The methodology for performance evaluation of open-source solutions and IRONBYTE proprietary software solutions

Table of Contents

1. Introduction 2

1.1 Purpose of Testing 2
1.2 Technical Equipment vs. Infrastructure 2
1.3 Operating Environment 2
1.4 Description of the Models Used 3
1.5 Description of Standard Tasks 3
1.6 Training Methods 3
2.1 Testing Phases 4

2. Procedure for Conducting Test Trials 5

2.1 First Phase (Open-source) 5
2.2 Second Phase of Testing (IRONBYTE Proprietary Software Solutions) 13

3. Test protocol 19

3.1. Conclusions *19*

3.2. Appendices *22*

1. Introduction

This document outlines the methodology for conducting test trials and describes the procedure for evaluating the effectiveness of IRONBYTE proprietary software solutions in comparison with open-source solutions.

1.1 Purpose of Testing

The purpose of the testing is to:

- Perform standard computational tasks in the field of LLM using IRONBYTE proprietary software solutions and open-source solutions.

1.2 Technical Equipment vs. Infrastructure

Technical Equipment:

- Rig Nvidia H100 – a node with 8 Nvidia H100 GPUs (80GB), hereinafter referred to as the node;
- Rig Nvidia H100 *2 – a cluster of 2 nodes (Rig0, Rig1) with 8 Nvidia H100 GPUs (80GB) on each node, hereinafter referred to as the cluster.

Network parameters:

- A 10 Gbit/s network providing communication between the nodes;
- An 80 Gbit/s network providing communication between the nodes for open source (2 nodes).

1.3 Operating Environment

Each node is prepared for autonomous application execution (all necessary libraries, drivers, and services are pre-installed).

Software composition:

1. OS AlmaLinux 9.5;

2. NVIDIA drivers, version 550 or higher;
3. CUDA libraries, version 12.2 or higher;
4. Container service (Docker);
5. A set of containers containing necessary ML libraries and frameworks (python/torch/numpy/etc);
6. Pre-loaded models and datasets.

Source data for running ML tasks (pre-installed):

1. Orchestrator for distributed training using IRONBYTE proprietary software solutions;
2. Containers with IRONBYTE proprietary software solutions;
3. Containers with open-source solutions.

1.4 Description of the Models Used

The following large language models were used for the tests:

1. **Llama-2-70b-chat-hf** (quantized to 4-bit), hereinafter referred to as Llama2-70B;
2. **Llama-2-7b-chat-hf** (original size), hereinafter referred to as Llama2-7B.

1.5 Description of Standard Tasks

The following three standard tasks were chosen for testing the computational nodes:

1. Pretraining of the model on the fineweb-edu dataset;
2. Finetuning of the model on the databricks-dolly-15k dataset.
3. Model inference loading and text generation.

1.6 Training Methods

1.6.1 Training Using open-source Solutions:

1. Model distribution across all GPUs (balanced mode);

2. FullyShardedDataParallel (FSDP) mode for data and model parameter segmentation. In accelerator mode (experimentally), it supports operation across multiple nodes. This solution is applicable exclusively to LLM models built on PyTorch and published on the Hugging Face platform. This mode requires high network communication speeds.

1.6.2 Training Using IRONBYTE Proprietary Software Solutions:

1. Model and dataset distribution across groups of GPUs and nodes, with results mixed in the final computation stage (IRONBYTE share protocol). This protocol is designed for networks with low speeds and high network latency. This solution has no limitations and is applicable to any models.

2.1 Testing Phases

The testing is conducted in two phases:

First Phase:

1. Performing standard computational tasks using open-source solutions on 1 node.
2. Performing standard computational tasks using open-source solutions on 2 nodes of the cluster.

Second Phase:

1. Performing standard computational tasks using open-source solutions on 2 nodes of the cluster;
2. *Optionally: Performing standard computational tasks using IRONBYTE proprietary software solutions on 1 node.*

2. Procedure for Conducting Test Trials

2.1 First Phase (Open-source)

2.1.1 Testing the Llama2-70B Model

2.1.1.1 Procedure for Starting the Pretraining Task of Llama2-70B on One Node

The task is run on one node. For pretraining, the LLAMA2-70B model is used, trained on the fineweb-edu 10BT dataset for 1000 steps.

1. Command to Start the Container:

```
docker run -it --rm -v $HOME/pretrain70:/root -v /opt/dekubelocal:/share:ro --gpus all -w /root registry.cn-chengdu.aliyuncs.com/ai-palm/rrllm-cu124:pretrain.004
```

2. Running the Command for Pretraining Without Creating an "Empty" Model[1]:

```
time python3 /wrk/run_clm_no_trainer.py --dataset_name=/share/datasets/fineweb-edu/sample/10BTsingle/ --config_name=/wrk/70BlankModel --max_train_steps=1000  --output_dir=/root/result70b  --blank_model_name_or_path=/share/models/LLAMA2/metaAi/TMP70  --use_quantization=Yes
```

The result of execution is:

1. A trained model of 1000 steps, saved in the directory $HOME/pretrain70s/result70b.
2. The recorded time spent on the pretraining task.

To monitor the utilization of computational resources, the nvtop utility is used, if necessary.

[1] "Empty" model — a model with randomly initialized weights. It can be created by specifying the following key: --create_model_from_config=True.

2.1.1.2 Procedure for Starting the Pretraining Task of Llama2-70B in FSDP Mode on the Cluster

The task is run on two nodes of the cluster. For pretraining, the LLAMA2-70B model is used, trained on the fineweb-edu 10BT dataset for 100 [2] steps.

1. Preparation Steps (Optional), Cloning the Launch Files.
   Since the launch procedure is quite complex, the task is run using Docker Compose. It is necessary to clone the files onto each node, and in the Compose file, specify the correct node names and their IP addresses.

```
GIT_SSL_NO_VERIFY=true git clone https://gitlab.clive.tk/clients/job-manifests-public
```

2. Command to Start the Container in Interactive[3] Mode on Node rig0:

```
Cd Llama2-70B-fsdp-twonode

docker compose -f interactive-compose-a.yaml up &

docker exec -it llama2a
```

3. Command to Start the Container on Node rig1:

```
Cd Llama2-70B-fsdp-twonode

docker compose -f interactive-compose-b.yaml up &

docker exec -it llama2b
```

4. Running the Command for Pretraining on Two Nodes:

```
/mnt/wrk/entrypoint.sh
```

The result of execution is:

1. A trained model of 100 steps, saved in the directory `$HOME/pretrain70/result70b-fsdp`.

---

[2] The choice of a small number of training steps is due to the fact that the network connectivity of the nodes at 10 Gbit/s is insufficient to ensure the intensive transfer of the computed backpropagated gradient during each iteration of the error function calculation. One step in this configuration takes 375 seconds to complete.

[3] To disable interactive mode, you need to remove the -f key from the parameter.

2. The recorded time spent on the pretraining task.

To monitor the utilization of computational resources, the nvtop, bmon utility is used, if necessary.

2.1.1.3 Procedure for Starting the Finetuning Task of Llama2-70B

The task is run on one node. For the finetuning task, the Llama2-70B model is used, finetuned on the databricks-dolly dataset for 500 steps.

1. Command to Start the Container:

```
docker run -it --rm -v $HOME/finetune70:/root -v /opt/dekubelocal:/share:ro --gpus all --pull=always registry.cn-chengdu.aliyuncs.com/ai-palm/rrllm-cu122:finetune
```

2. Command to start the finetuning task:[4]

```
time python3 /wrk/ft_70full_n_h100_1ep.py --model_dir=/share/models/LLAMA2/metaAi/Llama-2-70b-chat-hf --output_dir=/root/full --num_step=500  --dataset_pth=/share/datasets/databricks/databricks-dolly-15k.jsonl
```

The result of execution is:

1. A finetuned model of 500 steps, saved in the directory $HOME/finetune70/full/.
2. The recorded time spent on the finetuning task.

To monitor the utilization of computational resources, the nvtop utility is used, if necessary.

2.1.1.4 Procedure for Starting the Finetuning Task of Llama2-70B in FSDP Mode on the Cluster

The task is run on two nodes of the cluster. For the finetuning task, the Llama2-70B model is used, finetuned on the databricks-dolly dataset for 100 steps.

1. Preparation steps (optional), cloning the launch files.

---

[4] To specify the training time in epochs, you need to use the --num_epoch key

Since the launch procedure is quite complex, the task is run using Docker Compose. It is necessary to clone the files onto each node, and in the Compose file, specify the correct node names and their IP addresses.

```
GIT_SSL_NO_VERIFY=true git clone https://gitlab.clive.tk/clients/job-manifests-public
```

2. Command to start the container in interactive[5] mode on node rig0:

```
Cd Llama2-70B-fsdp-twonode-ft
docker compose -f interactive-compose-a.yaml up &
docker exec -it llama2a
```

3. Command to start the container on node rig1:

```
Cd Llama2-70B-fsdp-twonode-ft
docker compose -f interactive-compose-b.yaml up &
docker exec -it llama2b
```

4. Command to start the finetuning task на двух узлах кластера:

```
/mnt/wrk/entrypoint.sh
```

The result of execution is:

1. A finetuned model of 100 steps, saved in the directory $HOME/finetune70/result70b-fsdp.
2. The recorded time spent on the finetuning task.

To monitor the utilization of computational resources, the nvtop, bmon utility is used, if necessary.

2.1.1.5 Procedure for Starting the Inference Task of Llama2-70B on One Node

The task is run on one node. For the inference task, the Llama2-70B model is used, which generates a specified number of tokens in response to a fixed question.

[5] To disable interactive mode, you need to remove the -f key from the parameter.

Task execution parameters:

- Fixed question: **Can there be life somewhere in the Solar System outside Earth?**
- Number of question repetitions: **30**;
- Number of tokens in each response: **от 100 до 400** tokens (variable value);
- Total number of generated tokens: **7750.**

Metrics:

- The total time spent generating all tokens is calculated.
- The generation speed of one token is determined.
- The number of tokens generated per second is computed.

Execution procedure:

1. Command to start the container:

```
docker run -ti --rm -v /opt/dekubelocal/models/LLAMA2/metaAi/Llama-2-70b-chat-hf:/model -v wrk -e MODEL_DIR='/model' --gpus all registry.cn-chengdu.aliyuncs.com /ai-palm/env:torch24cu124.002
```

2. Command to start the inference task:

```
export MAX_TOKENS_LST='100,110,120,130,140,150,160,170,180,190,200,210,220,230,240,250,260,270,280,290,300,310,320,330,340,350,360,370,380,390,400' QUESTION='Can there be life somewhere in the Solar System outside Earth?'

python3 in_fer.py
```

The result of execution is:

1. The time taken to generate a response for each query.
2. The time taken to generate responses depending on their length.
3. The average time taken to generate one token.

To monitor the utilization of computational resources, the nvtop utility is used, if necessary.

2.1.2 Testing the Llama2-7B Model

2.1.2.1 Procedure for Starting the Pretrain Task of Llama2-7B

The task is run on one node. For the pretraining task, the Llama2-7B model (without quantization) is used, trained on the fineweb-edu 10BT dataset for 1000 steps.

1. Command to Start the Container:

```
docker run -it --rm -v $HOME/pretrain:/root -v /opt/dekubelocal:/share:ro --
gpus all -w /root registry.cn-chengdu.aliyuncs.com/ai-palm/rrllm-
cu124:pretrain.004
```

2. Command to start pretraining without creating an "empty" model:

```
time python3 /wrk/run_clm_no_trainer.py --
dataset_name=/share/datasets/fineweb-edu/sample/10BTsingle/ --
config_name=/wrk/7BlankModel  --max_train_steps=1000  --
output_dir=/root/result7b  --
blank_model_name_or_path=/share/models/LLAMA2/metaAi/TMP7
```

The result of execution is

1. A trained model of 1000 steps, saved in the directory $HOME/pretrain/result7b/.
2. The recorded time spent on the pretraining task.

To monitor the utilization of computational resources, the nvtop utility is used, if necessary.

2.1.2.2 Procedure for Starting the Pretraining Task of Llama2-7B in FSDP Mode on the Cluster

The task is run on two nodes of the cluster. For pretraining, the Llama2-70B model is used, trained on the fineweb-edu 10BT dataset for 100 steps.

1. Preparation Steps (Optional), Cloning the Launch Files.
   Since the launch procedure is quite complex, the task is run using Docker Compose. It is necessary to clone the files onto each node, and in the Compose file, specify the correct node names and their IP addresses.

```
GIT_SSL_NO_VERIFY=true git clone https://gitlab.clive.tk/clients/job-
manifests-public
```

2. Command to start the container in interactive[6] mode on node rig0:

```
Cd llama2-7b-fsdp-twonode

docker compose -f interactive-compose-a.yaml up &

docker exec -it llama2a
```

3. Command to start the container on node rig1:

```
Cd llama2-7b-fsdp-twonode

docker compose -f interactive-compose-b.yaml up &

docker exec -it llama2b
```

4. Command to start pretraining on two nodes of the cluster:

```
/mnt/wrk/entrypoint.sh
```

The result of execution is:

1. A trained model of 100 steps, saved in the directory `$HOME/pretrain7/result7b-fsdp`.
2. The recorded time spent on the pretraining task.

To monitor the utilization of computational resources, the nvtop, bmon utility is used, if necessary.

2.1.2.3 Procedure for Starting the Finetuning Task of Llama2-7B

The task is run on one node. For the finetuning task, the Llama2-7B model (without quantization) is used, which is finetuned on the databricks-dolly dataset for 1 epoch.

1. Command to Start the Container:

```
docker run --rm -it --name finetune7 --gpus all -v /opt/dekubelocal:/share:ro
-v $HOME/finetune:/root registry.cn-chengdu.aliyuncs.com/ai-palm/rrllm-
cu122:finetune.001
```

2. Command to start the finetuning task:

```
time python3 ft_7full_many_gpu.py --
model_dir=/share/models/LLAMA2/metaAi/Llama-2-7b-chat-hf/ --num_epoch=1 --
```

[6] To disable interactive mode, you need to remove the -f key from the parameter.

```
dataset_pth=/share/datasets/databricks/databricks-dolly-15k.jsonl --
output_dir=/root/result7b
```

The result of execution is:

1. The finetuned model of 1 epoch, saved in the directory $HOME/finetune/result7b.
2. The recorded time spent on the finetuning task.

To monitor the utilization of computational resources, the nvtop utility is used, if necessary.

2.1.2.4 Procedure for Starting the Inference Task of Llama2-7B

The task is run on one node. For the inference task, the Llama2-7B model is used, which generates a specified number of tokens in response to a fixed question.

Task execution parameters:

- Fixed question: **Can there be life somewhere in the Solar System outside Earth?**
- Number of question repetitions: **30**;
- Number of tokens in each response: **from 100 to 400** tokens (variable value);
- Total number of generated tokens: **7750.**

Metrics:

- The total time spent generating all tokens is calculated.
- The generation speed of one token is determined.
- The number of tokens generated per second is computed.

Execution procedure:

1. Command to Start the Container:

```
docker run -ti --rm -v /opt/dekubelocal/models/LLAMA2/metaAi/Llama-2-7b-chat-
hf:/model -v wrk -e MODEL_DIR='/model' --gpus all registry.cn-
chengdu.aliyuncs.com /ai-palm/env:torch24cu124.002
```

2. Command to start the inference task:

```
export
MAX_TOKENS_LST='100,110,120,130,140,150,160,170,180,190,200,210,220,230,240,2
50,260,270,280,290,300,310,320,330,340,350,360,370,380,390,400' QUESTION='Can
there be life somewhere in the Solar System outside Earth?'

python3 in_fer.py
```

The result of execution is:

1. The time taken to generate a response for each query.
2. The time taken to generate responses depending on their length.
3. The average time taken to generate one token.

To monitor the utilization of computational resources, the nvtop utility is used, if necessary.

2.2 Second Phase of Testing (IRONBYTE Proprietary Software Solutions)

2.2.1 Testing the Llama2-70B Model

2.2.1.1 Procedure for Starting the Pretraining Task of Llama2-70B on the Cluster

The task is run on two nodes of the cluster. For the pretraining task, the Llama2-70B model is used, trained on the fineweb-edu 10BT dataset for 1000 steps.

1. Command to Start the Orchestrator on One of the Nodes:

```
python ./orch.py --gpu_type H100 -n rig0.IRONBYTE:2x4 rig1.IRONBYTE:2x4 -t
pretrain --num_step 1000 -m rig0.IRONBYTE --dataset_pth
/opt/dekubelocal/datasets/fineweb-edu/sample/10BT/000_00000.parquet
```

2. **Optionally, the task can be run on a single computational node. To do this, simply remove the second node from the launch parameters. For example*:

```
python ./orch.py --gpu_type H100 -n rig0.IRONBYTE:2x4 -t pretrain --num_step
1000 -m rig0.IRONBYTE --dataset_pth /opt/dekubelocal/datasets/fineweb-
edu/sample/10BT/000_00000.parquet
```

The result of execution is:

1. A trained model of 1000 steps, saved in the directory /opt/pyorch/{timestamp}/ on the node where the task was launched.
2. The recorded time spent on executing the pretraining task using IRONBYTE proprietary software solutions.

To monitor the utilization of computational resources, the nvtop utility is used, if necessary, on all nodes.

2.2.1.2 Procedure for Starting the Finetuning Task of Llama2-70B

The task is run on two nodes of the cluster. For the finetuning task, the Llama2-70B model is used, finetuned on the databricks-dolly dataset for 500 steps.

1. Command to start the orchestrator on one of the nodes:

```
python ./orch.py --gpu_type H100 -n rig0.IRONBYTE:2x4 rig1.IRONBYTE:2x4 -t
finetune --num_step 500 -m rig0.IRONBYTE
```

2. **Optionally, the task can be run on a single computational node. To do this, simply remove the second node from the launch parameters. For example*:

```
python ./orch.py --gpu_type H100 -n rig0.IRONBYTE:2x4 -t finetune --num_step
500 -m rig0.IRONBYTE
```

The result of execution is:

1. A finetuned model of 500 steps, saved in the directory /opt/pyorch/{timestamp}/ on the node where the task was launched.
2. The recorded time spent on executing the finetuning task using IRONBYTE proprietary software solutions.

To monitor the utilization of computational resources, the nvtop utility is used, if necessary.

2.2.1.3 Procedure for Starting the Inference Task of Llama2-70B

The task is run on two nodes of the cluster. For the inference task, the Llama2-70B model is used, which generates a specified number of tokens in response to the fixed question.

Task execution parameters:

- Fixed question: **Can there be life somewhere in the Solar System outside Earth?**
- Number of question repetitions: **30**;
- Number of tokens in each response: **от 100 до 400** tokens (variable value);
- Total number of generated tokens: **7750.**

Execution procedure:

1. Command to start multi-inference on two nodes of the cluster:

```
./orch.py -t inference -n rig0.IRONBYTE:4x2 rig1.IRONBYTE:4x2 --model_dir
/opt/dekubelocal/models/LLAMA2/metaAi/Llama-2-70b-chat-hf --max_tokens_from
100 --max_tokens_to 401 --question 'Can there be life somewhere in the Solar
System outside Earth?'
```

2. Command to start inference on one node (optional, for performance comparison with the open-source solution in a single thread):

```
./orch.py -t inference -n rig0.IRONBYTE:1x8 --model_dir
/opt/dekubelocal/models/LLAMA2/metaAi/Llama-2-70b-chat-hf --max_tokens_from
100 --max_tokens_to 401 --question 'Can there be life somewhere in the Solar
System outside Earth?'
```

The result of execution is:

1. The time taken to generate a response for each query.
2. The time taken to generate responses depending on their length.
3. The average time taken to generate one token.

To monitor the utilization of computational resources, the `nvtop` utility is used, if necessary.

2.2.2 Testing the Llama2-7B Model

2.2.2.1 Procedure for Starting the Pretraining Task of Llama2-7B on the Cluster

The task is run on two nodes of the cluster. For the pretraining task, the Llama2-7B model is used, trained on the fineweb-edu 10BT dataset for 2000 steps.

1. Command to start the task orchestrator on two nodes of the cluster:

```
python ./orch.py --gpu_type h100 -n rig0.IRONBYTE:4x2 rig1.IRONBYTE:4x2 -t
pretrain --num_step 2000 -m rig0.IRONBYTE --dataset_pth
/opt/dekubelocal/datasets/databricks
```

2. **Optionally, the task can be run on a single computational node. To do this, simply remove the second node from the launch parameters. For example*:

```
python ./orch.py --gpu_type H100 -n rig0.IRONBYTE:4x2 -t pretrain --num_step
2000 -m rig0.IRONBYTE --dataset_pth /opt/dekubelocal/datasets/databricks
```

The result of execution is:

1. A finetuned model of 2000 steps, saved in the directory /opt/pyorch/{timestamp}/ on the node where the task was launched.
2. The recorded time spent on executing the pretraining task using IRONBYTE proprietary software solutions.

To monitor the utilization of computational resources, the nvtop utility is used, if necessary.

2.2.2.2 Procedure for Starting the Finetuning Task of Llama2-7B on the Cluster

The task is run on two nodes of the cluster. For the finetuning task, the Llama2-7B model is used, trained on the databricks-dolly dataset for 1 epoch.

1. Command to start the task orchestrator on two nodes of the cluster:

```
python ./orch.py --gpu_type h100 -n rig0.IRONBYTE:4x2 rig1.IRONBYTE:4x2 -t
finetune --num_epoch 1 -m rig0.IRONBYTE --num_epoch 1 --dataset_pth
/opt/dekubelocal/datasets/databricks
```

2. **Optionally, the task can be run on a single computational node. To do this, simply remove the second node from the launch parameters. For example:*

```
python ./orch.py --gpu_type h100 -n rig0.IRONBYTE:4x2 -t finetune --num_epoch
1 -m rig0.IRONBYTE --num_epoch 1 --dataset_pth
/opt/dekubelocal/datasets/databricks
```

The result of execution is:

1. A finetuned model of 1 epoch, saved in the directory /opt/pyorch/{timestamp}/ on the node where the task was launched.
2. The recorded time spent on executing the finetuning task using IRONBYTE proprietary software solutions.

To monitor the utilization of computational resources, the nvtop utility is used, if necessary.

2.2.2.3 Procedure for Starting the Inference Task of Llama2-7B

The task is run on two nodes of the cluster. For the inference task, the Llama2-7B model is used, which generates a specified number of tokens in response to a fixed question.

Task execution parameters:

- Fixed question: **Can there be life somewhere in the Solar System outside Earth?**
- Number of question repetitions: **30**;
- Number of tokens in each response: **от 100 до 400** tokens (variable value);
- Total number of generated tokens: **7750.**

Execution procedure:

1. Command to start multi-inference on two nodes of the cluster:

```
./orch.py -t inference -n rig0.IRONBYTE:5x2 rig1.IRONBYTE:5x2 --model_dir
/opt/dekubelocal/models/LLAMA2/metaAi/Llama-2-7b-chat-hf --max_tokens_from
100 --max_tokens_to 401 --question 'Can there be life somewhere in the Solar
System outside Earth?'
```

2. Command to start inference on one node (optional, for performance comparison with the open-source solution in a single thread):

```
./orch.py -t inference -n rig0.IRONBYTE:1x8 --model_dir
/opt/dekubelocal/models/LLAMA2/metaAi/Llama-2-7b-chat-hf --max_tokens_from
100 --max_tokens_to 401 --question 'Can there be life somewhere in the Solar
System outside Earth?'
```

The result of execution is:

1. The time taken to generate a response for each query.
2. The time taken to generate responses depending on their length.
3. The average time taken to generate one token.

To monitor the utilization of computational resources, the nvtop utility is used, if necessary.

3. Test protocol

3.1. Conclusions

As part of the conducted tests, we analyzed the results of Pretraining, Finetuning, and Inference tasks for the Llama2-70B and Llama2-7B models using Open Source and IRONBYTE methods. For each task type, execution time was determined, and efficiency was calculated.

Based on the data presented in the tables, we computed speed ratio comparisons for IRONBYTE and Open-source solutions across all task types.

1. **Llama2-70B**

Pretraining Task (1000 steps)

1 node:

- The IRONBYTE solution achieved a speed of 29 minutes, which is 321% more efficient than the Open-source result of 93 minutes.

Cluster (2 nodes):

- The IRONBYTE solution achieved a speed of 20 minutes, which is 6,050% more efficient than the Open-source result of 1,210 minutes, using two nodes in IB DG Mode, 80G.
- The efficiency of the IRONBYTE solution was 290% compared to the Open-source result of 58 minutes in IB Conn Mode (Result other Cloud).

Finetuning Task (500 steps)

1 node:

- The IRONBYTE solution achieved a speed of 68 minutes, which is 182% more efficient than the Open-source result of 124 minutes.

Cluster (2 nodes):

- The IRONBYTE solution achieved a speed of 39 minutes, which is 1,377% more efficient than the Open-source result of 537 minutes, using two nodes in IB DG Mode, 80G.
- The efficiency of the IRONBYTE solution was 190% compared to the Open-source result of 74 minutes in IB Conn Mode (Result other Cloud).

Inference Task (Time to Generate 1 Token)

1 node:

- The IRONBYTE solution in multi-inference mode achieved a speed of 0.0139 seconds, which is 410% more efficient than the Open-source result of 0.057 seconds.
- In single-thread mode, the IRONBYTE solution achieved a speed of 0.054 seconds, which is 106% more efficient than the Open-source result of 0.057 seconds.

2. Llama2-7B:

Pretraining Task (2000 steps)

1 node:

- The IRONBYTE solution achieved a speed of 7 minutes, which is 1,714% more efficient than the Open-source result of 120 minutes.

Cluster (2 nodes):

- The IRONBYTE solution achieved a speed of 5 minutes, which is 9,840% more efficient than the Open-source result of 492 minutes, using two nodes in IB DG Mode, 80G.
- The efficiency of the IRONBYTE solution is 1,440% compared to the Open-source result of 72 minutes in IB Conn Mode (Result other Cloud).

Finetuning Task (1 epoch)

1 node:

- The IRONBYTE solution achieved a speed of 28 minutes, which is 1,161% more efficient than the Open-source result of 325 minutes.

Cluster (2 nodes):

- The IRONBYTE solution achieved a speed of 15 minutes, which is 25,000% more efficient than the Open-source result of 3,750 minutes, using two nodes in IB DG Mode, 80G.
- The efficiency of the IRONBYTE solution was 1,273% compared to the Open-Source result of 191 minutes in IB Conn Mode (Result other Cloud).

Inference Task (Time to Generate 1 Token)

1 node:

- The IRONBYTE solution in multi-inference mode achieved a speed of 0.0019 seconds, which is 1,053% more efficient than the Open-source result of 0.02 seconds.
- In single-thread mode, the IRONBYTE solution achieved a speed of 0.0175 seconds, which is 114% more efficient than the Open-source result of 0.02 seconds.

The data presented above shows that the IRONBYTE solution efficiently leverages the capabilities of H100 GPUs, significantly reducing task execution time and demonstrating optimizations that ensure high performance.

Thus, the use of the IRONBYTE solution significantly improves performance across all task types, making it a more advantageous choice for working with LLMs.

3.2. Appendices

Table №1. Test Results for the Quantized **Llama2-70B** Model: Pretraining and Finetuning.

Task Type	Parameters	Open source	IRONBYTE	Open source	Open source	IRONBYTE
		1 node	1 node	2 nodes (IB DG Mode, 80G)	2 nodes (IB Conn Mode)	2 nodes
Pretraining (min)					Result other Cloud	
	1000 steps	93	**29**	1210*****	58	**20**
	100 steps	10	**7**	110	6	X
The efficiency coefficient of Pretraining						
	1000 steps	3.2	**1**	60.5*****	2.9	**1**
	100 steps	1.4	**1**	-	-	-
Finetuning (min)						
	500 steps	124	68	537*****	74	39
	100 steps	26	10	133	15	X
The efficiency coefficient of Finetuning						
	500 steps	1.8	**1**	13.8*****	1.9	**1**
	100 steps	2.6	**1**	-	-	-

Table № 2. Inference testing results of the quantized **Llama2-70B** model.

Task Type	Parameters	Open source	IRONBYTE		Open source	IRONBYTE	
Inference (sec)		1 node	1 node	**Multi Inference (sec)**	1, 2 nodes	1 node	2 nodes
	Total Generation Time (sec)	439	**419**	Total Generation Time (sec)	No solution	107	54
	Time to Generate 1 Token (sec)	0.057	**0.054**	Time to Generate 1 Token (sec)	No solution	0.0139	0,0079
	Token Generation per 1 Second	17.6	**18.4**	Token Generation per 1 Second	No solution	71	131
The efficiency coefficient of Finetuning		The efficiency is calculated by comparing both the inference values and the multi-inference values					
	Total Generation Time (sec)	4.1	**3.9**		No solution	**1**	**-**
	Time to Generate 1 Token (sec)	4.1	**3.9**		No solution	**1**	**-**
	Token Generation per 1 Second	0.25	**0.26**		No solution	**1**	**-**

Table №3. Testing results on Pretraining and Finetuning tasks of **the Llama2-7B** Model (Without Quantization).

Task Type	Parameters	Open source	IRONBYTE	Open source	Open source	IRONBYTE
		1 node	1 node	2 nodes (IB DG Mode, 80G)	2 nodes (IB Conn Mode)	2 nodes
Pretraining (min)					Result other Cloud	
	2000 steps	120	**7**	492*	72	**5**
	1000 steps	62	**4**	235*	36	X
	100 steps	6	X	22	4	X
The efficiency coefficient of Pretraining						
	2000 steps	17.14	**1**	98.4*	14.4	**1**
	1000 steps	15.5	**1**	-	-	-
	100 steps	-	**-**	-	-	-
Finetuning (min)		Open source	IRONBYTE	Open source	Open source	IRONBYTE
	1 epoch	325	28	3750*	191	**15**
The efficiency coefficient of Finetuning						
	1 epoch	11.6	**1**	250	12.7	**1**

Table №4. Inference testing results of the **Llama2-7B** model (Without Quantization).

Task Type	Parameters	Open source	IRONBYTE		Open source	IRONBYTE	
Inference (sec)		1 node	1 node	**Multi Inference (sec)**	1, 2 nodes	1 node	2 nodes
	Total Generation Time (sec)	155	**136**	Total Generation Time (sec)	No solution	15	7
	Time to Generate 1 Token (sec)	0.02	**0.0175**	Time to Generate 1 Token (sec)	No solution	0.0019	0.0009
	Token Generation per 1 Second	49	**56**	Token Generation per 1 Second	No solution	515	1075
The efficiency coefficient of Finetuning		The efficiency is calculated by comparing both the inference values and the multi-inference values					
	Total Generation Time (sec)	10.33	**9**		No solution	**1**	**-**
	Time to Generate 1 Token (sec)	10.5	**9.21**		No solution	**1**	**-**
	Token Generation per 1 Second	0.095	**0.1**		No solution	**1**	**-**

« ***** » – The values marked are those set through calculations, due to the inability to reproduce such a long training process.

« **X** » – Values are not provided because for a small number of steps, the IRONBYTE algorithm is not recommended due to the time it takes to transmit the results and perform the result merging, which exceeds the time spent on the training procedure itself. The transmission and result merging time does not depend on the number of steps (epochs) specified.

« **-** » – The efficiency ratio calculation has not been performed due to the lack of results, see the note above.

«**No solution**» – The idea of multi-inference is not new, and there are many projects on GitHub that announce the scaling of the output task and are at various stages of development. However, these projects cannot be used as reference solutions, as they primarily achieve faster inference by caching queries and balancing context during request processing. In such solutions, isolating the "pure" time would take an unjustifiably long time. Therefore, we suggested using the speed characteristics of the reference code from HF as reference values.

IRONBYTE solutions represent a more low-level platform on top of which scalable and manageable solutions can be built to meet customer requirements.