```
import os
from utils import predict_food, fetch_recipe

def main():
    print("đ˝ď¸ Welcome to the Food Classifier & Recipe Finder!")

    while True:
        image_path = input("\nđˇ Enter the path to your food image (or type 'q' to quit):
").strip()
        if image_path.lower() in ['q', 'quit', 'exit']:
            print("đ Goodbye! Thanks for using the app.")
            break

        if not os.path.exists(image_path):
            print("â Error: File does not exist.")
            continue

        try:
            dish = predict_food(image_path)
            print(f"\nđ Predicted Dish: {dish.title()}")

            name, ingredients, instructions = fetch_recipe(dish)
            print(f"\nđ Recipe for {name}:\n")
            print("đ Ingredients:")
            for ing in ingredients:
                print(f"  â˘ {ing}")

            print("\nđ¨âđł Instructions:")
            print(instructions.strip())
        except Exception as e:
            print(f"â ď¸ An error occurred: {e}")

if __name__ == "__main__":
    main()
```

```python
import torch
import pandas as pd
import requests
from PIL import Image
from difflib import get_close_matches
from transformers import (
    AutoProcessor, AutoModelForImageClassification,
    BeitImageProcessor, BeitForImageClassification
)
from collections import Counter




# Constants & Setup
SPOONACULAR_API_KEY = "88f477773a694bca85126abd582af035"
INDIAN_DATASET = "IndianFoodDatasetCSV.csv"
indian_df = pd.read_csv(INDIAN_DATASET)

# Model loading
beit_processor = BeitImageProcessor.from_pretrained("microsoft/beit-large-patch16-224")
beit_model = BeitForImageClassification.from_pretrained("microsoft/beit-large-patch16-224")





# -------------------- Prediction Functions --------------------
def predict_indian(image_path):
    processor = AutoProcessor.from_pretrained("dima806/indian_food_image_detection")
    model =
AutoModelForImageClassification.from_pretrained("dima806/indian_food_image_detection")
    image = Image.open(image_path).convert("RGB")
    inputs = processor(images=image, return_tensors="pt")
    with torch.no_grad():
        logits = model(**inputs).logits
    return model.config.id2label[logits.argmax(-1).item()]

def predict_western(image_path):
    processor = AutoProcessor.from_pretrained("nateraw/food")
    model = AutoModelForImageClassification.from_pretrained("nateraw/food")
    image = Image.open(image_path).convert("RGB")
    inputs = processor(images=image, return_tensors="pt")
    with torch.no_grad():
        logits = model(**inputs).logits
    return model.config.id2label[logits.argmax(-1).item()]

def predict_third(image_path):
    image = Image.open(image_path).convert("RGB")
    inputs = beit_processor(images=image, return_tensors="pt")
    with torch.no_grad():
        outputs = beit_model(**inputs)
    return beit_model.config.id2label[outputs.logits.argmax(-1).item()]

# -------------------- Recipe Retrieval --------------------
def find_recipe_mealdb(dish):
    try:
        url = f"https://www.themealdb.com/api/json/v1/1/search.php?s={dish}"
        res = requests.get(url).json()
        if res['meals']:
            meal = res['meals'][0]
            ingredients = [
                f"{meal[f'strIngredient{i}']} - {meal[f'strMeasure{i}']}"
                for i in range(1, 21) if meal[f'strIngredient{i}']
            ]
            return meal['strMeal'], ingredients, meal['strInstructions']
    except:
        pass
    return None

def find_recipe_csv(dish):
    match = indian_df[indian_df['TranslatedRecipeName'].str.lower() == dish.lower()]
    if not match.empty:
        row = match.iloc[0]
        ingredients = row['TranslatedIngredients'].split(", ")
        return row['TranslatedRecipeName'], ingredients, row['TranslatedInstructions']
    return None

def find_recipe_spoonacular(dish):
    try:
        search = f"https://api.spoonacular.com/recipes/complexSearch?query=
{dish}&number=1&apiKey={SPOONACULAR_API_KEY}"
        search_res = requests.get(search).json()
        if search_res['results']:
            rid = search_res['results'][0]['id']
            detail = f"https://api.spoonacular.com/recipes/{rid}/information?apiKey=
{SPOONACULAR_API_KEY}"
            data = requests.get(detail).json()
            ingredients = [i['original'] for i in data['extendedIngredients']]
            return data['title'], ingredients, data.get('instructions', 'No instructions.')
    except:
        pass
    return None

def closest_match(dish, dataset):
    if dataset == "mealdb":
        # Try exact match first
        result = find_recipe_mealdb(dish)
        if result:
            return result
    elif dataset == "csv":
        # Try exact match
        result = find_recipe_csv(dish)
        if result:
            return result
        # Fallback to fuzzy match
        names = indian_df['TranslatedRecipeName'].dropna().tolist()
        close = get_close_matches(dish.lower(), [n.lower() for n in names], n=1, cutoff=0.6)
        if close:
            return find_recipe_csv(close[0])
    elif dataset == "spoonacular":
        # Try exact match
        result = find_recipe_spoonacular(dish)
        if result:
            return result
    return None



def fetch_recipe(dish):
    for source in ["mealdb", "csv", "spoonacular"]:
        recipe = closest_match(dish, source)
        if recipe:
            return recipe
    return dish, ["No ingredients found."], "No instructions available."



GENERIC_TERMS = {"plate", "dish", "food", "meal", "cuisine"}


def predict_food(image_path):
    indian = predict_indian(image_path)
    western = predict_western(image_path)
    beit_raw = predict_third(image_path)

    print(f"\nđ§  Indian Model         : {indian}")
    print(f"đ Western Model        : {western}")
    print(f"đ Microsoft BEiT       : {beit_raw}")

    # Normalize predictions
    beit = [b.strip().lower() for b in beit_raw.split(",")]
    predictions = [indian.lower(), western.lower()] + beit

    # Filter out generic predictions
    predictions = [p for p in predictions if p not in GENERIC_TERMS]

    # Count frequency
    counts = Counter(predictions)
    sorted_dishes = [dish for dish, _ in counts.most_common()]

    # Try exact match in all sources (prioritized)
    for dish in sorted_dishes:
        for source in ["mealdb", "csv", "spoonacular"]:
            result = closest_match(dish, source)
            if result:
                return result[0]

    # Try fuzzy match
    all_dishes = set(indian_df['TranslatedRecipeName'].dropna().str.lower())
    for dish in sorted_dishes:
        match = get_close_matches(dish, list(all_dishes), n=1, cutoff=0.5)
        if match:
            return match[0]

    # Final fallback to first valid prediction
    return sorted_dishes[0] if sorted_dishes else "unknown"
```