```

Title: Cog\*Ify

Story

Problem

Promise / Objective

NO DJ? NO PROBLEM

* Mixing songs because no person should

Outline

Deployment Pipeline

* Gui
* Spotify Playlist link
* YouTube Ids
* Download Wav convert to Mel-Spectrogram
* Model predict on spectrogram
* Extract Best Timestamps
* Use timestamps to get a section of the recorded audio for each wav file
* Convert audio sections to Hertz-Spectrogram
* Calculate cosine similarity between all possible pairs of spectrograms
* Generate order to play the songs using cosine similarity
* Apply crossfade transitions between the songs and stitch them together
* Run combined track through diffusion model and generate art to stitch into music video
* Use gui to play, pause, and stop the mix

Demo

Training Pipeline

* Created Dataset of NCS music and metadata on YouTube channels
* Download Wav convert to Mel-Spectrogram
* Pulled replay graphs from dataset and extract best time stamps
* Train Model on spectrograms and time stamps

Future Improvements

General Overview (while we take questions) and Thank you

* Stay on General Overview Slide when

```

**SCRIPT**

Slide #1: Title

Hi everyone, we are the developers of project Cog\*Ify.

I am Rohan Dugad from Basking Ridge, New Jersey

I am Medha Mittal from Sammamish, Washington.

I am Khoi Nguyen from Northern Virginia

I am Aadi Shah from Sunnyvale, California

I am Aswin Surya from San Jose, California.

We are students of BWSI’s CogWorks 2023.

[Aadi] Slide #2: Story & Problem:

Imagine you have your friends over, and you want to kick off the party. You decide to play music

**[Click for Slide Animation]**

but you want to spend time with your friends, too.

**[Click for Slide Animation]**

So how do you keep the music going? Do you hire a DJ?

**[Click for Slide Animation]**

Can you afford to, every time your friends come over?

[Aadi] Slide #3:

Have no fear: with Cog\*Ify, no DJ is no problem! In making Cog\*Ify, we hoped to create an AI music mixer that maintains the excitement of a party by mixing the most high energy segments of songs together, because no person should!

[Aswin] Slide # : Outline

Before we start, we’ll discuss what we hope to cover today. We’ll cover our deployment process and demo, before discussing our model’s training pipeline and the challenges we faced. Then, we’ll take a look at how we generated a music video for our remixed playlist. Finally, we’ll discuss any future improvements that we could make in Cog\*Ify 2.0.

[Medha] Slide # : Deployment Pipeline [MEDHA]

But how exactly does Cog\*Ify work? How can people use it?

[Medha] Slide # : Graphical User Interface [MEDHA]

Our program starts with a graphical user interface

**[Click for Slide Animation]**

where the user can paste the url link to their spotify playlist at the very top.

**[Click for Slide Animation]**

From there, they can generate their mashup by clicking the query spotify playlist button

**[Click for Slide Animation]**

and then clicking create the mashup.

The song titles and artist names will be displayed in the bottom box of the GUI in the same order as the spotify playlist,

**[Click for Slide Animation]**

and this order will be updated after the mashup is created to show the song order in the mashup.

[Khoi] Slide # : Getting Song Data

Our program takes the user’s playlist link and uses spotify’s official API to fetch the song names and artists for the songs in the playlist. Then, using the YouTube API we get the top search result for the song name and artist name, and then we scrape the song data if it is a no copyright song. Next, the model, using the song data, predicts the best part of the song and the program splices together a mashup of the best song parts.

Slide # \_\_: Generate spectrograms

We found the most optimal format to represent the song audio to be 2D images of spectrograms. So, using the scraped song data, our program downloads the song audio from the links in the descriptions of the NCS videos. Then, the program reads in the audio samples

**[Click for Slide Animation]**

And makes them all a 240-second uniform length. Finally, we convert the audio to a Mel-spectrogram

**[Click for Slide Animation]**

The x axis is time, the y axis is frequency, and color is in decibels where brighter is louder.

Slide #\_\_: Model prediction

The spectrogram is then passed to the model, which processes it and makes a prediction.

**[Click for Slide Animation]**

**[Click for Slide Animation]**

It predicts 240 values, each for a 1s segment of the original audio, that denote the probability that it is in the best part of the song. Using a threshold of 0.5

**[Click for Slide Animation]**

And using the peak

**[Click for Slide Animation]**

The program determines the best interval of the song to use.

**[Click for Slide Animation]**

**[Click for Slide Animation]**

In this example,

**[Click for Slide Animation]**

This is the best interval of the song

**[Click for Slide Animation]**

[Khoi] Slide # : Spectrogram

Our program then converts the best part of the song into a regular spectrogram to extract descriptors for the song.

[Khoi] Slide # : Cosine Similarity

The program then measures the similarity of each song using cosine similarity, which measures the angle between them. A computer represents the data as an array of numbers and we can envision each arrow, or vector, as the program’s representation of a song.

**[Click for Animation]**

As you can see, the pink and the purple song are very similar, while the yellow and pink songs are not very similar. Our program runs a cosine similarity on the end and start of every song to find the smoothest songs to transition to.

Slide # : Determining the Best Order [MEDHA]

To create the best mashup, the songs should be played in an order that enables smooth transitions between songs. To do this, first, a random song is chosen from the playlist to be the first song.

**[Click for Slide Animation]**

The next song is chosen by finding the song with the highest cosine similarity to the first song.

**[Click for Slide Animation]**

This process is repeated for all songs in the playlist while ensuring that the song chosen to go next isn’t already part of the mashup.

**[Click for Slide Animation]**

Next, we add transitions.

Slide # : Adding Transitions Between Songs [MEDHA]

We applied 3 transitions to the mashup to improve its quality. We added a 4 second fade-in to the very first song in the mashup, 2 second crossfades between all consecutive songs, and a 4 second fade-out from the last song in the mashup.

Slide # : Final Product [MEDHA]

Once the mashup is created, the user can play it using the play button on the GUI.

**[Click for Animation]**

There are also pause and restart buttons

**[Click for Animation]**

as well as a slider that allows users to move to other parts of the mix.

**[Click for Animation]**

Additionally, there is a time bar on the bottom that tells the user how long the mashup is and how far they’ve gotten in the mashup.

Slide #\_\_: Demo

So without further ado, here is a demo of Cog\*Ify

Slide #\_\_: Training Pipeline

Now we’ll get into our training pipeline, specifically how the model was made.

Slide #\_\_: Training Pipeline Slide 2

We’ll first cover the dataset, then the spectrograms, extracting time stamps, and finally training

[Aadi] Slide # : Create Dataset

The first step in our training pipeline was building our dataset.

[Aadi] Slide #\_\_: Create Dataset Slide 2

One major difficulty in our project was finding a dataset. We just couldn’t find a dataset large enough for the scope of our project.

So, we decided to create our own. We used the YouTube API to scrape relevant data from the videos of several NCS music channels. From this data, we filtered the dataset to remove videos that weren’t songs, such as instructional videos, and those that were either duplicates, mashups or compilations.

Slide #\_\_: Generate Spectrograms

Next, we generate the spectrograms

Slide #\_\_: Generate Spectrograms Slide 2

Just as we did at deployment time, we download the no copyright audio

**[Click for Animation]**

Read it in as audio samples

**[Click for Animation]**

And convert it to spectrograms

Slide #\_\_: Extracting the Best Time Stamps

Now it’s time to extract the best time stamps

Slide #\_\_: Extracting the Best Time Stamps Slide 2

When playing a YouTube video you might have noticed that if you hover your mouse to scrub through the song, a replay graph appears.

**[Click for Animation]**

As humans we tend to replay our favorite parts of the song the most

**[Click for Animation]**

So, the program scrapes the replay graph from the video in form of 100 data points

**[Click for Animation]**

**[Click for Animation]**

Which we process like we did for the model predictions at deployment time. Using a threshold of 0.8 here,

**[Click for Animation]**

And the peak,

**[Click for Animation]**

The program determines the best part of the song to use

**[Click for Animation]**

**[Click for Animation]**

**[Click for Animation]**

Then, the program scales the values to 0 or 1

**[Click for Animation]**

And correlates them with time stamps depending on the duration of the song. These 240 values, one for each second in four minutes of audio, are the labels associated with the spectrograms.

Slide #\_\_: Training Model

Finally, the actual training of the model

Slide #\_\_: Hyperparameter optimization

Since the input is images, the best model for us is a convolutional neural network. However, being that our model is made from scratch, we still need to know how many layers, how many filters, kernel size, and other variables. So, we tested 50 configurations whose loss and validation loss graphs are visible here. We chose the configuration that yielded the lowest validation loss

Slide #\_\_: Model Architecture & Hyperparameters

The best configuration was firstly three convolutional layers.

**[Click for Animation]**

The first with 4 5x5 filters, the next with 16 3x3 filters, and finally 64 3x3 filters. Now transitioning to the 1D space the layer is flattened and then fed into three dense layers.

**[Click for Animation]**

The first and second with 256 nodes and the last one with 240 nodes for 240 seconds in four minutes. The model was trained for 50 epochs.

[Aswin] Slide #: Music Video Generation

Using the generated mashup of songs, the user can also create a music video with a simple press of a button!

[Aswin] Slide # : Automated Music Video Generation

First, a Stable Diffusion model generates images based on inputted prompts. Then, intermediate frames are created using image interpolation to transition between images. Next, the images are synchronized with the audio file. Finally, the music video is played and downloaded.

[Aswin] Slide # : Music Video Demo

Here’s a sample of the music video generator! The prompts given to the diffusion model were “abstract geometry” and “cosmic landscape.”

**Click to play demo (8 seconds)**

Slide # : Future Improvements [MEDHA]

In the future, we want to integrate the music video generator into the GUI. Additionally, given a music genre, the program should recommend its own playlist to mashup using a song database. And finally, improving the accuracy of our model by using a computer with more memory for higher resolution spectrograms.

[Aswin] Slide #: Recap:

To recap, we covered the problem of not having a DJ which we solved by automating the DJ with Cog\*Ify!

To do this, we generated spectrograms, extracted the timestamps from songs from a Spotify playlist, and trained a model to predict the best parts, before mashing them. Then we deployed this functionality in a user interface and also created a music video generator to play with the song mashup.

Thank you so much to all the instructors and TAs for guiding us throughout this project, and to everyone here for listening! We’re happy to take any questions on our project.

If anyone asks how we get the song data from YouTube we downloaded the songs from the download links in the description by using an api and writing a couple scripts for the rest