Use Case

Serving Llama-3.1-8B-Instruct model

Model Information

The Llama 3.2-Vision collection of multimodal large language models (LLMs) is a collection of pretrained and instruction-tuned image reasoning generative models in 11B and 90B sizes (text + images in / text out). The Llama 3.2-Vision instruction-tuned models are optimized for visual recognition, image reasoning, captioning, and answering general questions about an image. The models outperform many of the available open source and closed multimodal models on common industry benchmarks.

Usage

SSH into you instance and

Starting with transformers >= 4.45.0 onward, you can run inference using conversational messages that may include an image you can query about.

Make sure to update your transformers installation via

```
pip install --upgrade transformers
```

Then the model can be used with the MllamaForConditionalGeneration and AutoProcessor class to generate text:

```python
import requests
import torch
from PIL import Image
from transformers import MllamaForConditionalGeneration, AutoProcessor

model_id = "meta-llama/Llama-3.2-11B-Vision-Instruct"
```

```python

model = MllamaForConditionalGeneration.from_pretrained(
    model_id,
    torch_dtype=torch.bfloat16,
    device_map="auto",
)
processor = AutoProcessor.from_pretrained(model_id)

url = "https://huggingface.co/datasets/huggingface/documentation-images/resolve/0052a70beed5bf71b92610a43a52df6d286cd5f3/diffusers/rabbit.jpg"
image = Image.open(requests.get(url, stream=True).raw)

messages = [
    {"role": "user", "content": [
        {"type": "image"},
        {"type": "text", "text": "If I had to write a haiku for this one, it would be: "}
    ]}
]
input_text = processor.apply_chat_template(messages, add_generation_prompt=True)
inputs = processor(
    image,
    input_text,
    add_special_tokens=False,
    return_tensors="pt"
).to(model.device)

output = model.generate(**inputs, max_new_tokens=30)
print(processor.decode(output[0]))
```

Serving stable-diffusion-3.5-large-turbo model

Model Information

Stable Diffusion 3.5 Large Turbo is a Multimodal Diffusion Transformer (MMDiT) text-to-image model with Adversarial Diffusion Distillation (ADD) that features improved performance in image quality, typography, complex prompt understanding, and resource-efficiency, with a focus on fewer inference steps.

Please note: This model is released under the Stability Community License. Visit Stability AI to learn or contact us for commercial licensing details.

Usage

SSH into your instance and start by installing the dependencies:

```
pip install -U diffusers
```

and then you can run

```
import torch
from diffusers import StableDiffusion3Pipeline

pipe = StableDiffusion3Pipeline.from_pretrained("stabilityai/stable-diffusion-3.5-large-turbo", torch_dtype=torch.bfloat16)
pipe = pipe.to("cuda")

image = pipe(
    "A capybara holding a sign that reads Hello Fast World",
    num_inference_steps=4,
    guidance_scale=0.0,
).images[0]
image.save("capybara.png")
```

Serving whisper-large-v3-turbo model

Model Information

Whisper is a state-of-the-art model for automatic speech recognition (ASR) and speech translation, proposed in the paper Robust Speech Recognition via Large-Scale Weak Supervision by Alec Radford et al. from OpenAI. Trained on >5M hours of labeled data, Whisper demonstrates a strong ability to generalise to many datasets and domains in a zero-shot setting.

Whisper large-v3-turbo is a finetuned version of a pruned Whisper large-v3. In other words, it's the exact same model, except that the number of decoding layers have reduced from 32 to 4. As a result, the model is way faster, at the expense of a minor quality degradation. You can find more details about it in this GitHub discussion.

Usage

SSH into your instance and start by installing the dependencies, like transformers. For this example, we'll also install datasets to load toy audio dataset from the Hugging Face Hub, and accelerate to reduce the model loading time:

```
pip install --upgrade pip
pip install --upgrade transformers datasets[audio] accelerate
```

The model can be used with the `pipeline` class to transcribe audios of arbitrary length:

```
import torch
from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor, pipeline
from datasets import load_dataset


device = "cuda:0" if torch.cuda.is_available() else "cpu"
torch_dtype = torch.float16 if torch.cuda.is_available() else torch.float32

model_id = "openai/whisper-large-v3-turbo"

model = AutoModelForSpeechSeq2Seq.from_pretrained(
    model_id, torch_dtype=torch_dtype, low_cpu_mem_usage=True, use_safetensors=True
)
model.to(device)

processor = AutoProcessor.from_pretrained(model_id)

pipe = pipeline(
    "automatic-speech-recognition",
    model=model,
    tokenizer=processor.tokenizer,
    feature_extractor=processor.feature_extractor,
    torch_dtype=torch_dtype,
```

```python
    device=device,
)

dataset = load_dataset("distil-whisper/librispeech_long", "clean", 
split="validation")
sample = dataset[0]["audio"]

result = pipe(sample)
print(result["text"])
```

To transcribe a local audio file, simply pass the path to your audio file when you call the pipeline:

```python
result = pipe("audio.mp3")
```

Multiple audio files can be transcribed in parallel by specifying them as a list and setting the `batch_size` parameter:

```python
result = pipe(["audio_1.mp3", "audio_2.mp3"], batch_size=2)
```

Transformers is compatible with all Whisper decoding strategies, such as temperature fallback and condition on previous tokens. The following example demonstrates how to enable these heuristics:

```python
generate_kwargs = {
    "max_new_tokens": 448,
    "num_beams": 1,
    "condition_on_prev_tokens": False,
    "compression_ratio_threshold": 1.35,  # zlib compression ratio threshold 
(in token space)
    "temperature": (0.0, 0.2, 0.4, 0.6, 0.8, 1.0),
    "logprob_threshold": -1.0,
    "no_speech_threshold": 0.6,
    "return_timestamps": True,
}

result = pipe(sample, generate_kwargs=generate_kwargs)
```

Whisper predicts the language of the source audio automatically. If the source audio language is known *a-priori*, it can be passed as an argument to the pipeline:

```
result = pipe(sample, generate_kwargs={"language": "english"})
```

By default, Whisper performs the task of *speech transcription*, where the source audio language is the same as the target text language. To perform *speech translation*, where the target text is in English, set the task to `"translate"`:

```
result = pipe(sample, generate_kwargs={"task": "translate"})
```

Finally, the model can be made to predict timestamps. For sentence-level timestamps, pass the `return_timestamps` argument:

```
result = pipe(sample, return_timestamps=True)
print(result["chunks"])
```

And for word-level timestamps:

```
result = pipe(sample, return_timestamps="word")
print(result["chunks"])
```

The above arguments can be used in isolation or in combination. For example, to perform the task of speech transcription where the source audio is in French, and we want to return sentence-level timestamps, the following can be used:

```python
result = pipe(sample, return_timestamps=True, generate_kwargs={"language": "french", "task": "translate"})
print(result["chunks"])
```

For more control over the generation parameters, use the model + processor API directly:

```python
import torch
from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor
from datasets import Audio, load_dataset


device = "cuda:0" if torch.cuda.is_available() else "cpu"
torch_dtype = torch.float16 if torch.cuda.is_available() else torch.float32

model_id = "openai/whisper-large-v3-turbo"

model = AutoModelForSpeechSeq2Seq.from_pretrained(
    model_id, torch_dtype=torch_dtype, low_cpu_mem_usage=True
)
model.to(device)

processor = AutoProcessor.from_pretrained(model_id)

dataset = load_dataset("hf-internal-testing/librispeech_asr_dummy", "clean", split="validation")
dataset = dataset.cast_column("audio", Audio(processor.feature_extractor.sampling_rate))
sample = dataset[0]["audio"]

inputs = processor(
    sample["array"],
    sampling_rate=sample["sampling_rate"],
    return_tensors="pt",
    truncation=False,
    padding="longest",
    return_attention_mask=True,
)
inputs = inputs.to(device, dtype=torch_dtype)
```

```python

gen_kwargs = {
    "max_new_tokens": 448,
    "num_beams": 1,
    "condition_on_prev_tokens": False,
    "compression_ratio_threshold": 1.35,  # zlib compression ratio threshold (in token space)
    "temperature": (0.0, 0.2, 0.4, 0.6, 0.8, 1.0),
    "logprob_threshold": -1.0,
    "no_speech_threshold": 0.6,
    "return_timestamps": True,
}

pred_ids = model.generate(**inputs, **gen_kwargs)
pred_text = processor.batch_decode(pred_ids, skip_special_tokens=True, decode_with_timestamps=False)

print(pred_text)
```