Programming ML Algorithms for HPC Project 5 Report: HuggingFace and DeepSpeed on HPC

Zaitsev Anton
anton.zaitsev.001@student.uni.lu

Mahfoud Othmane
othmane.mahfoud.001@student.uni.lu

1 Introduction

This project explores the integration of DeepSpeed into the training of HuggingFace Transformer models on HPC. The primary objective is to use DeepSpeed's parallelization capabilities, such as data parallelism and optimizer sharding, to accelerate training time, reduce evaluation losses, and improve resource utilization when training deep learning models.

2 Environment Setup and File Management

2.1 Logging into the IRIS Cluster and File Transfer

The working environment is set on the IRIS cluster. We first log into the cluster and copy our project files into the designated directory. The following code snippet demonstrates the login process and file transfer from the local machine to IRIS:

```
# login
ssh -p 8022 azaitsev@access-iris.uni.lu
# make a local directory to store scripts
mkdir -p ~/project_5
# copy scripts from local machine to iris
scp -P 8022 project_5/* azaitsev@access-iris.uni.lu:~/project_5
```

Code 1: Logging in and copying project files to IRIS

2.2 DeepSpeed Environment Setup

The DeepSpeed environment is already configured on the cluster. To access it, we simply source the provided environment script in our SLURM files:

```
source /work/projects/ulhpc-tutorials/PS10-Horovod/env_ds.sh
```

Code 2: Sourcing the DeepSpeed environment

To run SLURM jobs in batch mode, we call:

```
sbatch project_5/launch_slurm_llm.sh
```

Code 3: Submitting SLURM jobs

3 LLM Tasks Overview

We focused on training a single large language model (LLM), specifically the `"facebook/opt-125m"` model, on the `"yelp_review_full"` dataset.

For each dataset configuration, we used `2048` data instances for both the training and evaluation sets by specifying `n_rows = 2048` in the `.py` scripts.

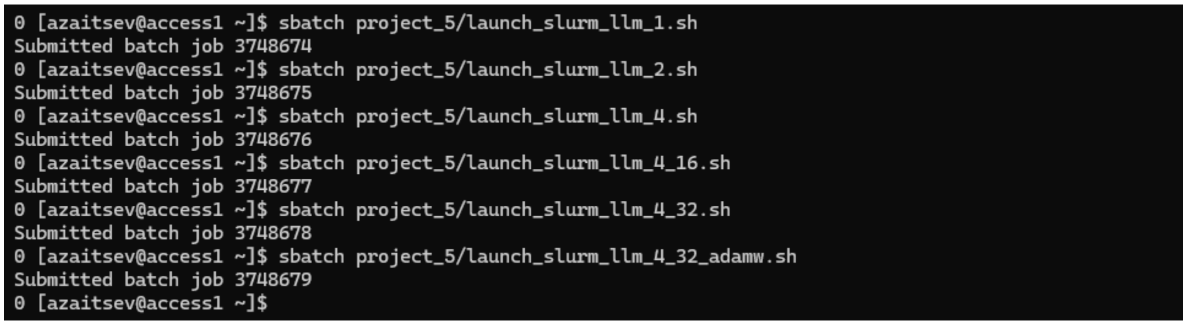
We perform a total of 6 tasks, where we vary the number of GPUs, batch sizes, and optimizer used:

- Task 1: 1 GPU, `batch_size = 4`
- Task 2: 2 GPUs, `batch_size = 4`
- Task 3: 4 GPUs, `batch_size = 4`
- Task 4: 4 GPUs, `batch_size = 16`
- Task 5: 4 GPUs, `batch_size = 32`
- Task 6: 4 GPUs, `batch_size = 32`, with `AdamW` optimizer

By varying these parameters, we hope to understand the impact of scaling the number of GPUs, increasing batch sizes, and switching optimizers on the training runtime and convergence.

4 Submitting Jobs on IRIS

To run the SLURM jobs on IRIS cluster, we used `batch-mode` by simply calling `sbatch <slurm_script_path>` (see Figure 1). In the SLURM script, we define the number of nodes to use for the job, number of GPUs, the source environment and which Python script to run.



```
0 [azaitsev@access1 ~]$ sbatch project_5/launch_slurm_llm_1.sh
Submitted batch job 3748674
0 [azaitsev@access1 ~]$ sbatch project_5/launch_slurm_llm_2.sh
Submitted batch job 3748675
0 [azaitsev@access1 ~]$ sbatch project_5/launch_slurm_llm_4.sh
Submitted batch job 3748676
0 [azaitsev@access1 ~]$ sbatch project_5/launch_slurm_llm_4_16.sh
Submitted batch job 3748677
0 [azaitsev@access1 ~]$ sbatch project_5/launch_slurm_llm_4_32.sh
Submitted batch job 3748678
0 [azaitsev@access1 ~]$ sbatch project_5/launch_slurm_llm_4_32_adamw.sh
Submitted batch job 3748679
0 [azaitsev@access1 ~]$
```

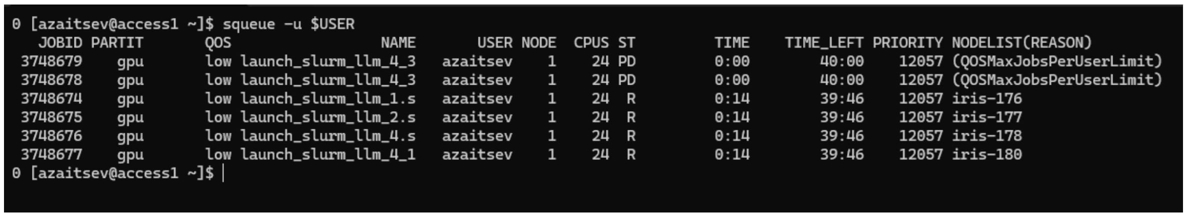
Figure 1: Submitting jobs on IRIS cluster for each task.

4.1 Checking Job Status

To check the status of submitted jobs on the IRIS cluster, we used `squeue -u $USER` command (see Figure 2).

```
#!/bin/sh -l
#SBATCH --partition=gpu
#SBATCH --gpus-per-node 1
#SBATCH -c 24
#SBATCH -t 40
#SBATCH -N 1
#SBATCH --export=ALL
# get host name
hosts_file="hosts_llm_1.txt"
scontrol show hostname $SLURM_JOB_NODELIST > $hosts_file
# collect public key and accept them
while read -r node; do
    ssh-keyscan "$node" >> ~/.ssh/known_hosts
done < "$hosts_file"
# create the host file containing node names and the number of GPUs
function makehostfile() {
perl -e '$slots=split␣/,/,␣$ENV{"SLURM_STEP_GPUS"};
$slots=4␣if␣$slots==0;
@nodes␣=␣split␣/\n/,␣qx[scontrol␣show␣hostnames␣$ENV{"SLURM_JOB_NODELIST"}];
print␣map␣{␣"$b$_␣slots=$slots\n"␣}␣@nodes'
}
makehostfile > hostfile_llm_1
source /work/projects/ulhpc-tutorials/PS10-Horovod/env_ds.sh
# launch HuggingFace+DeepSpeed code
deepspeed --num_gpus 1 --num_nodes 1  --hostfile hostfile_llm_1 project_5/
   LLM_1.py > 1_llm.txt
```

Code 4: SLURM script with 1 node and 1 GPU

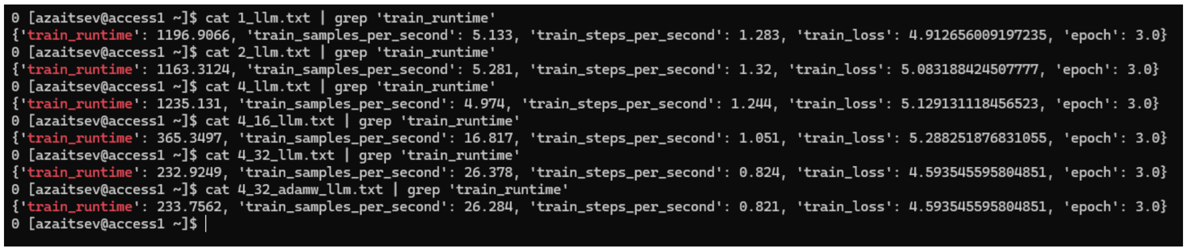


```
0 [azaitsev@access1 ~]$ squeue -u $USER
   JOBID PARTIT       QOS                 NAME       USER NODE  CPUS ST         TIME    TIME_LEFT PRIORITY NODELIST(REASON)
 3748679    gpu       low launch_slurm_llm_4_3   azaitsev    1    24 PD         0:00        40:00    12057 (QOSMaxJobsPerUserLimit)
 3748678    gpu       low launch_slurm_llm_4_3   azaitsev    1    24 PD         0:00        40:00    12057 (QOSMaxJobsPerUserLimit)
 3748674    gpu       low launch_slurm_llm_1.s   azaitsev    1    24  R         0:14        39:46    12057 iris-176
 3748675    gpu       low launch_slurm_llm_2.s   azaitsev    1    24  R         0:14        39:46    12057 iris-177
 3748676    gpu       low launch_slurm_llm_4.s   azaitsev    1    24  R         0:14        39:46    12057 iris-178
 3748677    gpu       low launch_slurm_llm_4_1   azaitsev    1    24  R         0:14        39:46    12057 iris-180
0 [azaitsev@access1 ~]$ |
```

Figure 2: Checking the status of submitted jobs.

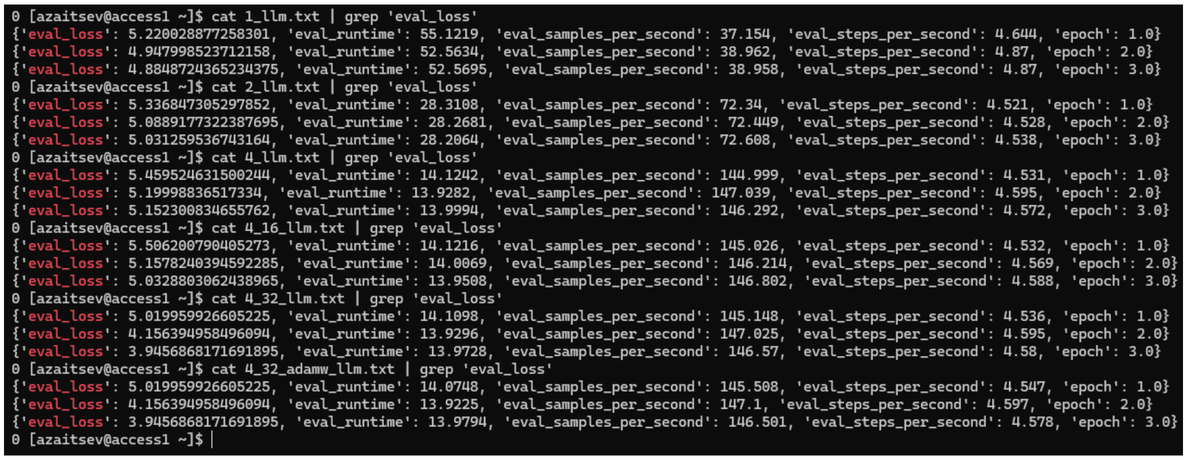
5 Results

In this section, we will present and analyze the results of the experiments. The results will include training runtimes (see Figure 3) and evaluation losses (see Figure 4) for each of the 6 tasks.



```
0 [azaitsev@access1 ~]$ cat 1_llm.txt | grep 'train_runtime'
{'train_runtime': 1196.9066, 'train_samples_per_second': 5.133, 'train_steps_per_second': 1.283, 'train_loss': 4.912656009197235, 'epoch': 3.0}
0 [azaitsev@access1 ~]$ cat 2_llm.txt | grep 'train_runtime'
{'train_runtime': 1163.3124, 'train_samples_per_second': 5.281, 'train_steps_per_second': 1.32, 'train_loss': 5.083188424507777, 'epoch': 3.0}
0 [azaitsev@access1 ~]$ cat 4_llm.txt | grep 'train_runtime'
{'train_runtime': 1235.131, 'train_samples_per_second': 4.974, 'train_steps_per_second': 1.244, 'train_loss': 5.129131118456523, 'epoch': 3.0}
0 [azaitsev@access1 ~]$ cat 4_16_llm.txt | grep 'train_runtime'
{'train_runtime': 365.3497, 'train_samples_per_second': 16.817, 'train_steps_per_second': 1.051, 'train_loss': 5.288251876831055, 'epoch': 3.0}
0 [azaitsev@access1 ~]$ cat 4_32_llm.txt | grep 'train_runtime'
{'train_runtime': 232.9249, 'train_samples_per_second': 26.378, 'train_steps_per_second': 0.824, 'train_loss': 4.593545595804851, 'epoch': 3.0}
0 [azaitsev@access1 ~]$ cat 4_32_adamw_llm.txt | grep 'train_runtime'
{'train_runtime': 233.7562, 'train_samples_per_second': 26.284, 'train_steps_per_second': 0.821, 'train_loss': 4.593545595804851, 'epoch': 3.0}
0 [azaitsev@access1 ~]$ |
```

Figure 3: Training runtimes for tasks 1-6.



```
0 [azaitsev@access1 ~]$ cat 1_llm.txt | grep 'eval_loss'
{'eval_loss': 5.220028877258301, 'eval_runtime': 55.1219, 'eval_samples_per_second': 37.154, 'eval_steps_per_second': 4.644, 'epoch': 1.0}
{'eval_loss': 4.947998523712158, 'eval_runtime': 52.5634, 'eval_samples_per_second': 38.962, 'eval_steps_per_second': 4.87, 'epoch': 2.0}
{'eval_loss': 4.8848724365234375, 'eval_runtime': 52.5695, 'eval_samples_per_second': 38.958, 'eval_steps_per_second': 4.87, 'epoch': 3.0}
0 [azaitsev@access1 ~]$ cat 2_llm.txt | grep 'eval_loss'
{'eval_loss': 5.336847305297852, 'eval_runtime': 28.3108, 'eval_samples_per_second': 72.34, 'eval_steps_per_second': 4.521, 'epoch': 1.0}
{'eval_loss': 5.0889177322387695, 'eval_runtime': 28.2681, 'eval_samples_per_second': 72.449, 'eval_steps_per_second': 4.528, 'epoch': 2.0}
{'eval_loss': 5.031259536743164, 'eval_runtime': 28.2064, 'eval_samples_per_second': 72.608, 'eval_steps_per_second': 4.538, 'epoch': 3.0}
0 [azaitsev@access1 ~]$ cat 4_llm.txt | grep 'eval_loss'
{'eval_loss': 5.459524631500244, 'eval_runtime': 14.1242, 'eval_samples_per_second': 144.999, 'eval_steps_per_second': 4.531, 'epoch': 1.0}
{'eval_loss': 5.19998836517334, 'eval_runtime': 13.9282, 'eval_samples_per_second': 147.039, 'eval_steps_per_second': 4.595, 'epoch': 2.0}
{'eval_loss': 5.152300834655762, 'eval_runtime': 13.9994, 'eval_samples_per_second': 146.292, 'eval_steps_per_second': 4.572, 'epoch': 3.0}
0 [azaitsev@access1 ~]$ cat 4_16_llm.txt | grep 'eval_loss'
{'eval_loss': 5.506200790405273, 'eval_runtime': 14.1216, 'eval_samples_per_second': 145.026, 'eval_steps_per_second': 4.532, 'epoch': 1.0}
{'eval_loss': 5.1578240394592285, 'eval_runtime': 14.0069, 'eval_samples_per_second': 146.214, 'eval_steps_per_second': 4.569, 'epoch': 2.0}
{'eval_loss': 5.0328803062438965, 'eval_runtime': 13.9508, 'eval_samples_per_second': 146.802, 'eval_steps_per_second': 4.588, 'epoch': 3.0}
0 [azaitsev@access1 ~]$ cat 4_32_llm.txt | grep 'eval_loss'
{'eval_loss': 5.019959926605225, 'eval_runtime': 14.1098, 'eval_samples_per_second': 145.148, 'eval_steps_per_second': 4.536, 'epoch': 1.0}
{'eval_loss': 4.156394958496094, 'eval_runtime': 13.9296, 'eval_samples_per_second': 147.025, 'eval_steps_per_second': 4.595, 'epoch': 2.0}
{'eval_loss': 3.9456868171691895, 'eval_runtime': 13.9728, 'eval_samples_per_second': 146.57, 'eval_steps_per_second': 4.58, 'epoch': 3.0}
0 [azaitsev@access1 ~]$ cat 4_32_adamw_llm.txt | grep 'eval_loss'
{'eval_loss': 5.019959926605225, 'eval_runtime': 14.0748, 'eval_samples_per_second': 145.508, 'eval_steps_per_second': 4.547, 'epoch': 1.0}
{'eval_loss': 4.156394958496094, 'eval_runtime': 13.9225, 'eval_samples_per_second': 147.1, 'eval_steps_per_second': 4.597, 'epoch': 2.0}
{'eval_loss': 3.9456868171691895, 'eval_runtime': 13.9794, 'eval_samples_per_second': 146.501, 'eval_steps_per_second': 4.578, 'epoch': 3.0}
0 [azaitsev@access1 ~]$ |
```

Figure 4: Validation losses per epoch for tasks 1-6.

From the results, we conclude that increasing the number of GPUs for the configured environment does not decrease the training runtime, which, in general, should not be the case. Changing the optimizer from `Adam` to `AdamW` also did not produce any substantial improvements in the runtime and losses . However, increasing the batch size consistently decreased training runtime and evaluation losses per epoch, which is a desired result.

6 Conclusion

This project explored the use of the DeepSpeed library for efficient training of HuggingFace Transformer LLM models under various conditions. We experimented with different GPU counts, batch sizes, and optimizers to improve training time and scalability. Although adjusting the number of GPUs and switching optimizers from `Adam` to `AdamW` did not demonstrate notable improvements, increasing the batch size significantly reduced the overall training runtime.