Featured

Getting started

Hello, world

Simple web scraper

Large language models (LLMs)

Featured: Serverless TensorRT-LLM

Play with the ControlNet demos

View on GitHub

This example allows you to play with all 10 demonstration Gradio apps from the new and amazing ControlNet project. ControlNet provides a minimal interface allowing users to use images to constrain StableDiffusion's generation process. With ControlNet, users can easily condition the StableDiffusion image generation with different spatial contexts including a depth maps, segmentation maps, scribble drawings, and keypoints!

0:00

Imports and config preamble

```python
import importlib
import os
import pathlib
from dataclasses import dataclass, field

from fastapi import FastAPI
from modal import App, Image, Secret, asgi_app
```

Below are the configuration objects for all **10** demos provided in the original lllyasviel/ControlNet repo. The demos each depend on their own custom pretrained StableDiffusion model, and these models are 5-6GB each. We can only run one demo at a time, so this module avoids downloading the model and 'detector' dependencies for all 10 demos and instead uses the demo configuration object to download only what's necessary for the chosen demo.

Even just limiting our dependencies setup to what's required for one demo, the resulting container image is *huge*.

```
@dataclass(frozen=True)
class DemoApp:
    """Config object defining a ControlNet demo app's specific dependencies."""

    name: str
    model_files: list[str]
    detector_files: list[str] = field(default_factory=list)


demos = [
    DemoApp(
        name="canny2image",
        model_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/models/control_sd15
        ],
    ),
    DemoApp(
        name="depth2image",
        model_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/models/control_sd15
        ],
        detector_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/annotator/ckpts/dpt
        ],
    ),
    DemoApp(
        name="fake_scribble2image",
        model_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/models/control_sd15
        ],
        detector_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/annotator/ckpts/net
        ],
    ),
    DemoApp(
        name="hed2image",
        model_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/models/control_sd15
        ],
        detector_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/annotator/ckpts/net
        ],
    ),
    DemoApp(
        name="hough2image",
        model_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/models/control_sd15
        ],
        detector_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/annotator/ckpts/mls
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/annotator/ckpts/mls
        ],
    ),
    DemoApp(
        name="normal2image",
        model_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/models/control_sd15
        ],
    ),
    DemoApp(
        name="pose2image",
        model_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/models/control_sd15
        ],
        detector_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/annotator/ckpts/bod
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/annotator/ckpts/han
        ],
    ),
    DemoApp(
        name="scribble2image",
        model_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/models/control_sd15
        ],
    ),
    DemoApp(
        name="scribble2image_interactive",
        model_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/models/control_sd15
        ],
    ),
    DemoApp(
        name="seg2image",
        model_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/models/control_sd15
        ],
        detector_files=[
            "https://huggingface.co/lllyasviel/ControlNet/resolve/main/annotator/ckpts/upe
        ],
    ),
]
demos_map: dict[str, DemoApp] = {d.name: d for d in demos}
```

Pick a demo, any demo

Simply by changing the `DEMO_NAME` below, you can change which ControlNet demo app is setup and run by this Modal script.

```
DEMO_NAME = "scribble2image"  # Change this value to change the active demo app.
selected_demo = demos_map[DEMO_NAME]
```

Setting up the dependencies

ControlNet requires *a lot* of dependencies which could be fiddly to setup manually, but Modal's programmatic container image building Python APIs handle this complexity straightforwardly and automatically.

To run any of the 10 demo apps, we need the following:

1. a base Python 3 Linux image (we use Debian Slim)
2. a bunch of third party PyPi packages
3. `git` , so that we can download the ControlNet source code (there's no `controlnet` PyPi package)
4. some image process Linux system packages, including `ffmpeg`
5. and demo specific pre-trained model and detector `.pth` files

That's a lot! Fortunately, the code below is already written for you that stitches together a working container image ready to produce remarkable ControlNet images.

Note: a ControlNet model pipeline is now available in Huggingface's `diffusers` package. But this does not contain the demo apps.

```
def download_file(url: str, output_path: pathlib.Path):
    import httpx
    from tqdm import tqdm

    with open(output_path, "wb") as download_file:
        with httpx.stream("GET", url, follow_redirects=True) as response:
            total = int(response.headers["Content-Length"])
            with tqdm(
                total=total, unit_scale=True, unit_divisor=1024, unit="B"
            ) as progress:
                num_bytes_downloaded = response.num_bytes_downloaded
                for chunk in response.iter_bytes():
                    download_file.write(chunk)
                    progress.update(
                        response.num_bytes_downloaded - num_bytes_downloaded
                )
                num_bytes_downloaded = response.num_bytes_downloaded


def download_demo_files() -> None:
    """
    The ControlNet repo instructs: 'Make sure that SD models are put in "ControlNet/models
    'ControlNet' is just the repo root, so we place in /root/models.

    The ControlNet repo also instructs: 'Make sure that... detectors are put in "ControlNe
    'ControlNet' is just the repo root, so we place in /root/annotator/ckpts.
    """
    demo = demos_map[os.environ["DEMO_NAME"]]
    models_dir = pathlib.Path("/root/models")
    for url in demo.model_files:
        filepath = pathlib.Path(url).name
        download_file(url=url, output_path=models_dir / filepath)
        print(f"download complete for {filepath}")

    detectors_dir = pathlib.Path("/root/annotator/ckpts")
    for url in demo.detector_files:
        filepath = pathlib.Path(url).name
        download_file(url=url, output_path=detectors_dir / filepath)
        print(f"download complete for {filepath}")
    print("🎉 finished baking demo file(s) into image.")


image = (
    Image.debian_slim(python_version="3.10")
    .pip_install(
        "gradio==3.16.2",
        "albumentations==1.3.0",
        "opencv-contrib-python",
        "imageio==2.9.0",
        "imageio-ffmpeg==0.4.2",
        "pytorch-lightning==1.5.0",
        "omegaconf==2.1.1",
        "test-tube>=0.7.5",
        "streamlit==1.12.1",
        "einops==0.3.0",
        "transformers==4.19.2",
        "webdataset==0.2.5",
        "kornia==0.6",
        "open_clip_torch==2.0.2",
        "invisible-watermark>=0.1.5",
        "streamlit-drawable-canvas==0.8.0",
        "torchmetrics==0.6.0",
        "timm==0.6.12",
        "addict==2.4.0",
        "yapf==0.32.0",
        "prettytable==3.6.0",
        "safetensors==0.2.7",
        "basicsr==1.4.2",
        "tqdm~=4.64.1",
    )
    # xformers library offers performance improvement.
    .pip_install("xformers", pre=True)
    .apt_install("git")
    # Here we place the latest ControlNet repository code into /root.
    # Because /root is almost empty, but not entirely empty, `git clone` won't work,
    # so this `init` then `checkout` workaround is used.
    .run_commands(
        "cd /root && git init .",
        "cd /root && git remote add --fetch origin https://github.com/lllyasviel/ControlNe
        "cd /root && git checkout main",
    )
    .apt_install("ffmpeg", "libsm6", "libxext6")
    .run_function(
        download_demo_files,
        secrets=[Secret.from_dict({"DEMO_NAME": DEMO_NAME})],
    )
)
app = App(
    name="example-controlnet", image=image
)  # Note: prior to April 2024, "app" was called "stub"

web_app = FastAPI()
```

Serving the Gradio web UI

Each ControlNet gradio demo module exposes a `block` Gradio interface running in queue-mode, which is initialized in module scope on import and served on `0.0.0.0`. We want the block interface object, but the queueing and launched webserver aren't compatible with Modal's serverless web endpoint interface, so in the `import_gradio_app_blocks` function we patch out these behaviors.

```
def import_gradio_app_blocks(demo: DemoApp):
    from gradio import blocks

    # The ControlNet repo demo scripts are written to be run as
    # standalone scripts, and have a lot of code that executes
    # in global scope on import, including the launch of a Gradio web server.
    # We want Modal to control the Gradio web app serving, so we
    # monkeypatch the .launch() function to be a no-op.
    blocks.Blocks.launch = lambda self, server_name: print(
        "launch() has been monkeypatched to do nothing."
    )

    # each demo app module is a file like gradio_{name}.py
    module_name = f"gradio_{demo.name}"
    mod = importlib.import_module(module_name)
    blocks = mod.block
    # disable queueing mode, which is incompatible with our Modal web app setup.
    blocks.enable_queue = False
    return blocks
```

Because the ControlNet gradio apps are so time and compute intensive to cold-start, the web app function is limited to running just 1 warm container (concurrency_limit=1). This way, while playing with the demos we can pay the cold-start cost once and have all web requests hit the same warm container. Spinning up extra containers to handle additional requests would not be efficient given the cold-start time. We set the container_idle_timeout to 600 seconds so the container will be kept running for 10 minutes after the last request, to keep the app responsive in case of continued experimentation.

```
@app.function(
    gpu="A10G",
    concurrency_limit=1,
    container_idle_timeout=600,
)
@asgi_app()
def run():
    from gradio.routes import mount_gradio_app

    # mount for execution on Modal
    return mount_gradio_app(
        app=web_app,
        blocks=import_gradio_app_blocks(demo=selected_demo),
        path="/",
    )
```

Have fun!

Serve your chosen demo app with `modal serve controlnet_gradio_demos.py`. If you don't have any images ready at hand, try one that's in the `06_gpu_and_ml/controlnet/demo_images/` folder.

StableDiffusion was already impressive enough, but ControlNet's ability to so accurately and intuitively constrain the image generation process is sure to put a big, dumb grin on your face.