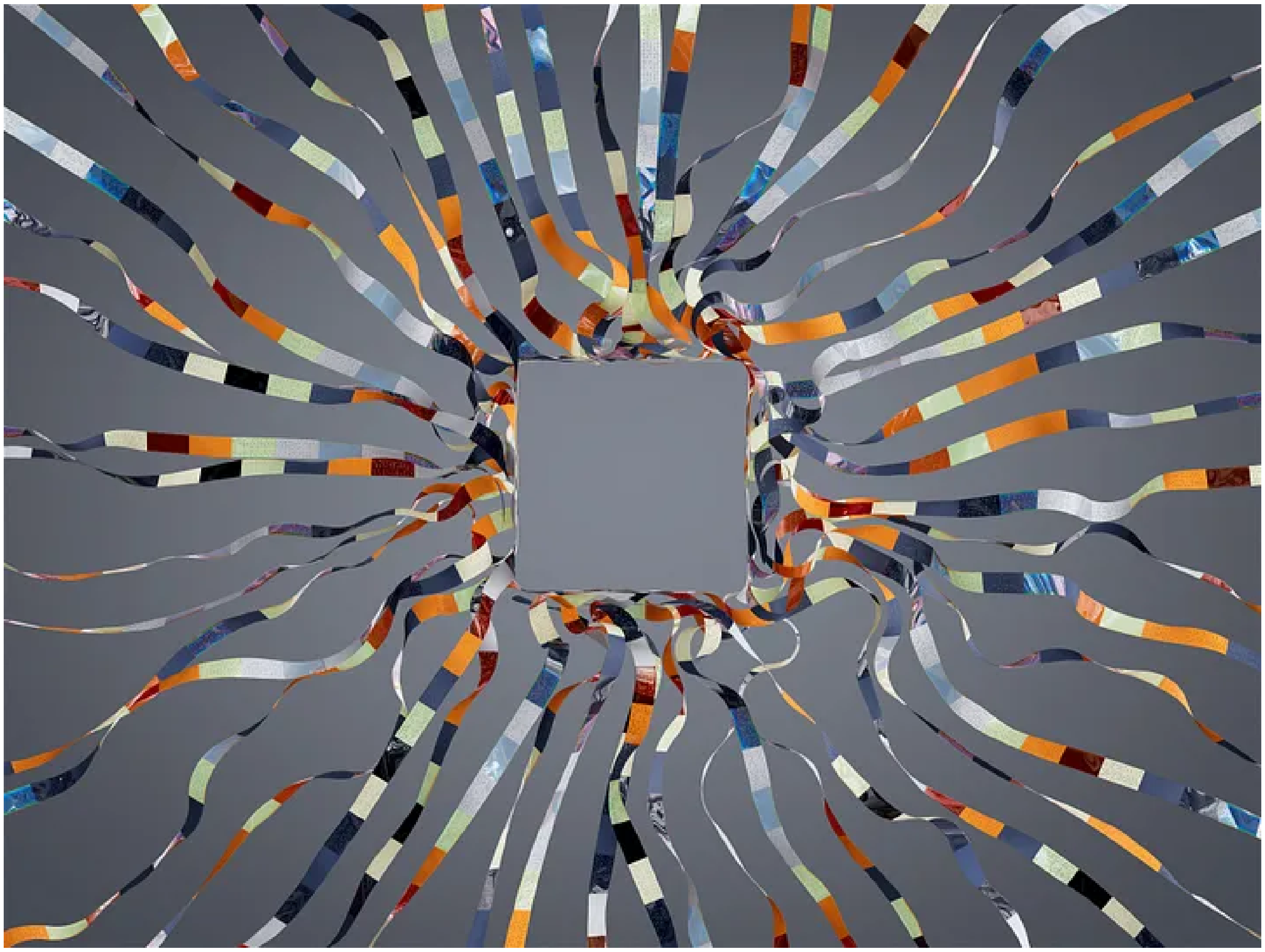


Photo by Google DeepMind on Unsplash

Supervised Fine-Tuning and Direct Preference Optimization on Intel Gaudi2

Demonstrating a Top-Ranked 7B Chat Model on the LLM Leaderboard

Intel(R) Neural Compressor · Follow
Published in Intel Analytics Software · 4 min read · Nov 14, 2023

89

Kaokao Lv, Wenxin Zhang, and Haihao Shen, Intel Corporation

Intel Extension For Transformers provides robust support for cross-platform training and inference with a particular emphasis on Intel Gaudi2 accelerators, which are designed to expedite large language model (LLM) training and inference. In this article, we will provide a comprehensive walkthrough of the process to apply supervised fine-tuning and direct preference optimization (DPO) on Intel Gaudi2. We also present benchmarks that achieve comparable or even better results compared to other open-source LLMs of similar size published on the open LLM leaderboard.

Model: https://huggingface.co/Intel/neural-chat-7b-v3

Dataset: https://huggingface.co/datasets/Open-Orca/SlimOrca

Preference Dataset: https://huggingface.co/datasets/Intel/orca_dpo_pairs

Codebase: https://github.com/intel/intel-extension-for-transformers

Hardware

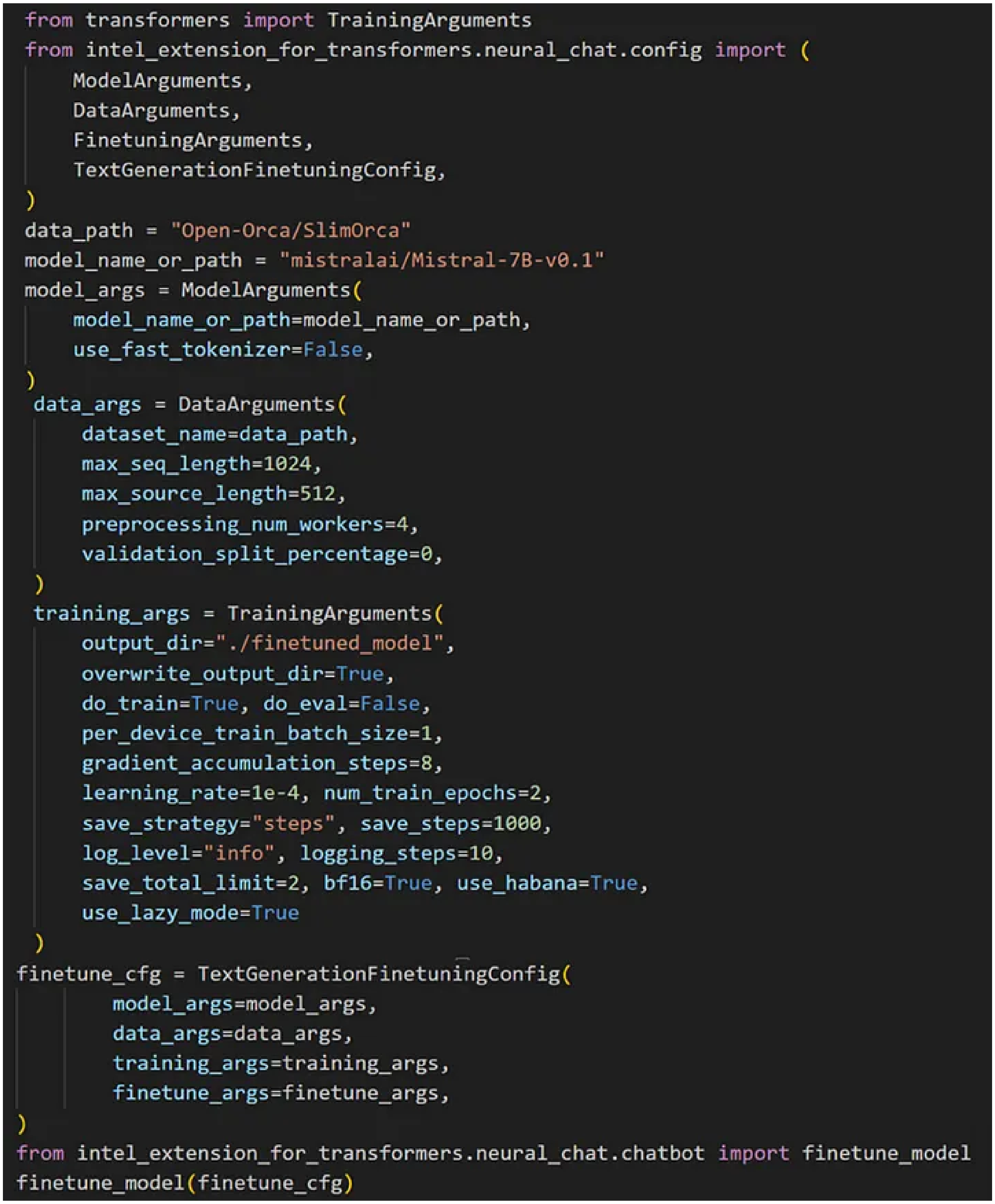
The Intel Gaudi2 AI accelerator was developed by Habana Labs for state-of-the-art deep learning training and inference. It has 96 GB of integrated memory and is available in servers containing eight Gaudi2 mezzanine cards via the Intel Developer Cloud or for on-premises infrastructure from Supermicro and IEI.

Training

To start the supervised fine-tuning, we select the latest mistralai/Mistral-7B-v0.1 Hugging Face as the base LLM model for two reasons: the pretrained model has strong benchmark results and it is commercially friendly under Apache 2.0.

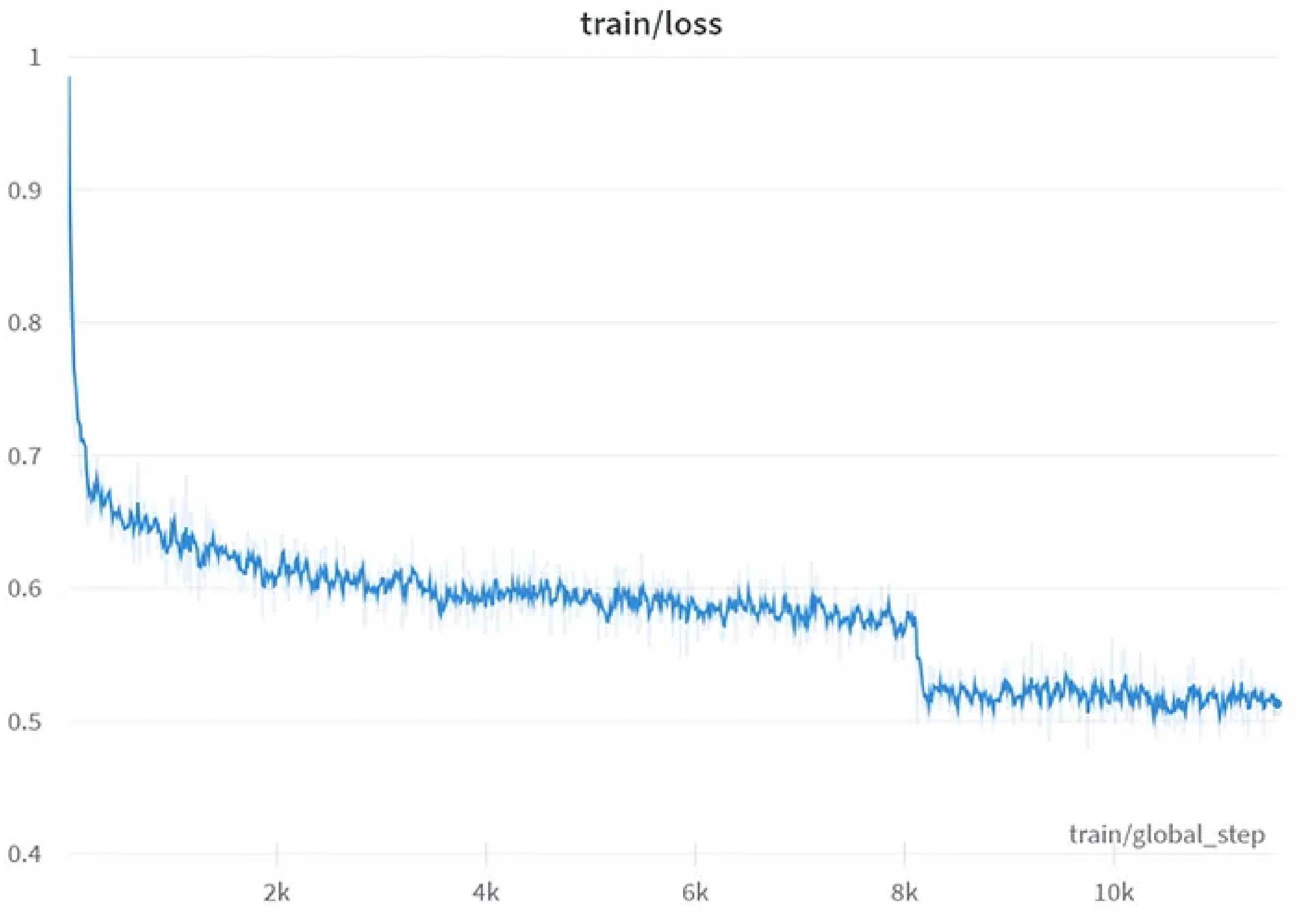
Supervised Fine-Tuning

We select the latest high-quality instruction dataset Open-Orca/SlimOrca Datasets at Hugging Face and leverage the fine-tuning pipeline provided in Intel Extension for Transformers to perform training with DeepSpeed ZeRO-2. The fine-tuning code and training loss curve are shown below:



```python
from transformers import TrainingArguments
from intel_extension_for_transformers.neural_chat.config import (
    ModelArguments,
    DataArguments,
    FinetuningArguments,
    TextGenerationFinetuningConfig,
)
data_path = "Open-Orca/SlimOrca"
model_name_or_path = "mistralai/Mistral-7B-v0.1"
model_args = ModelArguments(
    model_name_or_path=model_name_or_path,
    use_fast_tokenizer=False,
)
 data_args = DataArguments(
     dataset_name=data_path,
     max_seq_length=1024,
     max_source_length=512,
     preprocessing_num_workers=4,
     validation_split_percentage=0,
 )
 training_args = TrainingArguments(
     output_dir="./finetuned_model",
     overwrite_output_dir=True,
     do_train=True, do_eval=False,
     per_device_train_batch_size=1,
     gradient_accumulation_steps=8,
     learning_rate=1e-4, num_train_epochs=2,
     save_strategy="steps", save_steps=1000,
     log_level="info", logging_steps=10,
     save_total_limit=2, bf16=True, use_habana=True,
     use_lazy_mode=True
 )
finetune_cfg = TextGenerationFinetuningConfig(
        model_args=model_args,
        data_args=data_args,
        training_args=training_args,
        finetune_args=finetune_args,
)
from intel_extension_for_transformers.neural_chat.chatbot import finetune_model
finetune_model(finetune_cfg)
```

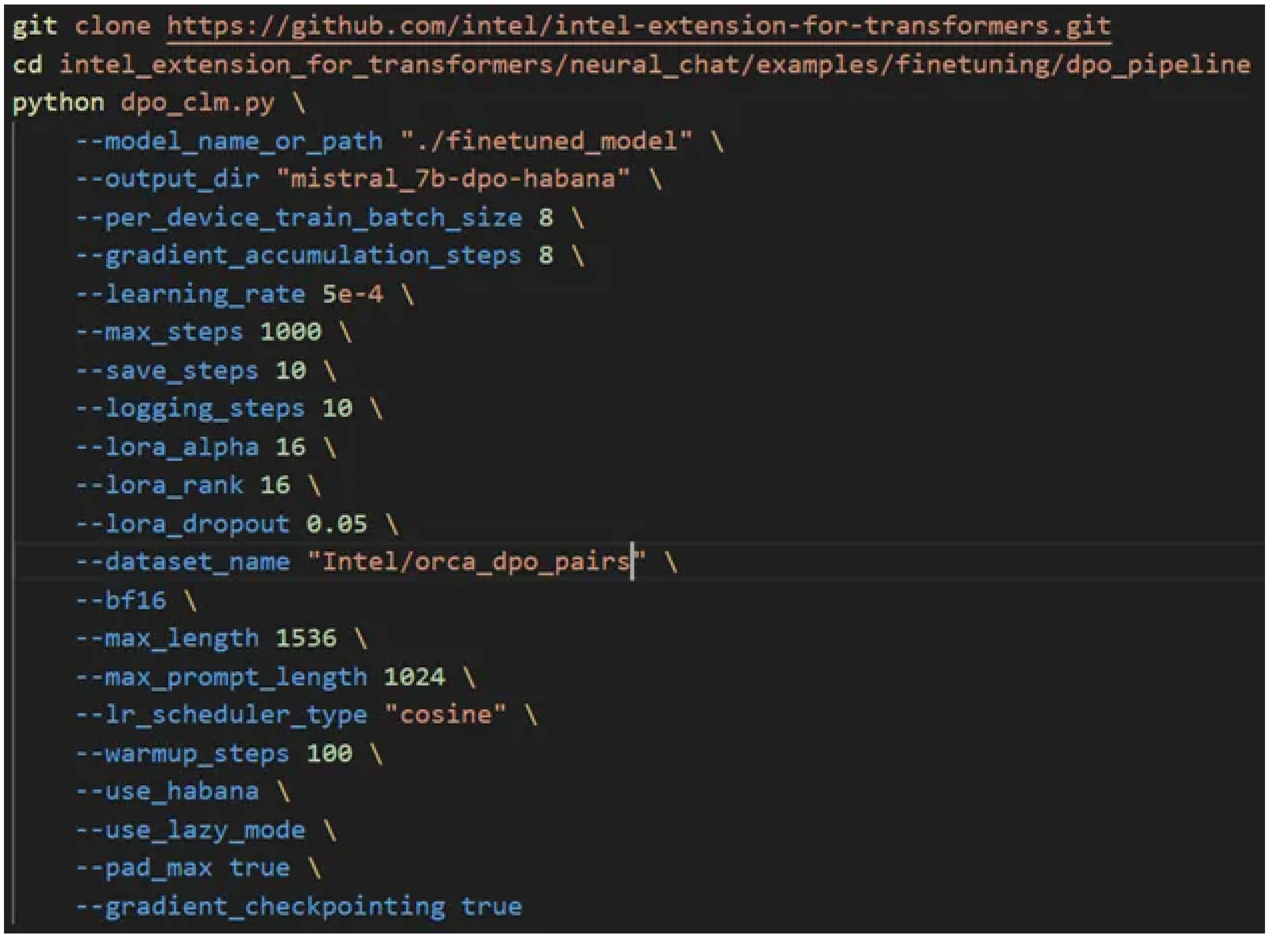
Fine-Tuning Code



Training Loss Curve

Direct Preference Optimization

We apply the DPO algorithm, which is stable and computationally lightweight, to better align with human preferences. DPO derives the probability of human preference data for the optimal policy to replace the reward model that reinforcement learning from human feedback needs and formulates a maximum likelihood objective for a parameterized policy. The preference dataset contains 12k examples selected from the Orca style dataset Open-Orca/OpenOrca. Its completions generated from GPT-4 or GPT-3.5 are regarded as the chosen response. We use the llama-2–13b-chat model to generate corresponding completions automatically because maybe better rejected responses are also better for alignment. Refer to Intel/orca_dpo_pairs and DPO example for more details on the dataset and DPO training code. The launch script is shown below:

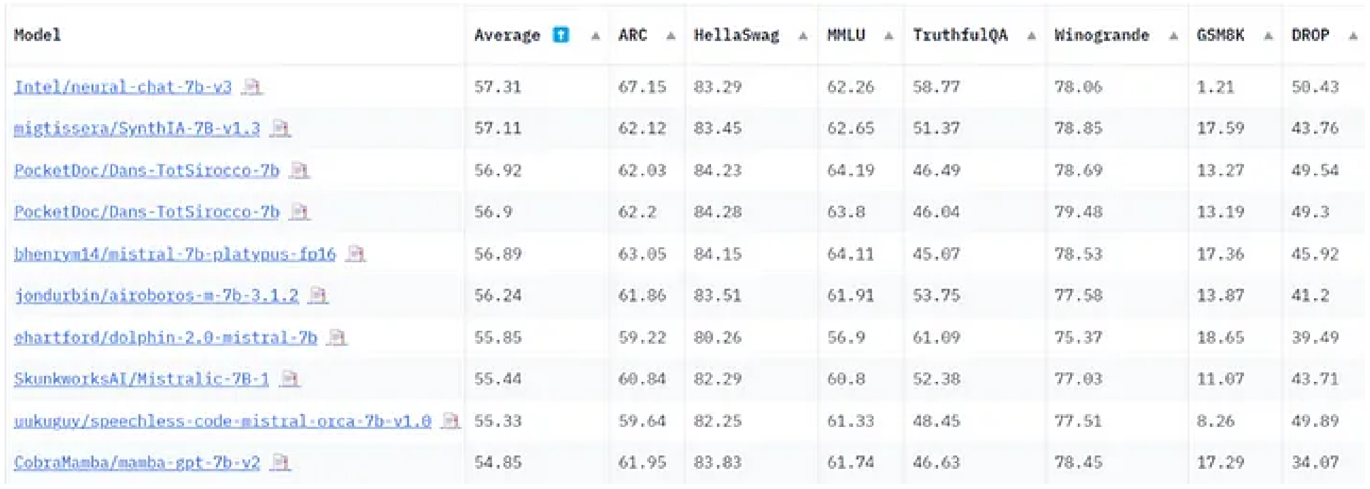


```
git clone https://github.com/intel/intel-extension-for-transformers.git
cd intel_extension_for_transformers/neural_chat/examples/finetuning/dpo_pipeline
python dpo_clm.py \
    --model_name_or_path "./finetuned_model" \
    --output_dir "mistral_7b-dpo-habana" \
    --per_device_train_batch_size 8 \
    --gradient_accumulation_steps 8 \
    --learning_rate 5e-4 \
    --max_steps 1000 \
    --save_steps 10 \
    --logging_steps 10 \
    --lora_alpha 16 \
    --lora_rank 16 \
    --lora_dropout 0.05 \
    --dataset_name "Intel/orca_dpo_pairs" \
    --bf16 \
    --max_length 1536 \
    --max_prompt_length 1024 \
    --lr_scheduler_type "cosine" \
    --warmup_steps 100 \
    --use_habana \
    --use_lazy_mode \
    --pad_max true \
    --gradient_checkpointing true
```

Launch Script

Benchmark Results

We submitted our model to the open_llm_leaderboard, which uses the Eleuther AI Language Model Evaluation Harness, a unified framework to test generative language models on a large number of different evaluation tasks. Our model performed quite well:

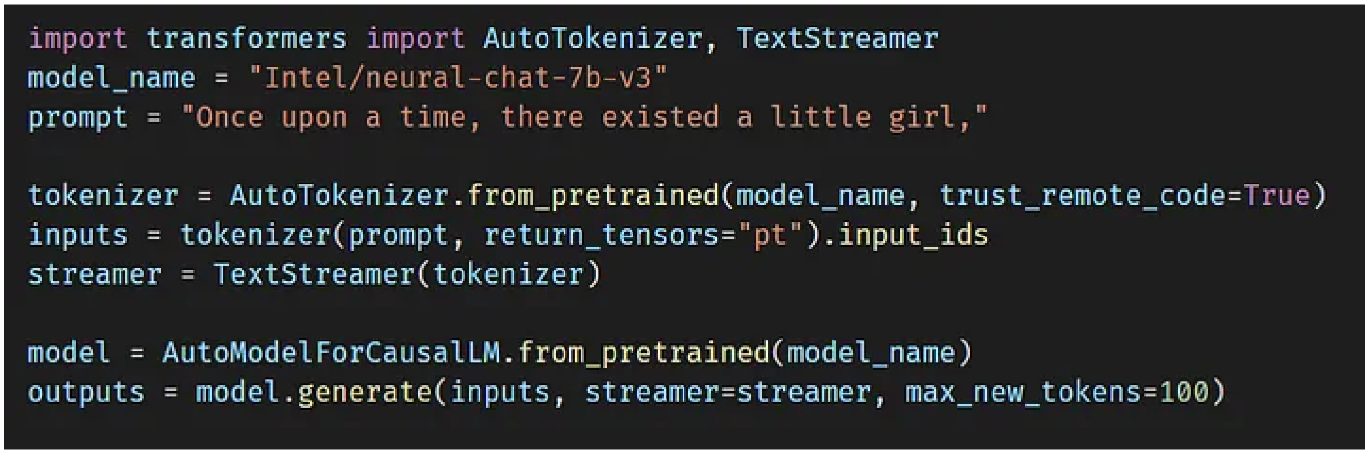


Model	Average	ARC	HellaSwag	MMLU	TruthfulQA	Winogrande	GSM8K	DROP
Intel/neural-chat-7b-v3	57.31	67.15	83.29	62.26	58.77	78.06	1.21	50.43
migtissera/SynthIA-7B-v1.3	57.11	62.12	83.45	62.65	51.37	78.85	17.59	43.76
PocketDoc/Dans-TotSirocco-7b	56.92	62.03	84.23	64.19	46.49	78.69	13.27	49.54
PocketDoc/Dans-TotSirocco-7b	56.9	62.2	84.28	63.8	46.04	79.48	13.19	49.3
bhenrym14/mistral-7b-platypus-fp16	56.89	63.05	84.15	64.11	45.07	78.53	17.36	45.92
jondurbin/airoboros-m-7b-3.1.2	56.24	61.86	83.51	61.91	53.75	77.58	13.87	41.2
ehartford/dolphin-2.0-mistral-7b	55.85	59.22	80.26	56.9	61.09	75.37	18.65	39.49
SkunkworksAI/Mistralic-7B-1	55.44	60.84	82.29	60.8	52.38	77.03	11.07	43.71
uukuguy/speechless-code-mistral-orca-7b-v1.0	55.33	59.64	82.25	61.33	48.45	77.51	8.26	49.89
CobraMamba/mamba-gpt-7b-v2	54.85	61.95	83.83	61.74	46.63	78.45	17.29	34.07

NeuralChat-7b-v3 Ranked First on the 7B-sized LLM Leaderboard (November 13th, 2023)

Inference

NeuralChat is fully compatible with Transformers. Use the same launcher code with the model's name "Intel/neural-chat-7b-v3" to perform the inference in FP32:



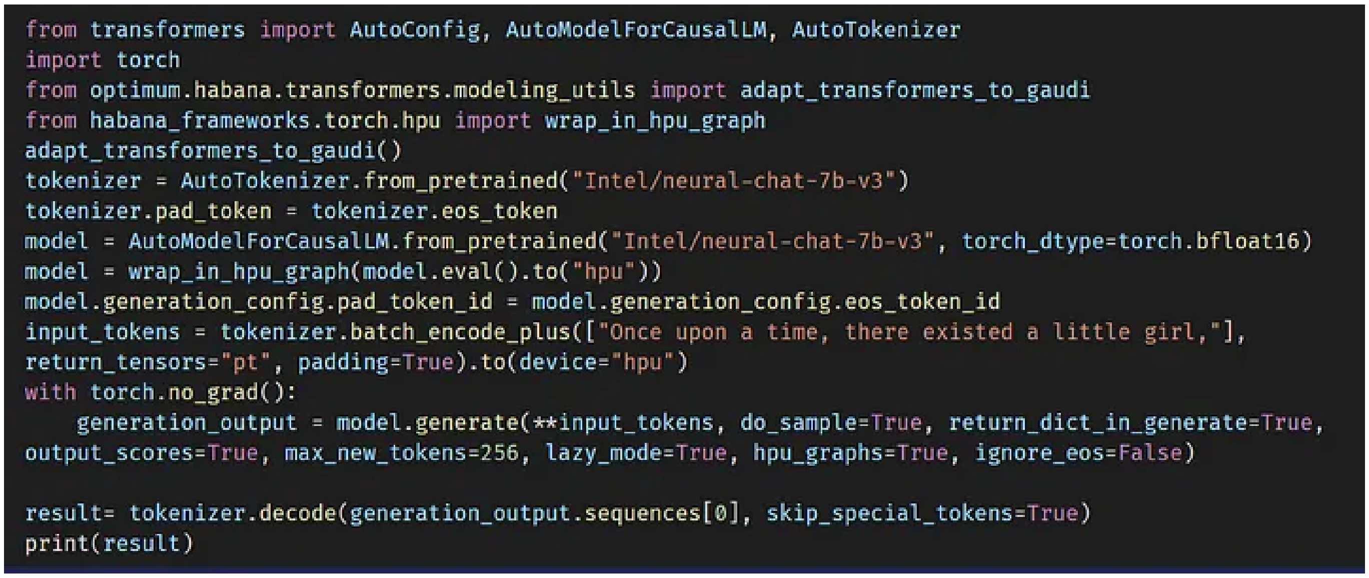
```
import transformers import AutoTokenizer, TextStreamer
model_name = "Intel/neural-chat-7b-v3"
prompt = "Once upon a time, there existed a little girl,"

tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
inputs = tokenizer(prompt, return_tensors="pt").input_ids
streamer = TextStreamer(tokenizer)

model = AutoModelForCausalLM.from_pretrained(model_name)
outputs = model.generate(inputs, streamer=streamer, max_new_tokens=100)
```

FP32 Inference

Follow these instructions to enable BF16 inference using Optimum-Habana to improve inference performance:



```
from transformers import AutoConfig, AutoModelForCausalLM, AutoTokenizer
import torch
from optimum.habana.transformers.modeling_utils import adapt_transformers_to_gaudi
from habana_frameworks.torch.hpu import wrap_in_hpu_graph
adapt_transformers_to_gaudi()
tokenizer = AutoTokenizer.from_pretrained("Intel/neural-chat-7b-v3")
tokenizer.pad_token = tokenizer.eos_token
model = AutoModelForCausalLM.from_pretrained("Intel/neural-chat-7b-v3", torch_dtype=torch.bfloat16)
model = wrap_in_hpu_graph(model.eval().to("hpu"))
model.generation_config.pad_token_id = model.generation_config.eos_token_id
input_tokens = tokenizer.batch_encode_plus(["Once upon a time, there existed a little girl,"],
return_tensors="pt", padding=True).to(device="hpu")
with torch.no_grad():
    generation_output = model.generate(**input_tokens, do_sample=True, return_dict_in_generate=True,
output_scores=True, max_new_tokens=256, lazy_mode=True, hpu_graphs=True, ignore_eos=False)

result= tokenizer.decode(generation_output.sequences[0], skip_special_tokens=True)
print(result)
```

BF16 Inference

Ethics Statement

We created NeuralChat and released the code, model, and dataset to the community for commercial. We strived to address the risks of hallucination, toxicity, and other potential ethics issues during fine-tuning. Like other LLMs, NeuralChat is not free from such issues. We also carefully performed the low-precision quantization for inference acceleration and ensured the quantized model didn't deviate too far from the baseline. We are hoping to collaborate with the community to improve these issues to make AI beneficial for everyone everywhere.

Concluding Remarks

We are excited to release NeuralChat, a commercially friendly 7B chat model, to the LLM community. This model shows higher performance than the original base model in typical generative language model benchmarks. We expect NeuralChat to help push the limits of 7B chat model deployment and motivate more researchers and developers to open their LLMs.

We encourage you to have a try and submit your fine-tuned model to the LLM leaderboard. Please give a star ⭐ to Intel Extension for Transformers repository if you find it useful. You are also welcome to create pull requests or submit issues or questions to the repository.

Llm · Intel Neural Compressor · Low Precision · Llm Leaderboard · Transformers

Written by Intel(R) Neural Compressor

Follow

289 Followers · Writer for Intel Analytics Software

LLM quantization (https://github.com/intel/neural-compressor) and highly-efficient LLM inference (https://github.com/intel/intel-extension-for-transformers)