Featured

Getting started

Hello, world

Simple web scraper

Large language models (LLMs)

Featured: Serverless TensorRT-LLM

Stable Diffusion XL 1.0

View on GitHub

This example is similar to the Stable Diffusion CLI example, but it generates images from the larger SDXL 1.0 model. Specifically, it runs the first set of steps with the base model, followed by the refiner model.

Try out the live demo here! The first generation may include a cold-start, which takes around 20 seconds. The inference speed depends on the GPU and step count (for reference, an A100 runs 40 steps in 8 seconds).

Basic setup

```
import io
from pathlib import Path

from modal import (
    App,
    Image,
    Mount,
    asgi_app,
    build,
    enter,
    gpu,
    method,
    web_endpoint,
)
```

Define a container image

To take advantage of Modal's blazing fast cold-start times, we'll need to download our model weights inside our container image with a download function. We ignore binaries, ONNX weights and 32-bit weights.

Tip: avoid using global variables in this function to ensure the download step detects model changes and triggers a rebuild.

```
sdxl_image = (
    Image.debian_slim(python_version="3.10")
    .apt_install(
        "libglib2.0-0", "libsm6", "libxrender1", "libxext6", "ffmpeg", "libgl1"
    )
    .pip_install(
        "diffusers==0.26.3",
        "invisible_watermark==0.2.0",
        "transformers~=4.38.2",
        "accelerate==0.27.2",
        "safetensors==0.4.2",
    )
)

app = App(
    "stable-diffusion-xl"
)  # Note: prior to April 2024, "app" was called "stub"

with sdxl_image.imports():
    import torch
    from diffusers import DiffusionPipeline
    from fastapi import Response
```

Load model and run inference

The container lifecycle `@enter` decorator loads the model at startup. Then, we evaluate it in the `run_inference` function.

To avoid excessive cold-starts, we set the idle timeout to 240 seconds, meaning once a GPU has loaded the model it will stay online for 4 minutes before spinning down. This can be adjusted for cost/experience trade-offs.

```
@app.cls(gpu=gpu.A10G(), container_idle_timeout=240, image=sdxl_image)
class Model:
    @build()
    def build(self):
        from huggingface_hub import snapshot_download

        ignore = [
            "*.bin",
            "*.onnx_data",
            "*/diffusion_pytorch_model.safetensors",
        ]
        snapshot_download(
            "stabilityai/stable-diffusion-xl-base-1.0", ignore_patterns=ignore
        )
        snapshot_download(
            "stabilityai/stable-diffusion-xl-refiner-1.0",
            ignore_patterns=ignore,
        )

    @enter()
    def enter(self):
        load_options = dict(
            torch_dtype=torch.float16,
            use_safetensors=True,
            variant="fp16",
            device_map="auto",
        )

        # Load base model
        self.base = DiffusionPipeline.from_pretrained(
            "stabilityai/stable-diffusion-xl-base-1.0", **load_options
        )

        # Load refiner model
        self.refiner = DiffusionPipeline.from_pretrained(
            "stabilityai/stable-diffusion-xl-refiner-1.0",
            text_encoder_2=self.base.text_encoder_2,
            vae=self.base.vae,
            **load_options,
        )

        # Compiling the model graph is JIT so this will increase inference time for the fi
        # but speed up subsequent runs. Uncomment to enable.
        # self.base.unet = torch.compile(self.base.unet, mode="reduce-overhead", fullgraph
        # self.refiner.unet = torch.compile(self.refiner.unet, mode="reduce-overhead", ful

    def _inference(self, prompt, n_steps=24, high_noise_frac=0.8):
        negative_prompt = "disfigured, ugly, deformed"
        image = self.base(
            prompt=prompt,
            negative_prompt=negative_prompt,
            num_inference_steps=n_steps,
            denoising_end=high_noise_frac,
            output_type="latent",
        ).images
        image = self.refiner(
            prompt=prompt,
            negative_prompt=negative_prompt,
            num_inference_steps=n_steps,
            denoising_start=high_noise_frac,
            image=image,
        ).images[0]

        byte_stream = io.BytesIO()
        image.save(byte_stream, format="JPEG")

        return byte_stream

    @method()
    def inference(self, prompt, n_steps=24, high_noise_frac=0.8):
        return self._inference(
            prompt, n_steps=n_steps, high_noise_frac=high_noise_frac
        ).getvalue()

    @web_endpoint()
    def web_inference(self, prompt, n_steps=24, high_noise_frac=0.8):
        return Response(
            content=self._inference(
                prompt, n_steps=n_steps, high_noise_frac=high_noise_frac
            ).getvalue(),
            media_type="image/jpeg",
        )
```

And this is our entrypoint; where the CLI is invoked. Explore CLI options with: `modal run stable_diffusion_xl.py —help

```
@app.local_entrypoint()
def main(prompt: str = "Unicorns and leprechauns sign a peace treaty"):
    image_bytes = Model().inference.remote(prompt)

    dir = Path("/tmp/stable-diffusion-xl")
    if not dir.exists():
        dir.mkdir(exist_ok=True, parents=True)

    output_path = dir / "output.png"
    print(f"Saving it to {output_path}")
    with open(output_path, "wb") as f:
        f.write(image_bytes)
```

A user interface

Here we ship a simple web application that exposes a front-end (written in Alpine.js) for our backend deployment.

The Model class will serve multiple users from a its own shared pool of warm GPU containers automatically.

We can deploy this with `modal deploy stable_diffusion_xl.py`.

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

web_image = Image.debian_slim().pip_install("jinja2")


@app.function(
    image=web_image,
    mounts=[Mount.from_local_dir(frontend_path, remote_path="/assets")],
    allow_concurrent_inputs=20,
)
@asgi_app()
def ui():
    import fastapi.staticfiles
    from fastapi import FastAPI, Request
    from fastapi.templating import Jinja2Templates

    web_app = FastAPI()
    templates = Jinja2Templates(directory="/assets")

    @web_app.get("/")
    async def read_root(request: Request):
        return templates.TemplateResponse(
            "index.html",
            {
                "request": request,
                "inference_url": Model.web_inference.web_url,
                "model_name": "Stable Diffusion XL",
                "default_prompt": "A cinematic shot of a baby raccoon wearing an intricate
            },
        )

    web_app.mount(
        "/static",
        fastapi.staticfiles.StaticFiles(directory="/assets"),
        name="static",
    )

    return web_app
```