Fast inference with vLLM (Mixtral 8x7B)

In this example, we show how to run basic inference, using `vLLM` to take advantage of PagedAttention, which speeds up sequential inferences with optimized key-value caching.

We are running a variant of Mistral AI's ~56 billion parameter mixture-of-experts model Mixtral 8×7B model that has been additionally finetuned by Nous Research. You can expect ~3 minute cold starts. For a single request, the throughput is around 50 tokens/second. The larger the batch of prompts, the higher the throughput (up to hundreds of tokens per second).

Setup

```
import os
import time

import modal

MODEL_DIR = "/model"
MODEL_NAME = "NousResearch/Nous-Hermes-2-Mixtral-8x7B-DPO"
MODEL_REVISION = "286ae6737d048ad1d965c2e830864df02db50f2f"
GPU_CONFIG = modal.gpu.A100(size="80GB", count=2)
```

Define a container image

We want to create a Modal image which has the model weights pre-saved to a directory. The benefit of this is that the container no longer has to re-download the model from Huggingface - instead, it will take advantage of Modal's internal filesystem for faster cold starts.

Download the weights

We can download the model to a particular directory using the HuggingFace utility function `snapshot_download`.

If you adapt this example to run another model, note that for this step to work on a gated model the `HF_TOKEN` environment variable must be set and provided as a Modal Secret.

Mixtral is beefy, at nearly 100 GB in `safetensors` format, so this can take some time — at least a few minutes.

```
def download_model_to_image(model_dir, model_name, model_revision):
    from huggingface_hub import snapshot_download
    from transformers.utils import move_cache

    os.makedirs(model_dir, exist_ok=True)

    snapshot_download(
        model_name,
        revision=model_revision,
        local_dir=model_dir,
        ignore_patterns=["*.pt", "*.bin"],  # Using safetensors
    )
    move_cache()
```

Image definition

We'll start from a basic Linux container image, install `vllm` and related libraries, and then use `run_function` to run the function defined above and ensure the weights of the model are saved within the container image.

```
vllm_image = (
    modal.Image.debian_slim(python_version="3.10")
    .pip_install(
        "vllm==0.4.0.post1",
        "torch==2.1.2",
        "transformers==4.39.3",
        "ray==2.10.0",
        "hf-transfer==0.1.6",
        "huggingface_hub==0.22.2",
    )
```

```
    .env({"HF_HUB_ENABLE_HF_TRANSFER": "1"})
    .run_function(
        download_model_to_image,
        timeout=60 * 20,
        kwargs={
            "model_dir": MODEL_DIR,
            "model_name": MODEL_NAME,
            "model_revision": MODEL_REVISION,
        },
    )
)

app = modal.App("example-vllm-mixtral")
```

The model class

The inference function is best represented with Modal's class syntax and the `@enter` decorator. This enables us to load the model into memory just once every time a container starts up, and keep it cached on the GPU for each subsequent invocation of the function.

The `vLLM` library allows the code to remain quite clean. We do have to patch the multi-GPU setup due to issues with Ray.

```
@app.cls(
    gpu=GPU_CONFIG,
    timeout=60 * 10,
    container_idle_timeout=60 * 10,
    allow_concurrent_inputs=10,
    image=vllm_image,
)
class Model:
    @modal.enter()
    def start_engine(self):
        from vllm.engine.arg_utils import AsyncEngineArgs
        from vllm.engine.async_llm_engine import AsyncLLMEngine

        print("🥶 cold starting inference")
        start = time.monotonic_ns()

        engine_args = AsyncEngineArgs(
            model=MODEL_DIR,
            tensor_parallel_size=GPU_CONFIG.count,
            gpu_memory_utilization=0.90,
            enforce_eager=False,  # capture the graph for faster inference, but slower col
            disable_log_stats=True,  # disable logging so we can stream tokens
            disable_log_requests=True,
        )
```

```
        # this can take some time!
        self.engine = AsyncLLMEngine.from_engine_args(engine_args)
        duration_s = (time.monotonic_ns() - start) / 1e9
        print(f"🏎️ engine started in {duration_s:.0f}s")

    @modal.method()
    async def completion_stream(self, user_question):
        from vllm import SamplingParams
        from vllm.utils import random_uuid

        sampling_params = SamplingParams(
            temperature=0.75,
            max_tokens=128,
            repetition_penalty=1.1,
        )

        request_id = random_uuid()
        result_generator = self.engine.generate(
            user_question,
            sampling_params,
            request_id,
        )
        index, num_tokens = 0, 0
        start = time.monotonic_ns()
        async for output in result_generator:
            if (
                output.outputs[0].text
                and "\ufffd" == output.outputs[0].text[-1]
            ):
                continue
            text_delta = output.outputs[0].text[index:]
            index = len(output.outputs[0].text)
            num_tokens = len(output.outputs[0].token_ids)

            yield text_delta
        duration_s = (time.monotonic_ns() - start) / 1e9

        yield (
            f"\n\tGenerated {num_tokens} tokens from {MODEL_NAME} in {duration_s:.1f}s,"
            f" throughput = {num_tokens / duration_s:.0f} tokens/second on {GPU_CONFIG}.\n
        )

    @modal.exit()
    def stop_engine(self):
        if GPU_CONFIG.count > 1:
            import ray

            ray.shutdown()
```

Run the model

We define a `local_entrypoint` to call our remote function sequentially for a list of inputs. You can run this locally with `modal run -q vllm_mixtral.py`. The `q` flag enables the text to stream in your local terminal.

```
@app.local_entrypoint()
def main():
    questions = [
        "Implement a Python function to compute the Fibonacci numbers.",
        "What is the fable involving a fox and grapes?",
        "What were the major contributing factors to the fall of the Roman Empire?",
        "Describe the city of the future, considering advances in technology, environmenta
        "What is the product of 9 and 8?",
        "Who was Emperor Norton I, and what was his significance in San Francisco's histor
    ]
    model = Model()
    for question in questions:
        print("Sending new request:", question, "\n\n")
        for text in model.completion_stream.remote_gen(question):
            print(text, end="", flush=text.endswith("\n"))
```

Deploy and invoke the model

Once we deploy this model with `modal deploy vllm_mixtral.py`, we can invoke inference from other apps, sharing the same pool of GPU containers with all other apps we might need.

```
$ python
>>> import modal
>>> f = modal.Function.lookup("example-vllm-mixtral", "Model.completion_stream")
>>> for text in f.remote_gen("What is the story about the fox and grapes?"):
>>>     print(text, end="", flush=text.endswith("\n"))
'The story about the fox and grapes ...
```

Coupling a frontend web application

We can stream inference from a FastAPI backend, also deployed on Modal.

You can try our deployment here.

```
from pathlib import Path

from modal import Mount, asgi_app

frontend_path = Path(__file__).parent.parent / "llm-frontend"
```

```python
@app.function(
    mounts=[Mount.from_local_dir(frontend_path, remote_path="/assets")],
    keep_warm=1,
    allow_concurrent_inputs=20,
    timeout=60 * 10,
)
@asgi_app()
def vllm_mixtral():
    import json

    import fastapi
    import fastapi.staticfiles
    from fastapi.responses import StreamingResponse

    web_app = fastapi.FastAPI()

    @web_app.get("/stats")
    async def stats():
        stats = await Model().completion_stream.get_current_stats.aio()
        return {
            "backlog": stats.backlog,
            "num_total_runners": stats.num_total_runners,
            "model": MODEL_NAME + " (vLLM)",
        }

    @web_app.get("/completion/{question}")
    async def completion(question: str):
        from urllib.parse import unquote

        async def generate():
            async for text in Model().completion_stream.remote_gen.aio(
                unquote(question)
            ):
                yield f"data: {json.dumps(dict(text=text), ensure_ascii=False)}\n\n"

        return StreamingResponse(generate(), media_type="text/event-stream")

    web_app.mount(
        "/", fastapi.staticfiles.StaticFiles(directory="/assets", html=True)
    )
    return web_app
```

© 2024