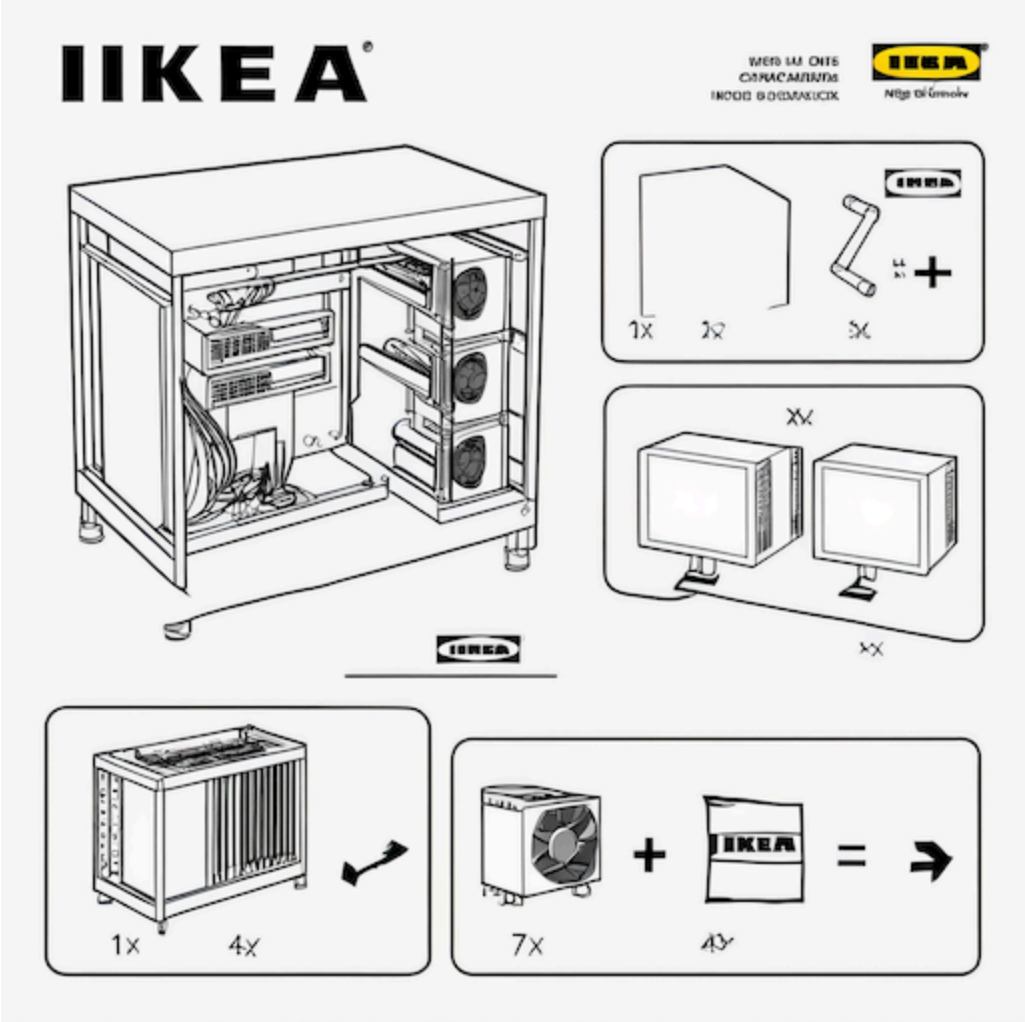
LoRAs Galore: Create a LoRA Playground with Modal, Gradio, and S3

View on GitHub

This example shows how to mount an S3 bucket in a Modal app using `CloudBucketMount`. We will download a bunch of LoRA adapters from the HuggingFace Hub into our S3 bucket then read from that bucket, on the fly, when doing inference.

By default, we use the IKEA instructions LoRA as an example, which produces the following image when prompted to generate “IKEA instructions for building a GPU rig for deep learning”:



By the end of this example, we've deployed a "playground" app where anyone with a browser can try out these custom models. That's the power of Modal: custom, autoscaling AI applications, deployed in seconds. You can try out our deployment here.

Basic setup

```
import io
import os
from pathlib import Path
from typing import Optional

from modal import (
    App,
    CloudBucketMount,  # the star of the show
    Image,
    Secret,
    asgi_app,
    build,
    enter,
    method,
)
```

You will need to have an S3 bucket and AWS credentials to run this example. Refer to the documentation for the detailed IAM permissions those credentials will need.

After you are done creating a bucket and configuring IAM settings, you now need to create a Modal Secret. Navigate to the "Secrets" tab and click on the AWS card, then fill in the fields with the AWS key and secret created previously. Name the Secret `s3-bucket-secret`.

```
bucket_secret = Secret.from_name("s3-bucket-secret")

MOUNT_PATH: Path = Path("/mnt/bucket")
LORAS_PATH: Path = MOUNT_PATH / "loras/v5"
```

Modal runs serverless functions inside containers. The environments those functions run in are defined by the container `Image`. The line below constructs an image with the dependencies we need — no need to install them locally.

```
image = Image.debian_slim().pip_install(
    "huggingface_hub==0.21.4",
    "transformers==4.38.2",
    "diffusers==0.26.3",
    "peft==0.9.0",
    "accelerate==0.27.2",
)

with image.imports():
    # we import these dependencies only inside the container
    import diffusers
    import huggingface_hub
    import torch
```

We attach the S3 bucket to all the Modal functions in this app by mounting it on the filesystem they see, passing a `CloudBucketMount` to the `volumes` dictionary argument. We can read and write to this mounted bucket (almost) as if it were a local directory.

```
app = App(
    "loras-galore",
    image=image,
    volumes={
        MOUNT_PATH: CloudBucketMount(
            "modal-s3mount-test-bucket",
            secret=bucket_secret,
        )
    },
)
```

Acquiring LoRA weights

`search_loras()` will use the Hub API to search for LoRAs. We limit LoRAs to a maximum size to avoid downloading very large model weights. We went with 800 MiB, but feel free to adapt to what works best for you.

```
@app.function()
def search_loras(limit: int, max_model_size: int = 1024 * 1024 * 1024):
    api = huggingface_hub.HfApi()

    model_ids: list[str] = []
    for model in api.list_models(
        tags=["lora", "base_model:stabilityai/stable-diffusion-xl-base-1.0"],
        library="diffusers",
        sort="downloads",  # sort by most downloaded
    ):
        try:
            model_size = 0
            for file in api.list_files_info(model.id):
                model_size += file.size

        except huggingface_hub.utils.GatedRepoError:
            print(f"gated model ({model.id}); skipping")
            continue

        # Skip models that are larger than file limit.
        if model_size > max_model_size:
            print(f"model {model.id} is too large; skipping")
            continue

        model_ids.append(model.id)
        if len(model_ids) >= limit:
            return model_ids

    return model_ids
```

We want to take the LoRA weights we found and move them from Hugging Face onto S3, where they'll be accessible, at short latency and high throughput, for our Modal functions. Downloading files in this mount will automatically upload files to S3. To speed things up, we will run this function in parallel using Modal's `map`.

```
@app.function()
def download_lora(repository_id: str) -> Optional[str]:
    os.environ["HF_HUB_DISABLE_SYMLINKS_WARNING"] = "1"

    # CloudBucketMounts will report 0 bytes of available space leading to many
    # unnecessary warnings, so we patch the method that emits those warnings.
```

```
    from huggingface_hub import file_download

    file_download._check_disk_space = lambda x, y: False

    repository_path = LORAS_PATH / repository_id
    try:
        # skip models we've already downloaded
        if not repository_path.exists():
            huggingface_hub.snapshot_download(
                repository_id,
                local_dir=repository_path.as_posix().replace(".", "_"),
                allow_patterns=["*.safetensors"],
            )
        downloaded_lora = len(list(repository_path.rglob("*.safetensors"))) > 0
    except OSError:
        downloaded_lora = False
    except FileNotFoundError:
        downloaded_lora = False
    if downloaded_lora:
        return repository_id
    else:
        return None
```

Inference with LoRAs

We define a `StableDiffusionLoRA` class to organize our inference code. We load Stable Diffusion XL 1.0 as a base model, then, when doing inference, we load whichever LoRA the user specifies from the S3 bucket. For more on the decorators we use on the methods below to speed up building and booting, check out the container lifecycle hooks guide.

```
@app.cls(gpu="a10g")  # A10G GPUs are great for inference
class StableDiffusionLoRA:
    pipe_id = "stabilityai/stable-diffusion-xl-base-1.0"

    @build()  # when we setup our image, we download the base model
    def build(self):
        diffusers.DiffusionPipeline.from_pretrained(
            self.pipe_id, torch_dtype=torch.float16
        )

    @enter()  # when a new container starts, we load the base model into the GPU
    def load(self):
        self.pipe = diffusers.DiffusionPipeline.from_pretrained(
            self.pipe_id, torch_dtype=torch.float16
        ).to("cuda")

    @method()  # at inference time, we pull in the LoRA weights and pass the final model t
    def run_inference_with_lora(
```

```
        self, lora_id: str, prompt: str, seed: int = 8888
    ) -> bytes:
        for file in (LORAS_PATH / lora_id).rglob("*.safetensors"):
            self.pipe.load_lora_weights(lora_id, weight_name=file.name)
            break

        lora_scale = 0.9
        image = self.pipe(
            prompt,
            num_inference_steps=10,
            cross_attention_kwargs={"scale": lora_scale},
            generator=torch.manual_seed(seed),
        ).images[0]

        buffer = io.BytesIO()
        image.save(buffer, format="PNG")

        return buffer.getvalue()
```

Try it locally!

To use our inference code from our local command line, we add a `local_entrypoint` to our `app`. Run it using `modal run cloud_bucket_mount_loras.py`, and pass `--help` to see the available options.

The inference code will run on our machines, but the results will be available on yours.

```
@app.local_entrypoint()
def main(
    limit: int = 100,
    example_lora: str = "ostris/ikea-instructions-lora-sdxl",
    prompt: str = "IKEA instructions for building a GPU rig for deep learning",
    seed: int = 8888,
):
    # Download LoRAs in parallel.
    lora_model_ids = [example_lora]
    lora_model_ids += search_loras.remote(limit)

    downloaded_loras = []
    for model in download_lora.map(lora_model_ids):
        if model:
            downloaded_loras.append(model)

    print(f"downloaded {len(downloaded_loras)} loras => {downloaded_loras}")

    # Run inference using one of the downloaded LoRAs.
    byte_stream = StableDiffusionLoRA().run_inference_with_lora.remote(
        example_lora, prompt, seed
    )
```

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

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

LoRA Exploradora: A hosted Gradio interface

Command line tools are cool, but we can do better! With the Gradio library by Hugging Face, we can create a simple web interface around our Python inference function, then use Modal to host it for anyone to try out.

To set up your own, run `modal deploy cloud_bucket_mount_loras.py` and navigate to the URL it prints out. If you're playing with the code, use `modal serve` instead to see changes live.

```
from fastapi import FastAPI

web_app = FastAPI()
web_image = Image.debian_slim().pip_install("gradio~=3.50.2", "pillow~=10.2.0")


@app.function(image=web_image, keep_warm=1, container_idle_timeout=60 * 20)
@asgi_app()
def ui():
    """A simple Gradio interface around our LoRA inference."""
    import io

    import gradio as gr
    from gradio.routes import mount_gradio_app
    from PIL import Image

    # determine with loras are available
    lora_ids = [
        f"{lora_dir.parent.stem}/{lora_dir.stem}"
        for lora_dir in LORAS_PATH.glob("*/*")
    ]

    # pick one to be default, set a default prompt
    default_lora_id = (
        "ostris/ikea-instructions-lora-sdxl"
        if "ostris/ikea-instructions-lora-sdxl" in lora_ids
        else lora_ids[0]
    )
    default_prompt = (
        "IKEA instructions for building a GPU rig for deep learning"
```

```
        if default_lora_id == "ostris/ikea-instructions-lora-sdxl"
        else "text"
    )

    # the simple path to making an app on Gradio is an Interface: a UI wrapped around a fu
    def go(lora_id: str, prompt: str, seed: int) -> Image:
        return Image.open(
            io.BytesIO(
                StableDiffusionLoRA().run_inference_with_lora.remote(
                    lora_id, prompt, seed
                )
            ),
        )

    iface = gr.Interface(
        go,
        inputs=[  # the inputs to go/our inference function
            gr.Dropdown(
                choices=lora_ids, value=default_lora_id, label="👉 LoRA ID"
            ),
            gr.Textbox(default_prompt, label="🎨 Prompt"),
            gr.Number(value=8888, label="🎲 Random Seed"),
        ],
        outputs=gr.Image(label="Generated Image"),
        # some extra bits to make it look nicer
        title="LoRAs Galore",
        description="# Try out some of the top custom SDXL models!"
        "\n\nPick a LoRA finetune of SDXL from the dropdown, then prompt it to generate an
        "\n\nCheck out [the code on GitHub](https://github.com/modal-labs/modal-examples/b
        " if you want to create your own version or just see how it works."
        "\n\nPowered by [Modal](https://modal.com) 🚀",
        theme="soft",
        allow_flagging="never",
    )

    return mount_gradio_app(app=web_app, blocks=iface, path="/")


def as_slug(name):
    """Converts a string, e.g. a prompt, into something we can use as a filename."""
    import re

    s = str(name).strip().replace(" ", "-")
    s = re.sub(r"(?u)[^-\w.]", "", s)
    return s
```