Featured

Getting started

Hello, world

Simple web scraper

Large language models (LLMs)

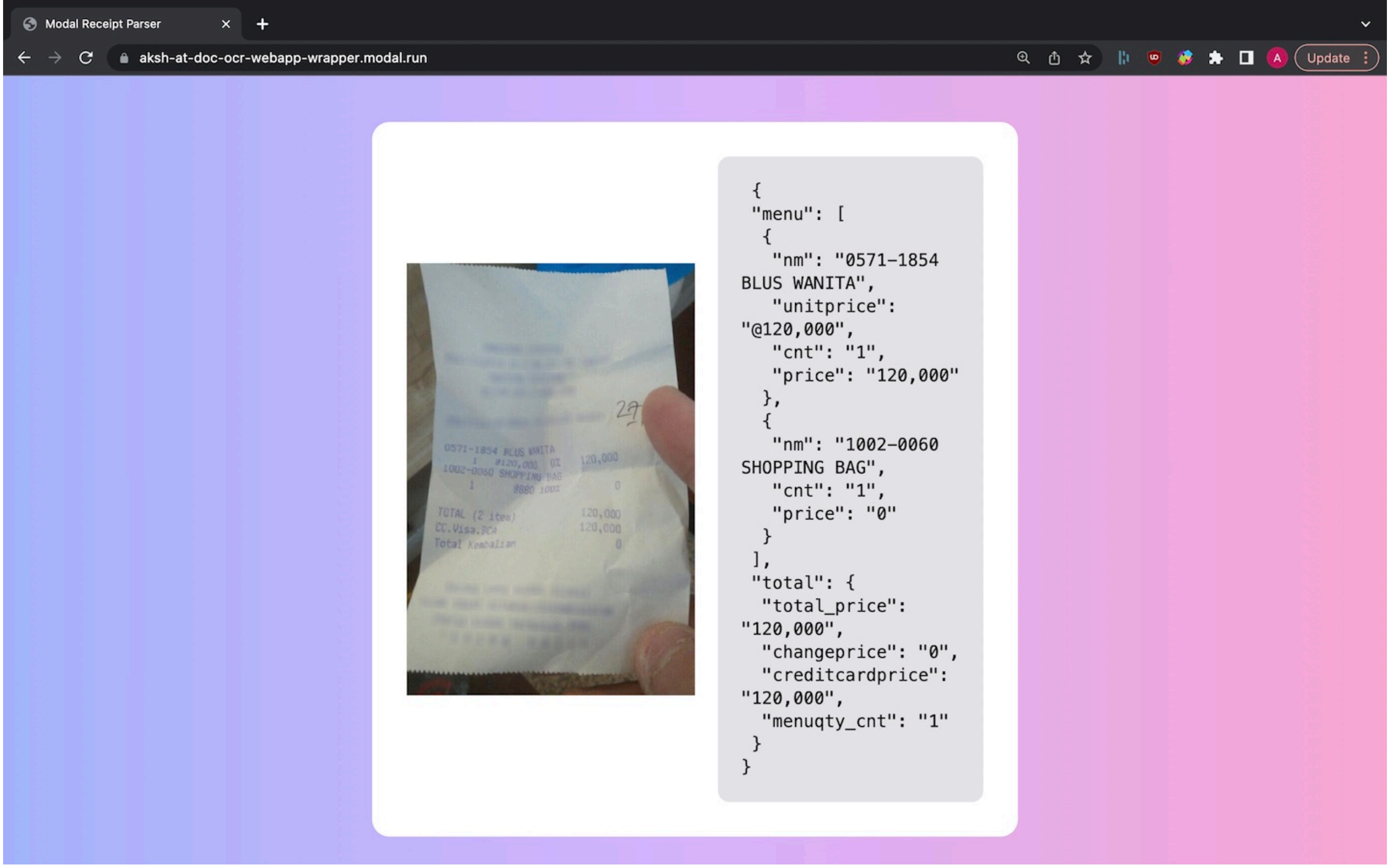
Featured: Serverless TensorRT-LLM

Document OCR job queue

View on GitHub

This tutorial shows you how to use Modal as an infinitely scalable job queue that can service async tasks from a web app. For the purpose of this tutorial, we’ve also built a React + FastAPI web app on Modal that works together with it, but note that you don’t need a web app running on Modal to use this pattern. You can submit async tasks to Modal from any Python application (for example, a regular Django app running on Kubernetes).

Our job queue will handle a single task: running OCR transcription for images. We’ll make use of a pre-trained Document Understanding model using the donut package to accomplish this. Try it out for yourself here.



```
{
"menu": [
 {
  "nm": "0571-1854
BLUS WANITA",
  "unitprice":
"@120,000",
  "cnt": "1",
  "price": "120,000"
 },
 {
  "nm": "1002-0060
SHOPPING BAG",
  "cnt": "1",
  "price": "0"
 }
],
"total": {
 "total_price":
"120,000",
 "changeprice": "0",
 "creditcardprice":
"120,000",
 "menuqty_cnt": "1"
}
}
```

Define an App

Let's first import `modal` and define a `App`. Later, we'll use the name provided for our `App` to find it from our web app, and submit tasks to it.

```
import urllib.request

import modal

app = modal.App(
    "example-doc-ocr-jobs"
)  # Note: prior to April 2024, "app" was called "stub"
```

Model cache

`donut` downloads the weights for pre-trained models to a local directory, if those weights don't already exist. To decrease start-up time, we want this download to happen just once, even across separate function invocations. To accomplish this, we use the `Image.run_function` method, which allows us to run some code at image build time to save the model weights into the image.

```
CACHE_PATH = "/root/model_cache"
MODEL_NAME = "naver-clova-ix/donut-base-finetuned-cord-v2"


def download_model_weights() -> None:
    from huggingface_hub import snapshot_download

    snapshot_download(repo_id=MODEL_NAME, cache_dir=CACHE_PATH)


image = (
    modal.Image.debian_slim(python_version="3.9")
    .pip_install(
        "donut-python==1.0.7",
        "huggingface-hub==0.16.4",
        "transformers==4.21.3",
        "timm==0.5.4",
    )
    .run_function(download_model_weights)
)
```

Handler function

Now let's define our handler function. Using the @app.function() decorator, we set up a Modal Function that uses GPUs, runs on a custom container image, and automatically retries failures up to 3 times.

```
@app.function(
    gpu="any",
    image=image,
    retries=3,
)
def parse_receipt(image: bytes):
    import io

    import torch
    from donut import DonutModel
    from PIL import Image

    # Use donut fine-tuned on an OCR dataset.
    task_prompt = "<s_cord-v2>"
    pretrained_model = DonutModel.from_pretrained(
        MODEL_NAME,
        cache_dir=CACHE_PATH,
    )

    # Initialize model.
    pretrained_model.half()
    device = torch.device("cuda")
    pretrained_model.to(device)
    # Run inference.
    input_img = Image.open(io.BytesIO(image))
    output = pretrained_model.inference(image=input_img, prompt=task_prompt)[
        "predictions"
    ][0]
    print("Result: ", output)

    return output
```

Deploy

Now that we have a function, we can publish it by deploying the app:

```
modal deploy doc_ocr_jobs.py
```

Once it's published, we can look up this function from another Python process and submit tasks to it:

```
fn = modal.Function.lookup("example-doc-ocr-jobs", "parse_receipt")
fn.spawn(my_image)
```

Modal will auto-scale to handle all the tasks queued, and then scale back down to 0 when there's no work left. To see how you could use this from a Python web app, take a look at the receipt parser frontend tutorial.

Run manually

We can also trigger `parse_receipt` manually for easier debugging: `modal run doc_ocr_jobs::app.main` To try it out, you can find some example receipts here.

```
@app.local_entrypoint()
def main():
    from pathlib import Path

    receipt_filename = Path(__file__).parent / "receipt.png"
    if receipt_filename.exists():
        with open(receipt_filename, "rb") as f:
            image = f.read()
    else:
        image = urllib.request.urlopen(
            "https://nwlc.org/wp-content/uploads/2022/01/Brandys-walmart-receipt-8.webp"
        ).read()
    print(parse_receipt.remote(image))
```