Fine-Tuning the Falcon 7-Billion Parameter Model with Hugging Face and oneAPI

Optimizing Large Language Models on Intel® Xeon® Processors with Intel® Advanced Matrix Extensions (Intel® AMX)

Get the Latest on All Things CODE

Eduardo Alvarez, *Senior AI Solutions Engineer*

Intel Corporation

Eduardo Alvarez, *Senior AI Solutions Engineer*

Intel Corporation

Open-sourcing large language models (LLMs) goes a long way toward making AI technology accessible everywhere. It's possible but unlikely that the next AI research breakthrough will come from someone without access to massively distributed clusters of accelerators. However, the story is quite different in AI application development, where there is more flexibility when selecting product development infrastructure. This makes the intersection of the availability and scalability of CPUs and the truly open-source license behind the Falcon LLM a major enabling factor for AI.

This article explores the exciting challenge of fine-tuning the state-of-the-art Falcon 7-billion language model (Falcon-7B) on Intel® Xeon® processors using the Hugging Face* Supervised Fine-tuning Trainer (SFTTrainer), Intel® Extension for PyTorch* (IPEX) with Intel® Advanced Matrix Extensions (Intel® AMX), and Auto Mixed Precision (AMP) with Bfloat16.

Environment Setup

Set up the environment as follows:

1. Install miniconda.
2. Create a conda environment: conda create -n falconft python==3.8.10
3. Install dependencies: pip install -r requirements.txt. The requirements.txt file lists the following dependencies:

    - ```
      torch==2.0.1
      transformers==4.30.1
      bitsandbytes==0.39.0
      peft==0.3.0
      accelerate==0.20.3
      datasets==2.12.0
      trl==0.4.4
      einops==0.6.1
      scipy==1.10.1
      intel_extension_for_pytorch==2.0.100
      ```

4. Activate the conda environment: conda activate falconft

Fine-Tuning for Causal Language Modeling

Causal language modeling involves predicting the next word in a sequence based on the preceding context, enabling tasks like text generation. Fine-tuning a model like Falcon-7B for a specific task involves adapting the pretrained model by providing task-specific labeled data. The model is further trained on this data, adjusting its parameters to optimize performance on the new task. Through this process, Falcon-7B gradually learns the patterns and intricacies of the specific causal task, enabling it to generate coherent and contextually appropriate text for that particular use case.

We will use a subset of the Open Assistant dataset that only contains the highest-rated paths in the conversation tree (a total of 9,846 samples). Check out this article to learn more about fine-tuning and transfer learning.

While GPUs have been the default choice for deep learning tasks, fine-tuning Falcon-7B on CPUs provides several advantages:
      ```

- **Availability:** CPUs are ubiquitous and easily accessible, making them an attractive option for researchers and practitioners who may not have access to expensive GPU clusters.
- **Cost:** CPUs are generally more cost-effective than GPUs for large-scale deployments.
- **Compatibility:** CPUs are compatible with a wide range of hardware and infrastructure, ensuring smooth integration into existing systems.

Fine-tuning Falcon-7B becomes even more efficient and effective by combining SFTTrainer with IPEX with Intel AMX and AMP with Bfloat16. SFTTrainer simplifies the fine-tuning process by providing a higher-level abstraction for complex tasks. IPEX and AMP take advantage of the latest hardware features in Intel Xeon processors. This extension introduces support for the newest optimizations and devices before they are upstreamed into open-source PyTorch*. It also supports AMP training and inference, converting parameters and operations to Bfloat16 to further accelerate Intel AMX while preserving full 32-bit accuracy where necessary.

Falcon-7B is a 7-billion parameter decoder-only model developed by the Technology Innovation Institute (TII) in Abu Dhabi. It outperforms several models, like LLaMA, StableLM, RedPajama, and MPT, utilizing the FlashAttention method to achieve faster inference, resulting in significant speed improvements across different tasks (**Figure 1**).

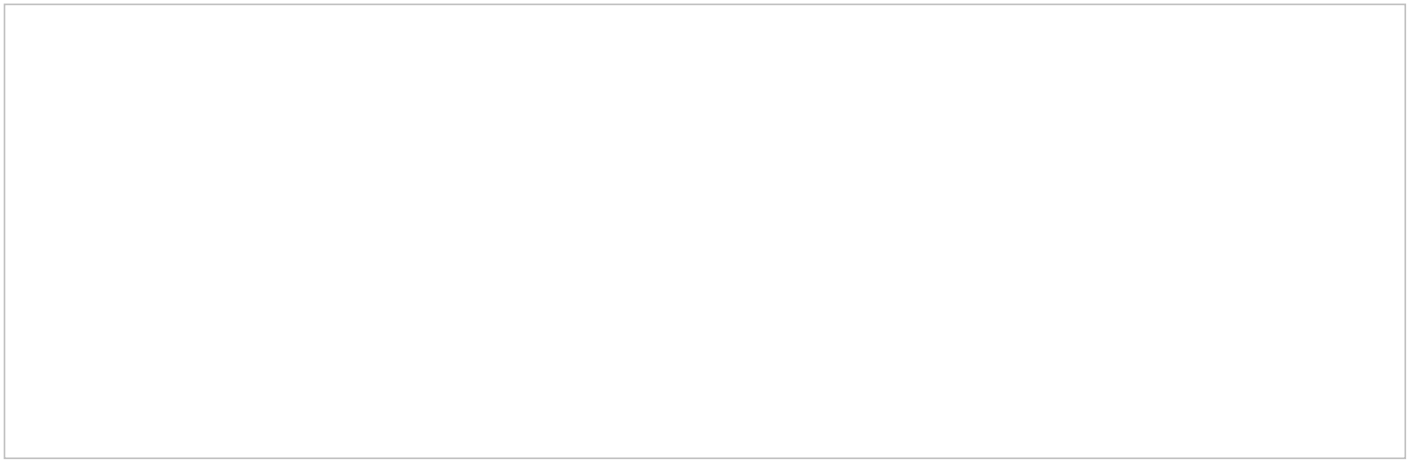

Figure 1. Hugging Face LLM leaderboard on June 6, 2023 (Image Source)

Running the script below will load the “tiiuae/falcon-7b” model from Hugging Face, tokenize, set training parameters, and use SFTTrainer for fine-tuning. The time it takes to fine-tune the model will vary depending on the compute and hyperparameters we set. Before running the script, it’s essential to set the following environment variable to ensure that we are selecting the Intel AMX ISA: export ONEDNN_MAX_CPU_ISA="AVX512_CORE_AMX"

```
# falcon-tune.py
import time
import argparse

from datasets import load_dataset
from trl import SFTTrainer
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    TrainingArguments)

def main(FLAGS):

    dataset = load_dataset("timdettmers/openassistant-guanaco", split="train")

    model_name = "tiiuae/falcon-7b"
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    tokenizer.pad_token = tokenizer.eos_token
    model = AutoModelForCausalLM.from_pretrained(model_name, trust_remote_code=True)

    print('setting training arguments')

    training_arguments = TrainingArguments(
        output_dir="./results",
        bf16=FLAGS.bf16, #change for CPU
        use_ipex=FLAGS.use_ipex, #change for CPU IPEX
        no_cuda=True,
        fp16_full_eval=False,
    )

    print('Creating SFTTrainer')

    trainer = SFTTrainer(
        model=model,
        train_dataset=dataset,
        dataset_text_field="text",
        max_seq_length=FLAGS.max_seq_length,
```

```
        tokenizer=tokenizer,
        args=training_arguments,
        packing=True,
    )

    print('Starting Training')
    start = time.time()

    trainer.train()

    total = time.time() - start
    print(f'Time to tune {total}')

if __name__ == "__main__":
    parser = argparse.ArgumentParser()

    parser.add_argument('-bf16',
                        '--bf16',
                        type=bool,
                        default=True,
                        help="activate mix precision training with bf16")
    parser.add_argument('-ipex',
                        '--use_ipex',
                        type=bool,
                        default=True,
                        help="used to control the maximum length of the generated text in text generation tasks")
    parser.add_argument('-msq',
                        '--max_seq_length',
                        type=int,
                        default=512,
                        help="specifies the number of highest probability tokens to consider at each step")

    FLAGS = parser.parse_args()
    main(FLAGS)
```

We can execute our script with the following command:

```
python falcon-tune.py --bf16 True --use_ipex True --max_seq_length 512
```

During training, we will see a progress bar indicating the estimated time to complete the process (**Figure 2**).

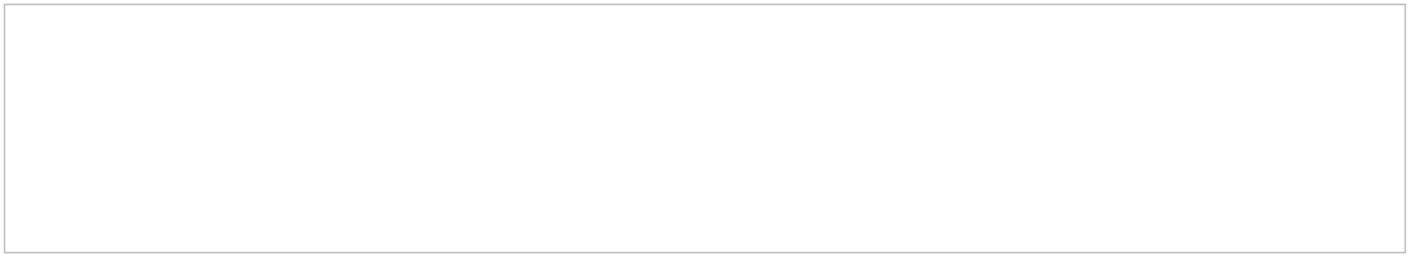

Figure 2. Training log from the fine-tuning process

Once training is complete, we should find a “results” directory with various checkpoint folders. The checkpoint folder with the highest number (checkpoint-3000) will contain all of our configurations, PyTorch model files, etc. (**Figure 3**). We will need the files in this folder to deploy our model and process inference requests.

Figure 3. Contents of final checkpoint folder

Inference with Our Tuned Falcon-7B Model

Now that our model has been fine-tuned, we can test it with a sample prompt using the following script to create a Hugging Face pipeline:

```
# falcon-tuned-inference.py

from transformers import AutoTokenizer, AutoModelForCausalLM, AutoConfig
import transformers
import torch
```

```
import argparse
import time

def main(FLAGS):

    model = AutoModelForCausalLM.from_pretrained(FLAGS.checkpoints, trust_remote_code=True)
    tokenizer = AutoTokenizer.from_pretrained(FLAGS.checkpoints, trust_remote_code=True)
    tokenizer.pad_token = tokenizer.eos_token

    generator = transformers.pipeline(
        "text-generation",
        model=model,
        tokenizer=tokenizer,
        torch_dtype=torch.bfloat16,
        trust_remote_code=True,
        device_map="auto",
    )

    user_input = "start"

    while user_input != "stop":

        user_input = input(f"Provide Input to tuned falcon: ")

        start = time.time()

        if user_input != "stop":
            sequences = generator(
            f""" {user_input}""",
            max_length=FLAGS.max_length,
            do_sample=False,
            top_k=FLAGS.top_k,
            num_return_sequences=1,
            eos_token_id=tokenizer.eos_token_id,)

        inference_time = time.time() - start

        for seq in sequences:
            print(f"Result: {seq['generated_text']}")

        print(f'Total Inference Time: {inference_time} seconds')

if __name__ == "__main__":
    parser = argparse.ArgumentParser()

    parser.add_argument('-c',
                        '--checkpoints',
                        type=str,
                        default=None,
                        help="path to model checkpoint files")
    parser.add_argument('-ml',
                        '--max_length',
                        type=int,
                        default="200",
                        help="used to control the maximum length of the generated text in text generation tasks")
    parser.add_argument('-tk',
                        '--top_k',
                        type=int,
                        default="10",
                        help="specifies the number of highest probability tokens to consider at each step")

    FLAGS = parser.parse_args()
    main(FLAGS)
```

To execute this script, run the following command:

```
python falcon-tuned-inference.py --checkpoints <PATH-TO-CHECKPOINT> --max_length 200 --top_k 10
```

When prompted by our script, we asked Falcon, *"Can you tell me three fun facts about space?"* Its response is shown in **Figure 4**. Except for the partially correct fact about Saturn, it seems like the model provided a factually accurate response and organized it in an easily interpretable format. There are encouraging signs that our fine-tuning has improved on the untuned model.

Figure 4. Response from tuned Falcon-7B

How does our model compare to the raw, untuned version of Falcon-7B? We tested the untuned Falcon-7B model with the same prompt as above (**Figure 5**). We can see that the untuned model has difficulty comprehending our request and formulating a coherent response. This is evidence that fine-tuning has improved the model's comprehension and overall response quality. Quantifying the degree of improvement would require running causal language modeling benchmarks, which is beyond the scope of this article.

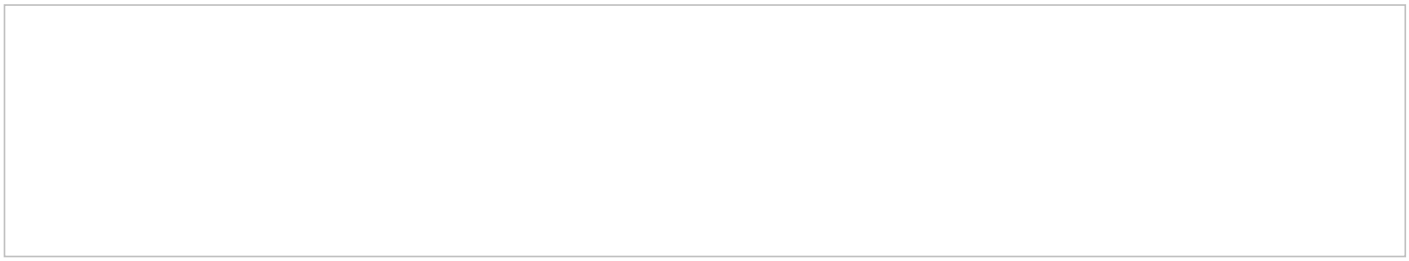

Figure 5. Response from untuned, raw Falcon-7B

Summary and Discussion

We now have a fine-tuned version of one of the most powerful "truly open-source" LLMs ever released! The intersection of Hugging Face's APIs, Intel's accelerated AI tooling, accessibility of CPU hardware, and Falcon's open-source licensing make this implementation an accessible option for various enterprises and AI application developers.

My goal was to enable this workload rather than analyze performance, so I omitted hardware performance and causal modeling metrics. Still, I encourage developers to explore opportunities to optimize this workflow using hyperparameter optimization, the Intel® Extension for Transformers, Intel® Neural Compressor, Parameter-Efficient Fine-tuning (PEFT), Low-Rank Adaptions of LLMs (LoRA), and fine-tuning Falcon on the Habana Gaudi*-1 and Gaudi-2 accelerators to improve training performance.
1