

# Contents

<b>DeepSeek-R1 Distillation Best Practice</b>	<b>1</b>
1. LLAMA-Factory Installation . . . . .	1
2. Dataset Preparation . . . . .	2
2.1 Adopt MagPie Synthetic Dataset . . . . .	2
2.2 Create Your Own CoT data . . . . .	3
2.3 Prepare with your own domain-specific dataset . . . . .	5
3. Data Distillation: LoRA fine-tune from base model . . . . .	7
4. Merge LoRA Adapter with Base Model . . . . .	8
5. Test with Fine-tuned Model . . . . .	9
5.1 Evaluation tool . . . . .	9
5.2 Quality Analysis . . . . .	9
5.3 Benchmarking . . . . .	10
6. Fine-tune Model on Intel® Arc™ A770 Graphics . . . . .	11
6.1 Install LLAMA-Factory with Pytorch-Extension-for-Intel . . . . .	11
6.2 Dataset Preparation. . . . .	12
6.3 Set configurations to enable training on XPU plugin. . . . .	12
6.4 Run with train script as follows: . . . . .	13
6.5 Merge LoRA adapter with base model . . . . .	15
6.6 Test with fine-tuned model . . . . .	15
7. Reference . . . . .	15

## DeepSeek-R1 Distillation Best Practice

This guide walks through the best practices for distilling a base model with Chain-of-Thought (CoT) data, utilizing the LLAMA-Factory training framework.

### 1. LLAMA-Factory Installation

Before you start, make sure you have installed the following packages:

**Step 1.** Follow the instructions of [LLaMA-Factory](#), and build the environment:

```
python3 -m venv ~/python-env/llamafactory
source ~/python-env/llamafactory/bin/activate

git clone --depth 1 https://github.com/hiyouga/LLaMA-Factory.git
cd LLaMA-Factory
pip install -e ".[torch,metrics]"
```

**Step 2.** Install these packages (Optional, for CUDA only)

```
pip install deepspeed
pip install flash-attn --no-build-isolation
```

**Note:**

- If you want to use [FlashAttention-2](#), make sure your CUDA is 11.6 and above.

## 2. Dataset Preparation

To enhance models with CoT behavior, we will either:

- [Adopt\(sec 2.1 Adopt MagPie Synthetic Dataset\)](#) an open-sourced synthetic dataset, or
- [Create \(sec 2.2 Create Your Own CoT data\)](#) a custom CoT dataset

And then distill a base model using this dataset to improve its reasoning capabilities.

Besides, you can also leverage your own domain-specific dataset following [sec 2.3 Prepare with your own domain-specific dataset](#)

LLaMA-Factory supports training datasets defined under the `data` folder, with datasets formatted in either `alpaca` or `sharegpt` styles. To use a custom dataset, you will need to convert it into one of these supported formats. Please prepare your dataset as follows.

### 2.1 Adopt MagPie Synthetic Dataset

**Step 1.** Download existing CoT synthetic dataset from huggingface

**Dataset link:** [Magpie-Align/Magpie-Reasoning-V2-250K-CoT-Deepseek-R1-Llama-70B](#)

This dataset is generated by [Meta's Llama 3.1 70B Instruct](#), [Llama 3.3 70B Instruct](#) and [deepseek-ai/DeepSeek-R1-Distill-Llama-70B](#) using [Magpie framework](#). Specifically, the instructions are generated by Llama 3.1 70B Instruct and Llama 3.3 70B Instruct, and the responses are generated by DeepSeek-R1-Distill-Llama-70B. Please refer to the [paper](#) and [codebase](#) for implementation details.

**Step 2.** Convert to `sharegpt` format.

```
import json
from datasets import load_dataset

# Load the dataset
dataset = load_dataset(
    "Magpie-Align/Magpie-Reasoning-V2-250K-CoT-Deepseek-R1-Llama-70B")
dataset = dataset["train"]

# Filter dataset
## Change the filter conditions according to your needs
```

```
dataset = dataset.filter(lambda example: len(example['response']) <= 1024)

# Save as sharegpt format
with open("Magpie-Reasoning-V2-250K-CoT-Deepseek-R1-Llama-70B-response1024.json",
          'w') as f:
    json.dump(list(dataset), f, ensure_ascii=False, indent=4)
```

**Step 3.** Register CoT dataset LLaMA-Factory dataset\_info.json

```
cd LLaMA-Factory/data
vim dataset_info.json

...
# make sure the file is put under `LLaMA-Factory/data`
"deepseek-r1-distill-sample": {
    "file_name": "Magpie-Reasoning-V2-250K-CoT-Deepseek-R1-Llama-70B-response1024.json",
    "formatting": "sharegpt",
    "columns": {
        "messages": "conversations"
    }
}
...
}
```

## 2.2 Create Your Own CoT data

**Step 1.** Deploy Deepseek-R1 model serving based on llama\_cpp.server

If you already have a well-running [OpenAI Chat Completions API](#) deployment for Deepseek-R1 model inference, you can skip this step.

- Install llama\_cpp.server following [llama-cpp-python installation configuration](#)
- Start OpenAI Compatible Server:

```
python -m llama_cpp.server \
    --model "/path/to/Model/DeepSeek-R1" \
    --n_ctx 8192 \
    --n_gpu_layers -1 \
    --n_threads 8 \
    --n_batch 32 \
    --port 7701
```

**Note:**

- For more details about llama\_cpp.server usage, please find the tutorial in [llama-cpp](#)
- Select the appropriate [DeepSeek-R1](#) model based on your platform.

## Step 2. Generate CoT synthetic dataset

**Pre-requisite:** Place your questions or instructions in a list file, with each question or instruction on a separate line. Here is an example: `instruction.list`

```
what is the probability of rolling a 7 with two dice?  
Solve the equation  $\frac{3}{4}x + 2 = 10$ .
```

Then, generate the synthetic data using DeepSeek-R1 model deployed in Step 1. Remember to replace `base_url`, `your_serve_model`, `your_instruction_list_file` to the actual value:

```
import json  
import requests  
  
url = "https://{base_url}/chat/completions"  
serve_model = "{your_serve_model}"  
instruction_file = "{your_instruction_list_file}"  
header = {"Content-Type": "application/json"}  
  
instruction_list = []  
with open(instruction_file, 'r') as f:  
    lines = f.readlines()  
    instruction_list = [ln.strip() for ln in lines]  
  
data_list = []  
for instruction in instruction_list:  
    request_data = {  
        "model": serve_model,  
        "messages": [  
            {"role": "user", "content": f"{instruction}."}  
        ],  
        "max_tokens": 8192,  
        "temperature": 0.6,  
        "top_p": 0.95  
    }  
    result = requests.post(url, json=request_data, headers=header,  
                           timeout=10000, verify=False)  
    completion = json.loads(result.text)  
    response = completion['choices'][0]['message']['content']  
    data = {  
        "conversations": [  
            {  
                "from": "human",  
                "value": instruction  
            },  
            {
```

```

        "from": "gpt",
        "value": response
    }
]
}
data_list.append(data)

# Save as sharegpt format
with open("CoT-Deepseek-R1-Synthetic-data.json", 'w') as f:
    json.dump(data_list, f, ensure_ascii=False, indent=4)

```

**Step 3.** Register the generated CoT dataset LLaMA-Factory dataset\_info.json

```

cd LLaMA-Factory/data
vim dataset_info.json

...
# make sure the file is put under `LLaMA-Factory/data`
"deepseek-r1-distill-sample": {
    "file_name": "CoT-Deepseek-R1-Synthetic-data.json",
    "formatting": "sharegpt",
    "columns": {
        "messages": "conversations"
    }
}
}
...

```

### 2.3 Prepare with your own domain-specific dataset

Following previous sections, you can build a synthetic CoT dataset for fine-tuning. However, if you already have a domain-specific dataset prepared, please proceed to register and utilize your dataset for fine-tuning by following the steps outlined in this section.

**Step 1.** Organize your data in a json file and put your data in data folder. LLaMA-Factory supports dataset in alpaca or sharegpt format.

The dataset in alpaca format should follow the below format:

```

[
  {
    "instruction": "user instruction (required)",
    "input": "user input (optional)",
    "output": "model response (required)",
    "system": "system prompt (optional)",
    "history": [
      ["user instruction in the first round (optional)",

```

```

        "model response in the first round (optional)"],
        ["user instruction in the second round (optional)",
         "model response in the second round (optional)"]
    ]
}
]

```

The dataset in sharegpt format should follow the below format:

```

[
  {
    "conversations": [
      {
        "from": "human",
        "value": "user instruction"
      },
      {
        "from": "gpt",
        "value": "model response"
      }
    ],
    "system": "system prompt (optional)",
    "tools": "tool description (optional)"
  }
]

```

**Step 2.** Register your domain-specific dataset definition LLAMA-Factory dataset\_info.json

```

cd LLaMA-Factory/data
vim dataset_info.json

```

For alpaca format dataset, the columns in dataset\_info.json should be:

```

...
"dataset_name": {
  "file_name": "dataset_name.json",
  "columns": {
    "prompt": "instruction",
    "query": "input",
    "response": "output",
    "system": "system",
    "history": "history"
  }
}
...

```

For sharegpt format dataset, the columns in dataset\_info.json should be:

```

...
"dataset_name": {
  "file_name": "dataset_name.json",
  "formatting": "sharegpt",
  "columns": {
    "messages": "conversations",
    "system": "system",
    "tools": "tools"
  },
  "tags": {
    "role_tag": "from",
    "content_tag": "value",
    "user_tag": "user",
    "assistant_tag": "assistant"
  }
}
}
...

```

### 3. Data Distillation: LoRA fine-tune from base model

The base model is usually pre-trained without long CoT.

In this section, we will fine-tune the base model with CoT dataset, using the following training script:

```

export CUDA_VISIBLE_DEVICES=0
MODEL_ID="microsoft/Phi-3-mini-4k-instruct"
EXP_NAME="Phi-3-mini-4k-instruct-r1-distill-finetuned"
DATASET_NAME="deepseek-r1-distill-sample"

cd LLaMA-Factory/
torchrun --standalone --nnodes=1 --nproc-per-node=1 src/train.py \
  --stage sft \
  --do_train \
  --use_fast_tokenizer \
  --new_special_tokens "<think>,</think>" \
  --resize_vocab \
  --flash_attn auto \
  --model_name_or_path ${MODEL_ID} \
  --dataset ${DATASET_NAME} \
  --template phi \
  --finetuning_type lora \
  --lora_rank 8 --lora_alpha 16 \
  --lora_target q_proj,v_proj,k_proj,o_proj \
  --additional_target lm_head,embed_tokens \
  --output_dir ./output/${EXP_NAME} \
  --overwrite_cache \

```

```

--overwrite_output_dir \
--warmup_steps 100 \
--weight_decay 0.1 \
--per_device_train_batch_size 4 \
--gradient_accumulation_steps 4 \
--ddp_timeout 9000 \
--learning_rate 5e-6 \
--lr_scheduler_type cosine \
--logging_steps 1 \
--cutoff_len 4096 \
--save_steps 1000 \
--plot_loss \
--num_train_epochs 3 \
--bf16

```

**Note:**

- If you have trouble with downloading models and datasets from Hugging Face, you can use ModelScope by setting environment variable: `export USE_MODELSCOPE_HUB=1`
- If you would like to fine-tune models on Intel® Arc™ A770 Graphics, please refer to [sec 6. Fine-tune Model on Intel® Arc™ A770 Graphics](#)

## 4. Merge LoRA Adapter with Base Model

After the training is done, you can merge the LoRA adapter into base model. Please convert the model as follows.

- Prepare export configurations in `merge_lora.yaml`:

```

### model
model_name_or_path: microsoft/Phi-3-mini-4k-instruct
adapter_name_or_path: ./output/Phi-3-mini-4k-instruct-r1-distill-finetuned
template: phi
trust_remote_code: true
new_special_tokens: <think>,</think>
resize_vocab: true

### export
export_dir: ./output/Phi-3-mini-4k-instruct-r1-distill-finetuned/merged
export_size: 5
export_device: cpu
export_legacy_format: false

```

- Export final model:



```
llamafactory-cli export merge_lora.yaml
```

Now, you can get the final model at: `./output/Phi-3-mini-4k-instruct-r1-distill-finetuned/merged`

## 5. Test with Fine-tuned Model

In this best practice, we demonstrate through 728 reasoning questions (consistent with the R1 technical report). The evaluation data specifically includes:

- **MATH-500**: A set of challenging high school mathematics competition problem datasets, covering seven subjects (such as elementary algebra, algebra, number theory) with a total of 500 questions.
- **GPQA-Diamond**: This dataset contains master’s level multiple-choice questions in the subfields of physics, chemistry, and biology, with a total of 198 questions.
- **AIME-2024**: A dataset for the American Invitational Mathematics Competition, containing 30 mathematics questions.

### 5.1 Evaluation tool

We recommend using [ModelScope/EvalScope](#) to evaluate performance on these three benchmarks. Please follow the [Best Practice for deepseek\\_r1\\_distill](#) to set up the evaluation environment.

### 5.2 Quality Analysis

#### Description:

- Base model: [microsoft/Phi-3-mini-4k-instruct](#)
- Question: What is the least positive integer multiple of 30 that can be written with only the digits 0 and 2?
- Expected answer: 2220

In this section, we use the model fine-tuned on collected Chain-of-Thought (CoT) data following [2.1 Adopt MagPie Synthetic Dataset](#) to perform the quality analysis before and after the distillation process.

The training data was filtered based on response length ( $<2048$ ) to create a subset of the [Magpie-Align/Magpie-Reasoning-V2-250K-CoT-Deepseek-R1-Llama-70B](#) dataset, resulting in a 50k dataset for LoRA fine-tuning.

**Before fine-tuning:** The answer is straightforward, but the model produced an incorrect output for this question.

```

We want to find the smallest positive integer multiple of 30 that can be written using only the digits 0 and 2.
First, let's consider the prime factorization of 30:  $30 = 2 \times 3 \times 5$ .
To find the least positive integer multiple of 30 that can be written using only the digits 0 and 2, we need to find the smallest number that has 2 as a factor and is divisible by 3 and 5.
Since the number must be divisible by 5, it must end in 0.
Therefore, the number must be of the form  $2^n \times 5^m \times 10^k$ , where  $n$  is a nonnegative integer.
To minimize the number, we want to minimize  $n$ .
The smallest value of  $n$  that satisfies the condition is  $n=1$ , since  $2^0 \times 5^1$  is not a multiple of 2.
Therefore, the least positive integer multiple of 30 that can be written using only the digits 0 and 2 is  $2^1 \times 5^1 \times 10^1 = \boxed{30}$ .
The answer is: 30

```

Figure 1: Phi-3-mini-4k-instruct before distillation

**After fine-tuning:** The model demonstrates reasoning skills similar to DeepSeek-R1’s, following a step-by-step thinking process to arrive at the correct output for this question.

```

<think>
First, I recognize that the number must be a multiple of 30, which means it must be divisible by both 3 and 10.
For a number to be divisible by 10, it must end in 0. This means the number must be of the form 2220, 22200, 222000, and so on.
Next, I need to ensure that the number is also divisible by 3. The divisibility rule for 3 states that the sum of the digits must be divisible by 3. I'll start by adding the digits of the first few numbers in the form 2220:
- For 2220, the sum of the digits is  $2 + 2 + 2 + 0 = 6$ , which is divisible by 3.
- For 22200, the sum is  $2 + 2 + 2 + 0 + 0 = 6$ , which is also divisible by 3.
Since the sum of the digits remains 6 regardless of how many zeros are added, the number will always be divisible by 3.
Therefore, the smallest such number is 2220.
</think>
To find the least positive integer multiple of  $\boxed{30}$  that can be written using only the digits  $\boxed{0}$  and  $\boxed{2}$ , we can follow these steps:
1. Understand the Requirements:
- The number must be a multiple of  $\boxed{30}$ .
- It can only contain the digits  $\boxed{0}$  and  $\boxed{2}$ .
2. Break Down the Number  $\boxed{30}$ :
-  $\boxed{30} = 2 \times 3 \times 5$ .
- To be a multiple of  $\boxed{30}$ , the number must be divisible by  $\boxed{2}$ ,  $\boxed{3}$ , and  $\boxed{5}$ .
3. Divisibility by  $\boxed{5}$ :
- A number is divisible by  $\boxed{5}$  if it ends in  $\boxed{0}$ .
- Therefore, the number must end with a  $\boxed{0}$ .
4. Divisibility by  $\boxed{3}$ :
- A number is divisible by  $\boxed{3}$  if the sum of its digits is divisible by  $\boxed{3}$ .
- Since the number can only contain  $\boxed{0}$  and  $\boxed{2}$ , the possible sums of digits are  $\boxed{0}$ ,  $\boxed{2}$ ,  $\boxed{4}$ ,  $\boxed{6}$ ,  $\boxed{8}$ ,  $\boxed{10}$ ,  $\boxed{12}$ ,  $\boxed{14}$ ,  $\boxed{16}$ , and so on.
- The smallest sum that is divisible by  $\boxed{3}$  is  $\boxed{6}$ .
5. Construct the Number:
- To have a sum of digits equal to  $\boxed{6}$ , we can use two  $\boxed{2}$ 's and two  $\boxed{0}$ 's:  $\boxed{2200}$ .
- This number is divisible by  $\boxed{3}$  ( $\boxed{2} + \boxed{2} + \boxed{0} + \boxed{0} = \boxed{4}$ ), not  $\boxed{3}$ . So it does not meet the requirement.
- Let's try adding another  $\boxed{2}$ :  $\boxed{2220}$ .
- The sum of the digits is  $\boxed{2} + \boxed{2} + \boxed{2} + \boxed{0} = \boxed{6}$ , which is divisible by  $\boxed{3}$ .
- Additionally,  $\boxed{2220}$  ends with a  $\boxed{0}$ , satisfying the divisibility by  $\boxed{5}$ .
6. Conclusion:
- The number  $\boxed{2220}$  is the smallest number that meets all the criteria.
\boxed{2220}

```

Figure 2: Phi-3-mini-4k-instruct after distillation

## 5.3 Benchmarking

Based on the evaluation tool mentioned in [sec 5.1 Evaluation tool](#), we run the benchmarking evaluation on the raw base model and some CoT distilled models.

**Evaluation Metrics:** pass@1 (averaged over 5 runs)

**Generation configs:**

- max\_tokens: 4096
- temperature: 0.6
- top\_p: 0.95

**Results:**

ID	Model	samples	math-500	gpqa-diamond	aime24
1	microsoft/Phi-3-mini-4k-instruct	-	34.80%	31.82%	0.00%
2	Phi-3-mini-4k-instruct-distill-r1024	13568 (response<1024)	46.92%	28.08%	2.00%
3	Phi-3-mini-4k-instruct-distill-r2048	54824 (response<2048)	46.32%	29.80%	5.33%

#### Experiment description:

- **Experiment ID #1:** Evaluate the raw model: microsoft/Phi-3-mini-4k-instruct on the given three reasoning benchmarks.
- **Experiment ID #2:** The model was fine-tuned on the collected CoT data using the training script outlined in [sec 3. Data Distillation: LoRA fine-tune from base model](#). The training dataset consists of 15,076 samples, collected following the methodology described in [2.1 Adopt MagPie Synthetic Dataset](#). These samples were filtered based on response length (less than 1024 tokens) to create a refined subset.
- **Experiment ID #3:** The experiment settings are identical to those of ID#2, except that the training data samples were filtered based on longer response length (less than 2048 tokens) to create a refined subset. This adjustment was made to better investigate the model’s capacity.

#### Observations:

- After data distillation fine-tuning, the base model gains reasoning capabilities similar to DeepSeek-R1 and demonstrates improved performance on math-related benchmarks, though it experiences a slight decline in factual QA tasks (gpqa-diamond).
- The CoT dataset ([Magpie-Align/Magpie-Reasoning-V2-250K-CoT-Deepseek-R1-Llama-70B](#)) we used here primarily consists of math-related tasks, such as data analysis, information-seeking, and reasoning. As a result, the accuracy improvement on math tasks is significant.

## 6. Fine-tune Model on Intel® Arc™ A770 Graphics

If you want to fine-tune model on Intel® Arc™ A770 Graphics instead of [sec 3. Data Distillation: LoRA finetune from base model](#), please follow the guides below.

### 6.1 Install LLAMA-Factory with Pytorch-Extension-for-Intel

**Pre-requisites** You should have dGPU driver installed on your system. For installation on Intel Arc B-Series GPU (such as B580), please refer to this [guide](#).

**Step 1.** Follow the instructions of [Pytorch Extension for Intel](#) to install necessary packages:

```
python3 -m venv ~/python-env/llamafactory-xpu
source ~/python-env/llamafactory-xpu/bin/activate

pip install torch==2.5.1+cxx11.abi torchvision==0.20.1+cxx11.abi \
    torchaudio==2.5.1+cxx11.abi intel-extension-for-pytorch==2.5.10+xpu \
    onecccl_bind_pt==2.5.0+xpu \
    --extra-index-url https://pytorch-extension.intel.com/release-whl/stable/xpu/us/

git clone --depth 1 https://github.com/hiyouga/LLaMA-Factory.git
cd LLaMA-Factory
pip install -e ".[metrics]"
```

**Step 2.** Install necessary packages

```
pip install "accelerate<=1.2.1,>=0.34.0"
pip install "transformers<=4.49.0"
```

## 6.2 Dataset Preparation.

Please refer to [sec 2. Dataset Preparation](#)

## 6.3 Set configurations to enable training on XPU plugin.

**Step 1.** Set Intel® OneAPI-2025.0 environment.

Download installer from [here](#).

```
sudo -E sh ./intel-oneapi-base-toolkit-2025.0.1.46_offline.sh \
    -a --install-dir /opt/intel/oneapi-2025.0 --silent \
    --cli --eula accept

source /opt/intel/oneapi-2025.0/setvars.sh
```

**Step 2.** Use the accelerate command to enable training on XPU plugin.

First run the following command to generate a configuration file after answering a series of questions according to your needs:

```
accelerate config
```

Here are two examples for configurations:

- Single GPU:

```

.....
In which compute environment are you running?
This machine
.....
Which type of machine are you using?
No distributed training
Do you want to run your training on CPU only (even if a GPU / Apple Silicon / Ascend NPU device is available)? [yes/NO]:NO
Do you want to use XPU plugin to speed up training on XPU? [yes/NO]:yes
Do you wish to optimize your script with torch dynamo?[yes/NO]:NO
Do you want to use DeepSpeed? [yes/NO]: NO
What XPU(s) (by id) should be used for training on this machine as a comma-seperated list? [all]:all
Would you like to enable numa efficiency? (Currently only supported on NVIDIA hardware). [yes/NO]:
.....
Do you wish to use mixed precision?
bf16
accelerate configuration saved at /home/user/.cache/huggingface/accelerate/default_config.yaml

```

Figure 3: single gpu accelerate config

- Multiple GPUs with FullShardedDataParallel(FSDP):

```

.....
In which compute environment are you running?
This machine
.....
Which type of machine are you using?
Multi-XPU
How many different machines will you use (use more than 1 for multi-node training)? [1]: 1
Should distributed operations be checked while running for errors? This can avoid timeout issues but will be slower. [yes/NO]: NO
Do you want to use XPU plugin to speed up training on XPU? [yes/NO]:yes
Do you wish to optimize your script with torch dynamo?[yes/NO]:NO
Do you want to use DeepSpeed? [yes/NO]: NO
Do you want to use FullyShardedDataParallel? [yes/NO]: yes
.....
What should be your sharding strategy?
FULL_SHARD
Do you want to offload parameters and gradients to CPU? [yes/NO]: NO
.....
What should be your auto wrap policy?
TRANSFORMER_BASED_WRAP
Do you want to use the model's '_no_split_modules' to wrap. Only applicable for @ Transformers [yes/NO]: yes
.....
What should be your FSDP's backward prefetch policy?
BACKWARD_PREFE
.....
What should be your FSDP's state dict type?
SHARDED_STATE_DICT
Do you want to enable FSDP's forward prefetch policy? [yes/NO]: yes
Do you want to enable FSDP's use_orig_params feature? [YES/no]: YES
Do you want to enable CPU RAM efficient model loading? Only applicable for @ Transformers models. [YES/no]: YES
Do you want to enable FSDP activation checkpointing? [yes/NO]: yes
How many GPU(s) should be used for distributed training? [1]:2
.....
Do you wish to use mixed precision?
bf16
accelerate configuration saved at /home/user/.cache/huggingface/accelerate/default_config.yaml

```

Figure 4: multiple gpus accelerate config

#### 6.4 Run with train script as follows:

```

export ONEAPI_DEVICE_SELECTOR="level_zero:0"
MODEL_ID="microsoft/Phi-3-mini-4k-instruct"
EXP_NAME="Phi-3-mini-4k-instruct-r1-distill-finetuned"
DATASET_NAME="deepseek-r1-distill-sample"

cd LLaMA-Factory/
accelerate launch src/train.py \
    --stage sft \
    --do_train \
    --use_fast_tokenizer \

```

```

--new_special_tokens "<think>,</think>" \
--resize_vocab \
--flash_attn auto \
--model_name_or_path ${MODEL_ID} \
--dataset ${DATASET_NAME} \
--template phi \
--finetuning_type lora \
--lora_rank 8 --lora_alpha 16 \
--lora_target q_proj,v_proj,k_proj,o_proj \
--additional_target lm_head,embed_tokens \
--output_dir ./output/${EXP_NAME} \
--overwrite_cache \
--overwrite_output_dir \
--warmup_steps 100 \
--weight_decay 0.1 \
--per_device_train_batch_size 1 \
--gradient_accumulation_steps 4 \
--ddp_timeout 9000 \
--learning_rate 5e-6 \
--lr_scheduler_type cosine \
--logging_steps 1 \
--save_steps 1000 \
--plot_loss \
--num_train_epochs 3 \
--torch_empty_cache_steps 10 \
--bf16

```

#### Note:

- You can use `export ONEAPI_DEVICE_SELECTOR=level_zero:[gpu_id]` to select device before excuting your command.
- If you are not sure about the `gpu_id`, please use the `sycl-ls` command to check:

```

source /opt/intel/oneapi-2025.0/setvars.sh
sycl-ls

```

Output example:

```

[level_zero:gpu][level_zero:0] Intel(R) oneAPI Unified Runtime over Level-Zero, Intel(R) Arc(TM) A770 Graphics 12.55.8 [1.3.29377]
[level_zero:gpu][level_zero:1] Intel(R) oneAPI Unified Runtime over Level-Zero, Intel(R) Arc(TM) A770 Graphics 12.55.8 [1.3.29377]
[opencl:cpu][opencl:0] Intel(R) OpenCL, Intel(R) Xeon(R) Gold 6430 OpenCL 3.0 (Build 0) [2024.18.12.0.05.160000]
[opencl:gpu][opencl:1] Intel(R) OpenCL Graphics, Intel(R) Arc(TM) A770 Graphics OpenCL 3.0 NEO [24.17.29377.6]
[opencl:gpu][opencl:2] Intel(R) OpenCL Graphics, Intel(R) Arc(TM) A770 Graphics OpenCL 3.0 NEO [24.17.29377.6]

```

Figure 5: Output example for `sycl-ls`

- To use multiple GPUs, specify the devices by separating them with a

comma. For example: `export ONEAPI_DEVICE_SELECTOR="level_zero:0,level_zero:1"`

### **6.5 Merge LoRA adapter with base model**

Please refer to [sec 4. Merge LoRA adapter with base model](#)

### **6.6 Test with fine-tuned model**

Please refer to [sec 5. Test with fine-tuned model](#)

## **7. Reference**

[1] <https://medium.com/@prabhudev.guntur/how-to-distill-deepseek-r1-a-comprehensive-guide-c8ba04e2c28c>

[2] [https://qwen.readthedocs.io/en/latest/training/SFT/llama\\_factory.html](https://qwen.readthedocs.io/en/latest/training/SFT/llama_factory.html)