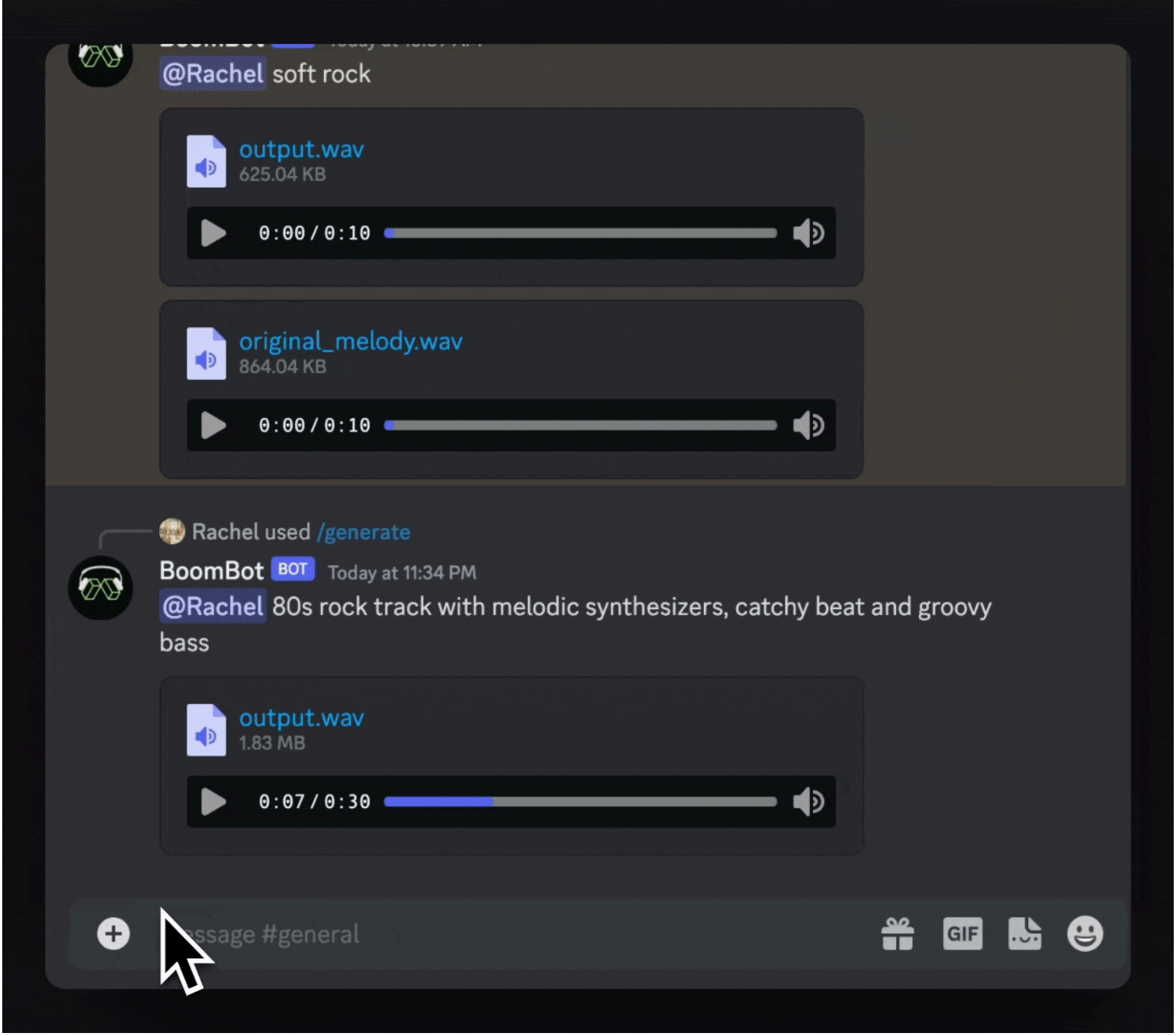
BoomBot: Create your own music samples on Discord

(quick links: try it out on Discord; watch demo with audio; view source code)

MusicGen is the latest milestone language model in conditional music generation, with great results. We wanted a space to easily play around with the model and share our creations, so we created a Discord community featuring BoomBot, an on-demand music sample generator.

You can call BoomBot in the Discord server by simply typing `/generate`, then prompting it with a text description of the music you'd like to create, and even a file of the melody you'd like to condition on, along with other specifiable parameters.



Everything, from the backend API to our React frontend, is deployed serverlessly using Modal, and the code is available here.

Code overview

BoomBot runs the MusicGen model remotely in a GPU-accelerated container — when a user interacts with the Discord bot using the slash command `/generate`, Discord sends a webhook call to Modal, which then generates a music sample using the text description and other user inputs from the command. We also deploy a React single page application as static files into a Modal web endpoint for our web app.

We go into detail into each of these steps below, and provide commands for running each of them individually. To follow along, clone the repo and set up a Discord token for yourself.

Language model

We use Audiocraft, a PyTorch library that provides the code and models for MusicGen, and load both the 3.3B `large` and 1.5B `melody` models to use depending on the user input (large for just text, melody for text and melody). We can install all our dependencies and "bake" both models into our image to avoid downloading our models during inference and take advantage of Modal's incredibly fast cold-start times:

```
def download_models():
    from audiocraft.models import MusicGen

    MusicGen.get_pretrained("large")
    MusicGen.get_pretrained("melody")


image = (
    modal.Image.debian_slim()
    .apt_install("git", "ffmpeg")
    .pip_install(
        "pynacl", # for Discord authentication
        "torch",
        "soundfile",
        "pydub",
        "git+https://github.com/facebookresearch/audiocraft.git",
    )
    .run_function(download_models, gpu="any")
)
app.image = image
```

We then write our model code within Modal's `@app.cls` decorator, with the `generate` function processing the user input and generating audio as bytes that we can save to a file later.

It's useful to test out our model directly before we build out our backend API, so we define a `local_entrypoint`. We can then run our model from the CLI using inputs of our choice:

```
modal run main.py --prompt "soothing jazz" --duration 20 --format "mp3" --melody "https://
```

Discord bot

Now that we've loaded our model and wrote our function, we'd like to trigger it from Discord. We can do this using slash commands — a feature that lets you register a text command on Discord that triggers a custom webhook when a user interacts with it. We handle all our Discord events in a FastAPI app in `bot.py`. Luckily, we can deploy this app easily and serverlessly using Modal's @asgi_app decorator.

Create a Discord app

To connect our model to a Discord bot, we're first going to create an application on the Discord Developer Portal.

1. Go to the Discord Developer Portal and login with your Discord account.
2. On the portal, go to **Applications** and create a new application by clicking **New Application** in the top right next to your profile picture.
3. Copy your Discord application's **Public Key** (in **General Information**) and create a custom Modal secret for it. On Modal's secret creation page, paste the public key as the secret value with the key `DISCORD_PUBLIC_KEY`, then name this secret `boombot-discord-secret`.

Then go back to your application on the Discord Developer Portal, as we need to do a few more things to finish setting up our bot.

Register a Slash Command

Next, we're going to register a command for our Discord app via an HTTP endpoint.

Run the following command in your terminal, replacing the appropriate variable inputs. `BOT_TOKEN` can be found by resetting your token in the application's **Bot** section, and `CLIENT_ID` is the **Application ID** available in **General Information**.

```
BOT_TOKEN='replace_with_bot_token'
CLIENT_ID='replace_with_application_id'
curl -X POST \
-H 'Content-Type: application/json' \
-H "Authorization: Bot $BOT_TOKEN" \
-d '{
  "name":"generate",
  "description":"generate music",
  "options":[
    {
      "name":"prompt",
      "description":"Describe the music you want to generate",
      "type":3,
      "required":true
    },
    {
      "name":"duration",
      "description":"Duration of clip, in seconds (max 60)",
      "type":4,
      "required":false,
      "min_value":2,
      "max_value":60
    },
    {
      "name":"melody",
      "description":"File of melody you\''d like to condition (takes first 30 secs if lon
      "type":11,
```

```
      "required":false
    },
    {
      "name":"format",
      "description":"Desired format of output (default .wav)",
      "type":3,
      "required":false,
      "choices":[
        {
          "name": ".wav",
          "value": "wav"
        },
        {
          "name": ".mp3",
          "value": "mp3"
        }
      ]
    }
  ]
}' "https://discord.com/api/v10/applications/$CLIENT_ID/commands"
```

This will register a Slash Command for your bot named `generate`, and has a few parameters like `prompt`, `duration`, and `format`. More information about the command structure can be found in the Discord docs here.

Deploy a Modal web endpoint

We then create a `POST /generate` endpoint in `bot.py` using FastAPI and Modal's @asgi_app decorator to handle interactions with our Discord app (so that every time a user does a slash command, we can respond to it).

Note that since Discord requires an interaction response within 3 seconds, we use `spawn` to kick off `Audiocraft.generate` as a background task while returning a `defer` message to Discord within the time limit. We then update our response with the results once the model has finished running.

You can deploy this app by running the following command from your root directory:

```
modal deploy src.bot
```

Copy the Modal URL that is printed in the output and go back to your application's **General Information** section on the Discord Developer Portal. Paste the URL, making sure to append the path of your `POST` endpoint, in the **Interactions Endpoint URL** field, then click **Save Changes**. If your endpoint is valid, it will properly save and you can start receiving interactions via this web endpoint.

Finish setting up Discord bot

To start using the Slash Command you just set up, you need to invite the bot to a Discord server. To do so, go to your application's **OAuth2** section on the Discord Developer Portal. Select `applications.commands` as the scope of your bot and copy the invite URL that is generated at the bottom of the page.

Paste this URL in your browser, then select your desired server (create a new server if needed) and click **Authorize**. Now you can open your Discord server and type `/{name of your slash command}` - your bot should be connected and ready for you to use!

React frontend

We also added a simple React frontend to our FastAPI app to serve as a landing page for our Discord server.

We added the frontend to our existing app in `bot.py` by simply using `Mount` to mount our local directory at `/assets` in our container, then instructing FastAPI to serve this static file directory at our root path.

For a more in-depth tutorial on deploying your web app on Modal, refer to our Document OCR web app example.

Run this app on Modal

All the source code for this example can be found on Github.

If you're interested in learning more about Modal, check out our docs and other examples.