside a virtual environment
  - best practice is to maintain separate virtual environments for:
    - each project you work on
    - different versions of a specific package you're working with
      e.g you would want different envs for `torch==2.X` vs `torch==2.Y`
  - *Mangled python environments are one of the most common issues faced by users*

📦 Install `ezpz`

1. Install[1]:

```
m pip install "git+https://github.com/saforem2/ezpz"
```

2. Run distributed test:

```

```

3. Launch *any* python *from* python

   - Launch a module:

     ```
     ch -m ezpz.test_dist
     ```

   - Launch a python string:

     ```
     ch -c "'import ezpz; ezpz.setup_torch()'"
     ```

1. You should *always* be working in a virtual environment. See: 🏖️ Shell Environment

✚ How to Modify Existing Code

```
ezpz
z.setup_torch()
o('cuda')
o(ezpz.get_torch_device_type())
```

✨ Features

- Initializing PyTorch across multiple processes

```
pz
setup_torch()
pz.get_rank()
e = ezpz.get_world_size()
k = ezpz.get_local_rank()
```

- Automatic device detection (`xpu`, `cuda`, `mps`, `cpu`, ...)

```
.rand((10, 10)).to(ezpz.get_torch_device_type())
```

- Automatic (single-process) logging

```
ezpz.get_logger(__name__)
```

- Distributed debugger:

```


_code()
ception:
breakpoint(0)
```

🧪 Experiment Tracking

```
zpz
zpz.setup_torch()
 ezpz.get_logger(__name__)
== 0:                    # — [1.] —

_ = ezpz.setup_wandb(
    "ezpz.examples.minimal"
)
pt Exception:
logger.exception(
    "Failed to initialize wandb, continuing without it"
)
ld {model, optimizer}, etc...
 range(train_iters):
ics = train_step(...)
er.info(                 # — [2.] —
history.update(metrics)  # — [3.] —

== 0:
ory.finalize()
```

1. Initialize W&B (if `WANDB_DISABLED` is not set)
2. Log summary of metrics to stdout
3. Update `history.history` with metrics[1]

1. Will automatically be reported to W&B if a run is detected

👈 Minimal Example

- See [ezpz/examples/minimal.py]

```
s
ime
zpz
orch
 ezpz.get_logger(__name__)
twork(torch.nn.Module):
__init__(
self,
input_dim: int,
output_dim: int,
sizes: list[int] | None,

super(Network, self).__init__()
nh = output_dim if sizes is None else sizes[0]
layers = [torch.nn.Linear(input_dim, nh), torch.nn.ReLU()]
if sizes is not None and len(sizes) > 1:
    for idx, size in enumerate(sizes[1:]):
        layers.extend(
            [torch.nn.Linear(sizes[idx], size), torch.nn.ReLU()]
        )
    layers.append(torch.nn.Linear(sizes[-1], output_dim))
self.layers = torch.nn.Sequential(*layers)
forward(self, x: torch.Tensor) -> torch.Tensor:
return self.layers(x)
```

🏃 Running the Minimal Example

To run the previous example we:

1. Source the `ezpz` utils script:

```
curl -L https://bit.ly/ezpz-utils)
```

2. Setup our environment:

```
p_env
```

3. Run the example:

```
ch -m ezpz.examples.minimal
```

📝 `ezpz-test`

- `ezpz-test` is a simple test script that trains a small model using DDP across all available GPUs
  - It will automatically detect the number of GPUs and launch an appropriate `mpiexec` command to run the training script across all GPUs
- See: [ezpz/test.py]
- Command:

```
ra_nre_models_frameworks-2025.0.0](👻 aurora_nre_models_frameworks-2025.0.0
5 @ 07:41:35][x4520c1s0b0n0][/f/d/f/p/s/ezpz][🌱 update-utils][📦👷✓] [⏱️ 5
st
```

🦜 Generate Text

- See: ezpz/generate.py
- Command:

```
m ezpz.generate --model_name meta-llama/Llama-3.1-8B
```

🤗 Huggingface Trainer

- See ezpz/hf_trainer.py
- Command:

```
nch -m ezpz.hf_trainer \
taset_name=eliplutchok/fineweb-small-sample \
reaming \
del_name_or_path=meta-llama/Llama-3.2-1B \
16=true \
_train=true \
_eval=true \
port-to=wandb \
gging-steps=1 \
clude-tokens-per-second=true \
ock-size=128 \
x-steps=10 \
clude-num-input-tokens-seen=true \
to_find_batch_size=true \
adient_checkpointing=true \
tim=adamw_torch \
erwrite-output-dir=true \
gging-first-step \
clude-for-metrics='inputs,loss' \
x-eval-samples=50 \
p-backend=ccl
```

🏎️ Megatron-DeepSpeed

```
 https://github.com/argonne-lcf/Megatron-DeepSpeed
on-DeepSpeed
curl -L https://bit.ly/ezpz-utils)
m pip install -e \
peed \
https://github.com/saforem2/ezpz"
n_alcf.sh
```

🙌 Acknowledgements

> This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.