There are a few notable projects that I have worked on.

Many git repositories I worked on are private (Georgia Tech class rules + protection of company information), so I can’t fully share them; I have collected a few code samples and uploaded them here.

Code samples include Spotifind, Dungeon Crawler, and a few zavvie projects I worked on (MovieGrub is excluded because of IP agreements with the client)

# **Dungeon Crawler**:

Developed an interactive 2D-game with a group in which a player can defeat monsters, collect items, and navigate rooms. Used JavaFX platform for game development. Applied object oriented programming principles to design the game with handles to test for functionality.

## Monster.java: contains Monster entity class with details such as movement, health, damage, etc. The Monster entity can move around, deal damage to the player, and takes damage/dies.

package dungeoncrawler.entity.monster;

import dungeoncrawler.entity.Player;

import javafx.animation.KeyFrame;

import javafx.animation.KeyValue;

import javafx.animation.Timeline;

//import javafx.animation.Animation;

//import javafx.geometry.Bounds;

//import javafx.scene.control.Alert;

import javafx.scene.image.Image;

import javafx.scene.paint.ImagePattern;

import javafx.scene.shape.Rectangle;

import javafx.scene.paint.Color;

import javafx.scene.layout.Pane;

import javafx.util.Duration;

//import java.security.Key;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.util.Random;

import java.util.Timer;

// In case where you wanna create your own monster,

// just extends this class and implement startMoving

public class Monster extends Rectangle {

private int health;

private int damage = 1;

private boolean alive = true;

private boolean itemDropAvailable;

public Monster(int width, int height, int health, Color color) {

super(width, height, color);

this.health = health;

this.itemDropAvailable = true;

}

/\*\*

\* Continuous function for animating the movement of the monster class.

\*

\* @param pane the pane the monster is in

\*/

public void move(Pane pane) {

if (this.alive && Player.isAlive()) {

Random rand = new Random();

int maxValue = (int) Math.abs(pane.getWidth() - this.getWidth());

int endX = rand.nextInt(maxValue);

int endY = rand.nextInt(maxValue);

double duration = 2.5;

int maxDistance = (int) Math.max(Math.abs(endX - this.getX()),

Math.abs(endY - this.getY()));

if (maxDistance <= 150) {

duration = .8;

} else if (maxDistance <= 275) {

duration = 1.5;

}

KeyValue x = new KeyValue(this.layoutXProperty(), endX);

KeyValue y = new KeyValue(this.layoutYProperty(), endY);

KeyFrame frame = new KeyFrame(Duration.seconds(rand.nextDouble() + duration), x, y);

Timeline timeline = new Timeline(frame);

timeline.setCycleCount(1);

timeline.play();

timeline.setOnFinished(e -> {

move(pane);

});

}

}

public void takeDamage(int damageCount) {

this.health -= damageCount;

if (this.health <= 0) {

Player.killMonster();

this.setVisible(false);

this.alive = false;

} else {

if (this instanceof GreenMonster) {

try {

this.setFill(new ImagePattern(new Image(new FileInputStream(

System.getProperty("user.dir") + "\\res\\greenMonster2.png"))));

this.damageAnimation(this, new ImagePattern(new Image(new FileInputStream(

System.getProperty("user.dir") + "\\res\\greenMonster.png"))));

} catch (FileNotFoundException exception) {

System.out.println("Green monster image not found " + exception);

}

} else if (this instanceof PinkMonster) {

try {

this.setFill(new ImagePattern(new Image(new FileInputStream(

System.getProperty("user.dir") + "\\res\\pinkMonster2.png"))));

this.damageAnimation(this, new ImagePattern(new Image(new FileInputStream(

System.getProperty("user.dir") + "\\res\\pinkMonster.png"))));

} catch (FileNotFoundException exception) {

System.out.println("Pink monster image not found " + exception);

}

} else if (this instanceof YellowMonster) {

try {

this.setFill(new ImagePattern(new Image(new FileInputStream(

System.getProperty("user.dir") + "\\res\\yellowMonster2.png"))));

this.damageAnimation(this, new ImagePattern(new Image(new FileInputStream(

System.getProperty("user.dir") + "\\res\\yellowMonster.png"))));

} catch (FileNotFoundException exception) {

System.out.println("Yellow monster image not found " + exception);

}

} else if (this instanceof DogeMonster) {

try {

this.setFill(new ImagePattern(new Image(new FileInputStream(

System.getProperty("user.dir") + "\\res\\doge2.png"))));

this.damageAnimation(this, new ImagePattern(new Image(new FileInputStream(

System.getProperty("user.dir") + "\\res\\doge.png"))));

} catch (FileNotFoundException exception) {

System.out.println("Doge monster image not found " + exception);

}

}

}

}

public void damageAnimation(Monster monster, ImagePattern img) {

Timer t = new java.util.Timer();

t.schedule(

new java.util.TimerTask() {

@Override

public void run() {

monster.setFill(img);

t.cancel();

}

}, 50);

}

public void attackPlayer(Player player) {

if (this.getBoundsInParent().intersects(player.getBoundsInParent())) {

if (player.getIsAggressive()) {

this.takeDamage(Player.getDamage());

} else {

player.takeDamage(this.damage);

}

}

}

public int getHealth() {

return health;

}

public void setHealth(int health) {

this.health = health;

}

public int getDamage() {

return damage;

}

public void setDamage(int damage) {

this.damage = damage;

}

public void setAlive(boolean alive) {

this.alive = alive;

}

public boolean isAlive() {

return this.alive;

}

public boolean isItemDropAvailable() {

return itemDropAvailable;

}

public void setItemDropAvailable(boolean itemDropAvailable) {

this.itemDropAvailable = itemDropAvailable;

}

}



## Player.java: Player entity class with details like movement, weapons, health, etc. Player can move around, pick up weapons/potions, deal damage to monsters, take damage, and navigates rooms.

package dungeoncrawler.entity;

import dungeoncrawler.Controller;

import dungeoncrawler.entity.potion.Potion;

import dungeoncrawler.entity.potion.HealthPotion;

import dungeoncrawler.entity.potion.AttackPotion;

import dungeoncrawler.entity.potion.ZoomPotion;

import javafx.scene.image.Image;

import javafx.scene.paint.Color;

import javafx.scene.paint.ImagePattern;

import javafx.scene.shape.Rectangle;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

public class Player extends Rectangle {

public static final int ORIGINAL\_HEALTH = 20;

public static final int ORIGINAL\_SPEED = 7;

private static int MONSTERS\_KILLED = 0;

private static int POTIONS\_DRANK = 0;

private static int ITEMS\_PURCHASED = 0;

private static int health = ORIGINAL\_HEALTH;

private static int damageModifier = 0; //Tracks bonuses to player damage

private static int damage;

private static final Weapon[] WEAPON\_INVENTORY = {new Weapon("Shortsword", 1),

new Weapon("Bludgeon", 2),

new Weapon("Greatsword", 3)};

private static final Potion[] POTION\_INVENTORY = {new HealthPotion(),

new AttackPotion(),

new ZoomPotion()};

//alter this to change potion quantity

private static final int[] INVENTORY\_QUANTITY = {0, 0, 0, 1, 1, 1};

private static Weapon currentWeapon;

private Rectangle weapon;

private boolean goNorth;

private boolean goSouth;

private boolean goEast;

private boolean goWest;

private static boolean alive = true;

private boolean isAggressive;

private static int speed = ORIGINAL\_SPEED;

public Player(double x, double y, int width, int height, Weapon startingWeapon) {

super(x, y, width, height);

this.setVisible(true);

this.setFill(Color.RED);

this.isAggressive = true;

this.setId("player");

if (currentWeapon == null) {

currentWeapon = startingWeapon;

}

damage = damageModifier + currentWeapon.getDamage();

}

public Rectangle getWeaponSprite() {

weapon = new Rectangle(this.getWidth(), this.getHeight());

if (Player.getCurrentWeapon().getName().equals("Bludgeon")) {

try {

weapon.setFill(new ImagePattern(new Image(new FileInputStream(

System.getProperty("user.dir") + "\\res\\bludgeon.png"))));

} catch (FileNotFoundException exception) {

System.out.println("Bludgeon image not found " + exception);

}

} else if (Player.getCurrentWeapon().getName().equals("Greatsword")) {

try {

weapon.setFill(new ImagePattern(new Image(new FileInputStream(

System.getProperty("user.dir") + "\\res\\greatsword.png"))));

} catch (FileNotFoundException exception) {

System.out.println("Greatsword image not found " + exception);

}

} else {

try {

weapon.setFill(new ImagePattern(new Image(new FileInputStream(

System.getProperty("user.dir") + "\\res\\shortsword.png"))));

} catch (FileNotFoundException exception) {

System.out.println("Shortsword image not found " + exception);

}

}

weapon.setX(this.getX() + 25);

weapon.setY(this.getY());

return (weapon);

}

public void move(int height, int width) {

int dx = 0;

int dy = 0;

if (goNorth) {

dy = -speed;

}

if (goWest) {

dx = -speed;

}

if (goSouth) {

dy = speed;

}

if (goEast) {

dx = speed;

}

double newX = this.getX() + dx;

double newY = this.getY() + dy;

if (newY < 0) {

newY = 0;

}

if (newX < 0) {

newX = 0;

}

if (newY + this.getHeight() > height) {

newY = height - this.getHeight();

}

if (newX + this.getWidth() > width) {

newX = width - this.getWidth();

}

this.setX(newX);

if (this.weapon != null) {

weapon.setX(newX + 25);

this.setY(newY);

weapon.setY(newY);

}

}

public void takeDamage(int damageCount) {

health -= damageCount;

if (health <= 0) {

alive = false;

}

}

public static void updateWeapon(Weapon newWeapon) {

currentWeapon = newWeapon;

}

public static void resetStats() {

Player.health = ORIGINAL\_HEALTH;

Player.speed = ORIGINAL\_SPEED;

Player.INVENTORY\_QUANTITY[3] = 1;

Player.INVENTORY\_QUANTITY[4] = 1;

Player.INVENTORY\_QUANTITY[5] = 1;

Player.INVENTORY\_QUANTITY[0] = 0;

Player.INVENTORY\_QUANTITY[1] = 0;

Player.INVENTORY\_QUANTITY[2] = 0;

Controller.setGold(100 - 25 \* Controller.getDifficulty().ordinal());

}

public static Weapon[] getWeaponInventory() {

return WEAPON\_INVENTORY;

}

public static Potion[] getPotionInventory() {

return POTION\_INVENTORY;

}

public static int[] getInventoryQuantity() {

return INVENTORY\_QUANTITY;

}

public static Weapon getCurrentWeapon() {

return currentWeapon;

}

public static boolean isAlive() {

return alive;

}

public static void setHealth(int newHealth) {

health = newHealth;

}

public static void setIsAlive(boolean isAlive) {

alive = isAlive;

}

public void setGoNorth(boolean goNorth) {

this.goNorth = goNorth;

}

public void setGoSouth(boolean goSouth) {

this.goSouth = goSouth;

}

public void setGoEast(boolean goEast) {

this.goEast = goEast;

}

public void setGoWest(boolean goWest) {

this.goWest = goWest;

}

public static int getDamage() {

return damage;

}

public static void setDamage(int newDamage) {

damage = newDamage;

}

public void setIsAggressive(boolean isAggressive) {

this.isAggressive = isAggressive;

}

public boolean getIsAggressive() {

return this.isAggressive;

}

public static int getHealth() {

return health;

}

public static int getSpeed() {

return speed;

}

public static void setSpeed(int speed) {

Player.speed = speed;

}

public static int getDamageModifier() {

return damageModifier;

}

public static void setDamageModifier(int newDamageMod) {

damageModifier = newDamageMod;

}

public static void killMonster() {

MONSTERS\_KILLED++;

}

public static int getMonstersKilled() {

return MONSTERS\_KILLED;

}

public static void drinkPotion() {

POTIONS\_DRANK++;

}

public static int getPotionsDrank() {

return POTIONS\_DRANK;

}

public static void purchaseItem() {

ITEMS\_PURCHASED++;

}

public static int getItemsPurchased() {

return ITEMS\_PURCHASED;

}

}



# **Spotifind**

Developed an application called SpotiFind with a group to create a Spotify playlist of similar-sounding songs from a given song. Used Spotify's API to access each song's features, including tempo, key, danceability, etc. Utilized PCA to reduce dimensionality and find the 3 directions that most correlated with feature similarity between songs. To group the songs, implemented three different methods for comparison; DBScan, Naive Bayes, and K-nearest neighbors.

## spotify\_data\_retrieval.py: retrieving songs and their features using Spotify API

import spotipy

from spotipy.oauth2 import SpotifyClientCredentials

import numpy as np

import pandas as pd

cid = ''

secret = ''

client\_credentials\_manager = SpotifyClientCredentials(client\_id = cid, client\_secret=secret)

sp = spotipy.Spotify(client\_credentials\_manager = client\_credentials\_manager)

# Spotify Song URL for comparison

search\_artist = 'example artist'

search\_track = 'example track'

# Arrays to titles and artists

song\_titles = []

song\_artists = []

artist\_ids = []

song\_years = []

song\_ids = []

# Arrays of data features

song\_danceability = []

song\_energy = []

song\_keys = []

song\_loudness = []

song\_modes = []

song\_speechiness = []

song\_acousticness = []

song\_instrumentalness = []

song\_liveness = []

song\_valence = []

song\_tempo = []

song\_duration\_ms = []

song\_genre = []

nulls = []

# Parameters for api retrieval

# number of years back

year\_range = 10

# number of songs

num\_songs = 1000

query\_limit = 50

def find\_features(ids):

song\_features = sp.audio\_features(ids)

for j in range(len(song\_features)):

features = song\_features[j]

if (features == None):

nulls.append((year \* num\_songs) + i + j - 1)

song\_danceability.append(None)

song\_energy.append(None)

song\_keys.append(None)

song\_loudness.append(None)

song\_modes.append(None)

song\_speechiness.append(None)

song\_acousticness.append(None)

song\_instrumentalness.append(None)

song\_liveness.append(None)

song\_valence.append(None)

song\_tempo.append(None)

song\_duration\_ms.append(None)

else:

song\_danceability.append(features['danceability'])

song\_energy.append(features['energy'])

song\_keys.append(features['key'])

song\_loudness.append(features['loudness'])

song\_modes.append(features['mode'])

song\_speechiness.append(features['speechiness'])

song\_acousticness.append(features['acousticness'])

song\_instrumentalness.append(features['instrumentalness'])

song\_liveness.append(features['liveness'])

song\_valence.append(features['valence'])

song\_tempo.append(features['tempo'])

song\_duration\_ms.append(features['duration\_ms'])

# Main loop

for year in range(year\_range):

search\_year = 'year:20' + str(21 - year)

print("Getting songs from", search\_year)

for i in range(0, num\_songs, query\_limit):

track\_results = sp.search(q=search\_year, type='track', limit=query\_limit, offset=i)

tracks = track\_results['tracks']['items']

for j in range(len(tracks)):

track = tracks[j]

song\_titles.append(track['name'])

song\_artists.append(track['artists'][0]['name'])

artist\_ids.append(track['artists'][0]['id'])

song\_years.append(track['album']['release\_date'][0:4])

song\_ids.append(track['id'])

find\_features(song\_ids[year\*num\_songs + i: year\*num\_songs + i +query\_limit])

print("Getting Genres...")

for i in range(0, len(artist\_ids), 20):

curr\_artists = sp.artists(artist\_ids[i:i+20])['artists']

for j in range(len(curr\_artists)):

if (curr\_artists[j]['genres'] == []):

song\_genre.append("none")

else:

# print(curr\_artists[j]['genres'][0])

song\_genre.append(curr\_artists[j]['genres'][0])

# Adds requested song

# song\_results = sp.search(q='artist:' + search\_artist + ' track:' + search\_track, type='track')

# search\_track = song\_results['tracks']['items'][0]

# song\_titles.append(search\_track['name'])

# song\_artists.append(search\_track['artists'][0]['name'])

# song\_years.append(search\_track['album']['release\_date'][0:4])

# song\_ids.append(search\_track['id'])

# find\_features([search\_track['id']])

# data dictionary

data = {'titles': song\_titles, 'artists': song\_artists, 'artist id': artist\_ids,

'year':song\_years, 'ids': song\_ids,

'danceability': song\_danceability, 'energy': song\_energy,

'key': song\_keys, 'loudness': song\_loudness, 'mode':song\_modes,

'speechiness': song\_speechiness, 'acousticness': song\_acousticness,

'instrumentalness': song\_instrumentalness, 'liveness': song\_liveness,

'valence': song\_valence, 'tempo': song\_tempo,

'duration\_ms': song\_duration\_ms,

'genre': song\_genre}

df = pd.DataFrame(data)

df = df.drop(df.index[nulls])

df = df.drop\_duplicates(keep='first')

df = df.dropna(how='all')

df = df.drop\_duplicates(subset=['titles', 'duration\_ms'])

df.to\_csv("spotify\_api.csv")

print("Done!")



## PCA: reduce dimensionalities of features to 5 most important directions

# PCA

import numpy as np

import pandas as pd

from sklearn.decomposition import PCA

df = pd.read\_csv('spotify\_api (1).csv', engine='python')

features = list(df)[5:]

df\_red = df.drop(['Unnamed: 0', 'titles', 'artists', 'year', 'ids'], axis=1)

array = df\_red.to\_numpy()

print(array)

pca = PCA(n\_components=5)

reduced = pca.fit\_transform(array)

#identify most impactful features

print(pca.components\_)

print(pca.components\_.shape)

var = np.square(pca.components\_)

var = np.sum(var, axis=0)

idx = np.argsort(var)

features = [features[i] for i in idx]

print(np.flip(features)) # list of features from most to least important

df['dir 1'] = reduced[:, 0]

df['dir 2'] = reduced[:, 1]

df['dir 3'] = reduced[:, 2]

df['dir 4'] = reduced[:, 3]

df['dir 5'] = reduced[:, 4]

df.to\_csv("spotify\_api\_pca5.csv")

print("PCA Done!")

## KNN: classify songs using 20 nearest neighbors

# K-nearest neighbors

from sklearn.neighbors import NearestNeighbors

from matplotlib import pyplot as plt

df = pd.read\_csv('spotify\_api\_pca.csv', engine='python')

df\_pca = df[['dir 1','dir 2','dir 3']]

print(df\_pca)

dataset = df\_pca.to\_numpy()

neighbors = NearestNeighbors(n\_neighbors=20)

neighbors\_fit = neighbors.fit(dataset)

distances, indices = neighbors\_fit.kneighbors(dataset)

distances = np.sort(distances, axis=0)

distances = distances[4500:,1]

plt.plot(distances)

# 

# **Dialogflow Solution Webhook (zavvie)**

Took in user input of (County, State) from Dialogflow, extracted parameters, passed them into zavvie API, reformatted output, and returned solutions in user’s requested location.

## Cloud Function: Use user input of location to call API and return solutions in location

const functions = require('@google-cloud/functions-framework');

const unirest = require('unirest');

// Here's where you set the entry point, it gets the request from Dialogflow CX

// and it holds the response which will be sent back to Dialogflow

functions.http('solutionhook', (req, res) => {

var output = "";

var userInput = req.body.sessionInfo.parameters.location.original;

console.log(userInput);

var county = userInput.substring(0, userInput.length - 4);

var state = userInput.substring(userInput.length - 2, userInput.length);

var apiResponse = unirest.post("https://api.zavvie.com/v2/api/eligible-programs");

apiResponse.headers({

'zavvie-api-key': 'CHANGE ME',

'content-type' : 'application/json'

});

apiResponse.send({

'client-type': 'seller',

'locations': [{'state': state,'county': county}]

});

apiResponse.end(function(apiRes) {

if(apiRes.error) {

console.log("OOPS! error");

console.log(JSON.stringify(apiRes));

} else {

output += "Solutions available for " + userInput + ":\n";

for (var i in apiRes.body.data) {

if (apiRes.body.data[i] != null) {

output += "Solution: " + JSON.stringify(apiRes.body.data[i]["display\_name"]) + "\n";

for (var j in apiRes.body.data[i]["programs"]) {

output += "- " + "Program provider: " + JSON.stringify(apiRes.body.data[i]["programs"][j]["program\_name"]) + "\n";

}

}

}

console.log("SUCCESS! response: ");

console.log(output);

console.log("full data:");

console.log(JSON.stringify(apiRes.body.data));

}

var response = {

fulfillmentResponse: {

messages: [

{

text: {

text: [

output // this is the message for the agent to return

],

},

},

],

},

sessionInfo: {

parameters:

{

"success": true,

}

}

};

res.status(200).json(response);

});

});



# **Dialogflow Text Embedding Webhook (zavvie)**

Generated text embeddings for FAQs, then set up Matching Engine’s Approximate Nearest Neighbors with Vertex AI to use cosine distance for vector comparison. Passed in user’s FAQ input from Dialogflow, and used cloud function to translate the input to text embedding vector, call Matching Engine endpoint with vector as parameter, and select the closest vector returned from the endpoint. Checked to ensure that the distance is close enough to suggest similar enough meaning, selected the answer associated with the closest stored FAQ, passed the answer through Generative AI to add variation to the wording, and returned the altered answer.

## Cloud Function: Map user input to closest stored FAQ through text embedding and return answer with salt from Generative AI

import functions\_framework

#dict mapping detected intents to answers

answers = {}

@functions\_framework.http

def call\_matching\_engine(request):

"""HTTP Cloud Function.

Args:

request (flask.Request): The request object.

<https://flask.palletsprojects.com/en/1.1.x/api/#incoming-request-data>

Returns:

The response text, or any set of values that can be turned into a

Response object using `make\_response`

<https://flask.palletsprojects.com/en/1.1.x/api/#flask.make\_response>.

"""

import google.cloud.logging

client = google.cloud.logging.Client()

client.setup\_logging()

import logging

import json

import google.cloud.aiplatform

from google.cloud.aiplatform.matching\_engine.matching\_engine\_index\_endpoint import Namespace

from vertexai.preview.language\_models import TextEmbeddingModel

import vertexai

from vertexai.language\_models import TextGenerationModel

request\_json = request.get\_json(silent=True)

request\_args = request.args

logging.info("mic check")

userInput = request\_json['text']

logging.info(userInput)

model = TextEmbeddingModel.from\_pretrained("textembedding-gecko@001")

embeddings = model.get\_embeddings([userInput])[0].values

logging.info(embeddings)

similarity\_engine = google.cloud.aiplatform.MatchingEngineIndexEndpoint(

index\_endpoint\_name = "7444793231671296000",

project = "intern-378317",

location = "us-central1"

)

neighbors = similarity\_engine.match(

deployed\_index\_id="similarity\_engine",

queries=[embeddings],

num\_neighbors=4

)

closest = [{"id": neighbor.id, "distance": neighbor.distance} for neighbor in neighbors[0]]

logging.info(closest)

closestIntent = closest[0]["id"]

closestIntent = closestIntent[0:closestIntent.rindex('-')]

closestDistance = closest[0]["distance"]

logging.info(closestIntent)

logging.info("distance: " + str(closestDistance))

if closestDistance < 0.2:

output = answers[closestIntent]

logging.info(output)

vertexai.init(project="intern-378317", location="us-central1")

parameters = {

"temperature": 0.5,

"max\_output\_tokens": 256,

"top\_p": 0.8,

"top\_k": 40

}

model = TextGenerationModel.from\_pretrained("text-bison@001")

modelResponse = model.predict(

"""Reword the following sentence(s) in a slightly different way:\n""" + output,

\*\*parameters

)

response = {"fulfillment\_response": {"messages": [{"text": {"text": [modelResponse.text]}}]}}

else:

response = {"fulfillment\_response": {"messages": [{"text": {"text": ["Sorry, I'm not sure if I know the answer to that."]}}]}}

return (response)

## Embeddings pipeline: generate embeddings for stored intents

import kfp

import kfp.dsl as dsl

from kfp import compiler

from kfp.dsl import Dataset, Input, Output

from typing import List

@dsl.component(

base\_image="python:3.11",

packages\_to\_install=["google-cloud-aiplatform", "appengine-python-standard"],

)

def generate\_embeddings(markdown\_gcs\_dir: str):

import json

import os

from vertexai.preview.language\_models import TextEmbeddingModel

model = TextEmbeddingModel.from\_pretrained("textembedding-gecko@001")

local\_dir = markdown\_gcs\_dir.replace("gs://", "/gcs/") + "embed-prompts/intents/"

# Loop over files in the GCS directory

data = []

for filename in os.listdir(local\_dir):

print(filename)

temp = os.path.join(local\_dir, filename)

if os.path.isdir(temp):

print(temp + " is dir!")

trainFolder = os.path.join(temp, 'trainingPhrases')

jsonFile = os.path.join(trainFolder, 'en.json')

with open(jsonFile) as f:

fullJson = json.load(f)

for i in range(len(fullJson["trainingPhrases"])):

text = ""

for j in fullJson["trainingPhrases"][i]["parts"]:

text += j["text"]

embeddings = model.get\_embeddings([text])

for embedding in embeddings:

data.append(

json.dumps(

{

"id": filename + "-" + str(i),

"embedding": embedding.values

}

)

)

with open(markdown\_gcs\_dir.replace('gs://', '/gcs/') + "engine/data.json", 'w') as f:

f.write("\n".join(data))

@dsl.pipeline(name="similarity-engine-posts")

def transcript\_extraction():

generate\_embeddings(markdown\_gcs\_dir="gs://[BUCKET NAME HERE]/posts/")

compiler.Compiler().compile(transcript\_extraction, "pipeline.yaml")

## Answer pipeline: create json with answers mapped to intents

import kfp

import kfp.dsl as dsl

from kfp import compiler

from kfp.dsl import Dataset, Input, Output

from typing import List

@dsl.component(

base\_image="python:3.11",

packages\_to\_install=["google-cloud-aiplatform", "appengine-python-standard"],

)

def generate\_answers(markdown\_gcs\_dir: str):

import json

import os

local\_dir = markdown\_gcs\_dir.replace("gs://", "/gcs/") + "embed-prompts/flows/"

# Loop over files in the GCS directory

data = dict()

for filename in os.listdir(local\_dir):

print(filename)

temp = os.path.join(local\_dir, filename)

if os.path.isdir(temp):

print(temp + " is dir!")

jsonFile = os.path.join(temp, filename + '.json')

with open(jsonFile) as f:

fullJson = json.load(f)

for i in range(len(fullJson["transitionRoutes"])):

intent = fullJson["transitionRoutes"][i]["intent"]

answer = ""

if "messages" in fullJson["transitionRoutes"][i]["triggerFulfillment"] and len(fullJson["transitionRoutes"][i]["triggerFulfillment"]["messages"]) > 0:

for j in fullJson["transitionRoutes"][i]["triggerFulfillment"]["messages"][0]["payload"]["richContent"][0]:

if "subtitle" in j:

answer += j["subtitle"] + "\n"

data[intent] = answer

with open(markdown\_gcs\_dir.replace('gs://', '/gcs/') + "engine/answers.json", 'w') as f:

json.dump(data, f)

@dsl.pipeline(name="similarity-engine-posts")

def transcript\_extraction():

generate\_answers(markdown\_gcs\_dir="gs://[BUCKET NAME HERE]/posts/")

compiler.Compiler().compile(transcript\_extraction, "answers.yaml")

