The Pragmatic Programmers

Serverless Apps on Cloudflare

Build Solutions, Not Infrastructure

Ashley Peacock

edited by Michael Swaine

Execute AI Models

It blows my mind how easy it is to use AI these days. Just a couple of years ago, AI technologies felt like an alien concept; they were complex and required specialized training for use. Fast forward to 2024, and the rapid development of generative AI has democratized access to AI for everyone, regardless of their background.

For your application, you'll use the Resnet 50 model from Microsoft, an AI model designed for image classification. When presented with an image, the model provides a list of identified objects along with confidence scores. For instance, uploading an image of a cat should result in the model recognizing it as a cat, and in some cases, even specifying the breed.

Let's use some AI, by first updating the environment to expect the AI binding. Open `env.d.ts` and update `CloudflareEnv` to the following:

10-additional.ts
```
interface CloudflareEnv {
  IMAGE_APP_UPLOADS: R2Bucket;
  AI: any;
}
```

Cloudflare's documentation defines the type as any, so I've followed suit. I suspect in future there'll be a defined type.

With the environment now expecting an AI binding to be injected at runtime, you can update the server-side endpoint defined at `src/app/api/files/route.ts` to the following:

10-update-server-side-endpoint.ts
```
import { getRequestContext } from '@cloudflare/next-on-pages'

export const runtime = 'edge';

export async function POST(request: Request) {
  const bucket = getRequestContext().env.IMAGE_APP_UPLOADS;
  const body = await request.formData();
  const files = body.getAll('files');
  const ai = getRequestContext().env.AI;
  let imageAnalysis = [];

  for(let x = 0; x < files.length; x++) {
    const f = files[x] as File;
    const uuid = crypto.randomUUID()

    await bucket.put(uuid, f);

    const blob = await f.arrayBuffer();

    const inputs = {
      image: Array.from(new Uint8Array(blob))
    };

    imageAnalysis.push(
      {
        id: uuid,
        name: f.name,
        analysis: await ai.run('@cf/microsoft/resnet-50', inputs)
      }
    )
  }

  return new Response(JSON.stringify({results: imageAnalysis}), {
    headers: { 'content-type': 'application/json' }
  });
}
```

Whenever you want to use bindings in Pages projects, you need to import getRequestContext to get access to them:

10-update-server-side-endpoint.ts

```
import { getRequestContext } from '@cloudflare/next-on-pages'
```

Then, to call a model, you use getRequestContext.env to access the environment where bindings will be set:

10-update-server-side-endpoint.ts

```
const ai = getRequestContext().env.AI;
let imageAnalysis = [];
```

You also need to create an empty array, which will be populated later in the function with the image analysis results that will be returned from the API. With the bindings retrieved, you can now call the model for each image uploaded:

10-update-server-side-endpoint.ts

```
const blob = await f.arrayBuffer();

const inputs = {
  image: Array.from(new Uint8Array(blob))
};

imageAnalysis.push(
  {
    id: uuid,
    name: f.name,
    analysis: await ai.run('@cf/microsoft/resnet-50', inputs)
  }
)
```

In order to call the Resnet 50 model, the image has to be converted to something a machine can understand. Therefore, you convert the image to a byte array using arrayBuffer, and then that is converted to an array of unsigned 8-bit integers, which is something the model can understand.

With the input prepared, the final line in this section calls the Resnet 50 model, and pushes the results to the imageAnalysis array. When calling any AI model on Cloudflare, you always call the run method, with the first parameter being the model, and the second being the input. If you're unsure how to format the input for a given model, Cloudflare has code examples for every model.[1]

Lastly, the server-side endpoint returns the image analysis results:

10-update-server-side-endpoint.ts

```
return new Response(JSON.stringify({results: imageAnalysis}), {
  headers: { 'content-type': 'application/json' }
});
```

That's all that's required to call an AI model. Cloudflare has made it nearly effortless to make use of such a powerful tool.

Your work isn't done just yet though, as in order to see the model in action, you need to update the frontend to render the image classification results.

Workers AI Limits

As with most things, there are limits. At the time of writing, everyone gets 10,000 neurons free per day. A neuron is how Cloudflare measures inputs across AI models. Roughly speaking, 10,000 neurons can generate 100-200 LLM responses, 500 translations, 500 seconds of speech-to-text audio, 10,000 text classifications, or 1,500 - 15,000 embeddings depending on which models you use.

Additionally, there are per-model limits in place. For example, the Llama 2 model has a limit of 50 requests per minute, Whisper has a limit of 4000 requests per minute, and the one you're using in this chapter, Resnet 50, allows up to 6000 requests per minute.

For a full list of limits by model, check out https://developers.cloudflare.com/workers-ai/platform/limits/.

1. https://developers.cloudflare.com/workers-ai/models/