In [1]:

```
!pip install transformers torch pandas scikit-learn tqdm rouge-score -q
```

1. Load and Prepare Data

In [2]:

```
import os
import json
import time
import torch
import pandas as pd
import numpy as np
from tqdm.auto import tqdm
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader
from torch.optim import AdamW
from transformers import BartForConditionalGeneration, BartTokenizer, get_linear_schedul

device = torch.device('mps' if torch.backends.mps.is_available() else 'cpu')
print(f'Using device: {device}')

df = pd.read_csv('AppetIte_Dataset_v1.csv')
print(f'Total records: {len(df)}')
print(f'Categories: {df["category"].value_counts().to_dict()}')
```

```
Using device: mps
Total records: 13501
Categories: {'Indulgent': 10685, 'Healthy': 1437, 'Quick Meals': 1308, 'Family-Friendl
y': 71}
```

In [3]:

```
device = torch.device('mps' if torch.backends.mps.is_available() else 'cpu')
print(f'Using device: {device}')
```

```
Using device: mps
```

In [4]:

```
df = pd.read_csv('AppetIte_Dataset_v1.csv')
print(f'Total records: {len(df)}')
print(f'Categories: {df["category"].value_counts().to_dict()}')
```

```
Total records: 13501
Categories: {'Indulgent': 10685, 'Healthy': 1437, 'Quick Meals': 1308, 'Family-Friendl
y': 71}
```

In [5]:

```
df
```

Out[5]:

	recipe_id	recipe_name	ingredients	instructions	image_path	category	storage_tips	nutrition
0	1	0	['1 (3½–4-lb.) whole chicken', '2¾ tsp. kosher...	pat chicken dry with paper towels, season all ...	miso-butter-roast-chicken-acorn-squash-panzanella	Indulgent	Store ingredients in airtight containers; refr...	

	recipe_id	recipe_name	ingredients	instructions	image_path	category	storage_tips	nutrition
1	2	1	['2 large egg whites', '1 pound new potatoes (...	preheat oven to 400°f and line a rimmed baking...	crispy-salt-and-pepper-potatoes-dan-kluger	Indulgent	Store ingredients in airtight containers; refr...	
2	3	2	['1 cup evaporated milk', '1 cup whole milk', ...	place a rack in middle of oven; preheat to 400...	thanksgiving-mac-and-cheese-erick-williams	Indulgent	Store ingredients in airtight containers; refr...	
3	4	3	['1 (¾- to 1-pound) round italian loaf, cut in...	preheat oven to 350°f with rack in middle. gen...	italian-sausage-and-bread-stuffing-240559	Healthy	Store ingredients in airtight containers; refr...	
4	5	4	['1 teaspoon dark brown sugar', '1 teaspoon ho...	stir together brown sugar and hot water in a c...	newtons-law-apple-bourbon-cocktail	Quick Meals	Store ingredients in airtight containers; refr...	
...	...	...	...	...	...	...	...	
13496	13497	13496	['1 cup all-purpose flour', '2/3 cup unsweeten...	preheat the oven to 350°f. into a bowl sift to...	brownie-pudding-cake-14408	Indulgent	Store ingredients in airtight containers; refr...	
13497	13498	13497	['1 preserved lemon', '1 1/2 pound butternut s...	preheat oven to 475°f. halve lemons and scoop ...	israeli-couscous-with-roasted-butternut-squash...	Indulgent	Store ingredients in airtight containers; refr...	
13498	13499	13498	['leftover katsuo bushi (dried bonito flakes) ...	if using katsuo bushi flakes from package, moi...	rice-with-soy-glazed-bonito-flakes-and-sesame-...	Indulgent	Store ingredients in airtight containers; refr...	
13499	13500	13499	['1 stick (1/2 cup) plus 1 tablespoon unsalted...	melt 1 tablespoon butter in a 12-inch heavy sk...	spanakopita-107344	Indulgent	Store ingredients in airtight containers; refr...	
13500	13501	13500	['12 medium to large fresh poblano chiles (2 1...	lay 4 chiles on their sides on racks of gas bu...	mexican-poblano-spinach-and-black-bean-lasagne...	Indulgent	Store ingredients in airtight containers; refr...	

13501 rows × 8 columns

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13501 entries, 0 to 13500
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   recipe_id       13501 non-null  int64  
 1   recipe_name     13501 non-null  int64  
 2   ingredients     13501 non-null  object 
 3   instructions    13493 non-null  object 
 4   image_path      13501 non-null  object 
 5   category        13501 non-null  object 
 6   storage_tips    13501 non-null  object 
 7   nutrition_score  13501 non-null  float64
dtypes: float64(1), int64(2), object(5)
memory usage: 843.9+ KB
```

In [7]:

```python
def prepare_input_text(row):
    ingredients = row['ingredients'] if pd.notna(row['ingredients']) else 'no ingredient
    category = row['category'] if pd.notna(row['category']) else 'general'
    return f"Generate a {category} recipe using: {ingredients}"
```

In [8]:

```python
def prepare_target_text(row):
    recipe_name = row['recipe_name'] if pd.notna(row['recipe_name']) else 'Delicious Rec
    instructions = row['instructions'] if pd.notna(row['instructions']) else 'Instructio
    return f"Recipe: {recipe_name}. Instructions: {instructions}"
```

In [9]:

```python
df['input_text'] = df.apply(prepare_input_text, axis=1)
df['target_text'] = df.apply(prepare_target_text, axis=1)
```

In [10]:

```python
train_df, val_df = train_test_split(df, test_size=0.15, random_state=42, stratify=df['ca

train_df = train_df.head(100).reset_index(drop=True)
val_df = val_df.head(25).reset_index(drop=True)
```

In [11]:

```python
print(f'Training samples: {len(train_df)}')
print(f'Validation samples: {len(val_df)}')
```

```
Training samples: 100
Validation samples: 25
```

2. Load Pre-trained BART Model (Smaller Version)

In [12]:

```python
model_name = 'facebook/bart-base'
tokenizer = BartTokenizer.from_pretrained(model_name)
model = BartForConditionalGeneration.from_pretrained(model_name).to(device)
```

In [13]:

```python
print(f'Model loaded: {model_name}')
print(f'Number of parameters: {sum(p.numel() for p in model.parameters()):,}')
```

```
Model loaded: facebook/bart-base
Number of parameters: 139,420,416
```

3. Test Pre-trained Model (Before Fine-tuning)

In [14]:

```
sample_inputs = [
    "Generate a Healthy recipe using: chicken, rice, broccoli, garlic",
    "Generate a Quick Meals recipe using: pasta, tomato sauce, basil",
    "Generate an Indulgent recipe using: chocolate, cream, butter"
]
```

In [15]:

```
print('Testing pre-trained model:\n')
model.eval()

for text in sample_inputs:
    inputs = tokenizer(text, return_tensors='pt', max_length=128, truncation=True).to(de

    with torch.no_grad():
        generated_ids = model.generate(
            **inputs,
            max_length=100,
            num_beams=4,
            early_stopping=True,
            temperature=0.8,
            do_sample=False  # Deterministic for testing
        )

    output = tokenizer.decode(generated_ids[0], skip_special_tokens=True)
    print(f'Input: {text}')
    print(f'Output: {output}\n')
```

```
The following generation flags are not valid and may be ignored: ['temperature']. Set `T
RANSFORMERS_VERBOSITY=info` for more details.
Testing pre-trained model:

Input: Generate a Healthy recipe using: chicken, rice, broccoli, garlic
Output: Generate a Healthy recipe using: chicken, rice, broccoli, garlic

Input: Generate a Quick Meals recipe using: pasta, tomato sauce, basil
Output: Generate a Quick Meals recipe using: pasta, tomato sauce, basil

Input: Generate an Indulgent recipe using: chocolate, cream, butter
Output: Generate an Indulgent recipe using: chocolate, cream, butter
```

4. Create PyTorch Dataset

In [16]:

```
class RecipeDataset(Dataset):
    def __init__(self, dataframe, tokenizer, max_input_length=128, max_target_length=200
        self.data = dataframe
        self.tokenizer = tokenizer
        self.max_input_length = max_input_length
        self.max_target_length = max_target_length
    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        input_text = self.data.loc[idx, 'input_text']
        target_text = self.data.loc[idx, 'target_text']

        input_encoding = self.tokenizer(
            input_text,
            max_length=self.max_input_length,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )

        target_encoding = self.tokenizer(
            target_text,
            max_length=self.max_target_length,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )

        labels = target_encoding['input_ids'].squeeze()
        labels[labels == self.tokenizer.pad_token_id] = -100
        return {
            'input_ids': input_encoding['input_ids'].squeeze(),
            'attention_mask': input_encoding['attention_mask'].squeeze(),
            'labels': labels
        }
train_dataset = RecipeDataset(train_df, tokenizer)
val_dataset = RecipeDataset(val_df, tokenizer)
```

In [17]:

```
print(f'Train dataset size: {len(train_dataset)}')
print(f'Validation dataset size: {len(val_dataset)}')
```

```
Train dataset size: 100
Validation dataset size: 25
```

5. Fine-tune Model (Optimized for Speed)

In [18]:

```
BATCH_SIZE = 2
GRADIENT_ACCUMULATION_STEPS = 4
EPOCHS = 3
LEARNING_RATE = 5e-5
WARMUP_STEPS = 10
```

In [19]:

```
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, num_worker
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False, num_workers=0
```

In [20]:

```
optimizer = AdamW(model.parameters(), lr=LEARNING_RATE, weight_decay=0.01)
total_steps = (len(train_loader) // GRADIENT_ACCUMULATION_STEPS) * EPOCHS
```

```
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=WARMUP_STEPS, nu
```

In [21]:

```
print(f'Total training steps: {total_steps}')
print(f'Effective batch size: {BATCH_SIZE * GRADIENT_ACCUMULATION_STEPS}')
```

```
Total training steps: 36
Effective batch size: 8
```

In [22]:

```
def train_epoch(model, train_loader, optimizer, scheduler, device, epoch):
    model.train()
    total_loss = 0
    progress_bar = tqdm(train_loader, desc=f'Epoch {epoch} Training')

    for batch_idx, batch in enumerate(progress_bar):
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)

        outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=label
        loss = outputs.loss / GRADIENT_ACCUMULATION_STEPS
        total_loss += loss.item() * GRADIENT_ACCUMULATION_STEPS
        loss.backward()

        if (batch_idx + 1) % GRADIENT_ACCUMULATION_STEPS == 0:
            optimizer.step()
            scheduler.step()
            optimizer.zero_grad()

        avg_loss = total_loss / (batch_idx + 1)
        progress_bar.set_postfix({'Loss': f'{avg_loss:.4f}'})

        if hasattr(torch, 'mps') and torch.backends.mps.is_available():
            if batch_idx % 10 == 0:
                torch.mps.empty_cache()

    return total_loss / len(train_loader)
```

In [23]:

```
def validate(model, val_loader, device):
    model.eval()
    total_loss = 0
    progress_bar = tqdm(val_loader, desc='Validating')

    with torch.no_grad():
        for batch_idx, batch in enumerate(progress_bar):
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=l
            total_loss += outputs.loss.item()

            progress_bar.set_postfix({'Val Loss': f'{outputs.loss.item():.4f}'})

    return total_loss / len(val_loader)
```

In [24]:

```python
os.makedirs('models', exist_ok=True)
os.makedirs('training_logs', exist_ok=True)

print('Training started...\n')

best_val_loss = float('inf')
training_history = {'train_loss': [], 'val_loss': [], 'epoch_times': []}

for epoch in range(1, EPOCHS + 1):
    start_time = time.time()

    print(f'\nEpoch {epoch}/{EPOCHS}')
    train_loss = train_epoch(model, train_loader, optimizer, scheduler, device, epoch)
    val_loss = validate(model, val_loader, device)

    training_history['train_loss'].append(train_loss)
    training_history['val_loss'].append(val_loss)
    epoch_time = time.time() - start_time
    training_history['epoch_times'].append(epoch_time)

    print(f'Train Loss: {train_loss:.4f}')
    print(f'Validation Loss: {val_loss:.4f}')
    print(f'Epoch Duration: {epoch_time/60:.2f} minutes')

    if val_loss < best_val_loss:
        best_val_loss = val_loss
        print('New best model - saving...')
        model.save_pretrained('models/appetite_bart_best')
        tokenizer.save_pretrained('models/appetite_bart_best')

        model_info = {
            'epoch': epoch,
            'train_loss': train_loss,
            'val_loss': val_loss,
            'training_time': sum(training_history['epoch_times']),
            'date_saved': time.strftime('%Y-%m-%d %H:%M:%S')
        }

        with open('models/appetite_bart_best/training_info.json', 'w') as f:
            json.dump(model_info, f, indent=2)

with open('training_logs/training_history.json', 'w') as f:
    json.dump(training_history, f, indent=2)

print('\nTraining complete!')
```

```
Training started...


Epoch 1/3
Epoch 1 Training:   0%|          | 0/50 [00:00<?, ?it/s]
Validating:   0%|          | 0/13 [00:00<?, ?it/s]
Train Loss: 4.4627
Validation Loss: 3.1086
Epoch Duration: 0.36 minutes
New best model - saving...
/opt/anaconda3/envs/aml/lib/python3.12/site-packages/transformers/modeling_utils.py:391
8: UserWarning: Moving the following attributes in the config to the generation config:
```

```
{'early_stopping': True, 'num_beams': 4, 'no_repeat_ngram_size': 3, 'forced_bos_token_i
d': 0}. You are seeing this warning because you've set generation parameters in the mode
l config, as opposed to in the generation config.
  warnings.warn(
Epoch 2/3
Epoch 2 Training:   0%|          | 0/50 [00:00<?, ?it/s]
Validating:   0%|          | 0/13 [00:00<?, ?it/s]
Train Loss: 3.5052
Validation Loss: 2.9424
Epoch Duration: 0.34 minutes
New best model - saving...

Epoch 3/3
Epoch 3 Training:   0%|          | 0/50 [00:00<?, ?it/s]
Validating:   0%|          | 0/13 [00:00<?, ?it/s]
Train Loss: 3.2148
Validation Loss: 2.8462
Epoch Duration: 0.33 minutes
New best model - saving...

Training complete!
```

6. Load Fine-tuned Model and Test

In [25]:

```python
finetuned_model = BartForConditionalGeneration.from_pretrained('models/appetite_bart_bes
finetuned_tokenizer = BartTokenizer.from_pretrained('models/appetite_bart_best')
finetuned_model = finetuned_model.to(device)
finetuned_model.eval()

print('Fine-tuned model loaded successfully!')

try:
    with open('models/appetite_bart_best/training_info.json', 'r') as f:
        training_info = json.load(f)
    print(f'Best epoch: {training_info["epoch"]}')
    print(f'Validation loss: {training_info["val_loss"]:.4f}')
    print(f'Training time: {training_info["training_time"]/60:.1f} minutes')
except:
    pass
```

```
Fine-tuned model loaded successfully!
Best epoch: 3
Validation loss: 2.8462
Training time: 1.0 minutes
```

In [26]:

```python
test_recipes = [
    "Generate a Healthy recipe using: chicken breast, broccoli, olive oil, garlic, lemon
    "Generate a Quick Meals recipe using: pasta, tomato sauce, basil, mozzarella cheese"
    "Generate an Indulgent recipe using: chocolate, cream, butter, vanilla, eggs",
]

print('Testing fine-tuned model:\n')

for i, test_input in enumerate(test_recipes, 1):
    print(f'Test {i}')
    print(f'Input: {test_input}')
    inputs = finetuned_tokenizer(test_input, return_tensors='pt', max_length=128, trunca

    with torch.no_grad():
        generated_ids = finetuned_model.generate(
            **inputs,
            max_length=150,
            num_beams=5,
            no_repeat_ngram_size=3,
            early_stopping=True,
            temperature=0.8,
            do_sample=True
        )

    ai_recipe = finetuned_tokenizer.decode(generated_ids[0], skip_special_tokens=True)
    print(f'Generated Recipe: {ai_recipe}')
    print('-' * 80)
    print()
```

```
Testing fine-tuned model:

Test 1
Input: Generate a Healthy recipe using: chicken breast, broccoli, olive oil, garlic, lem
on
Generated Recipe: Recipe: 1. Instructions: preheat oven to 350°f. season chicken breast
with salt and pepper. add broccoli, broccoli, and broccoli to a large bowl. season with
garlic and season with salt, pepper, and garlic. cook until tender, about 2 minutes. rem
ove chicken breast from the breast and let cool slightly. stir in broccoli and broccoli.
let cool, stirring occasionally, until the broccoli is tender and tender. add chicken br
east and broccoli and cook until browned, about 3 minutes. add the broccoli and chicken
breast to the chicken breast, then season with olive oil. serve chicken breast in a smal
l bowl over medium-high heat, stirring often, until cooked through, about 10 minutes. tr
ansfer to a plate and
--------------------------------------------------------------------------------

Test 2
Input: Generate a Quick Meals recipe using: pasta, tomato sauce, basil, mozzarella chees
e
Generated Recipe: Recipe: 1125. Instructions: preheat oven to 400°f. season with salt an
d pepper. add mozzarella cheese, basil, and basil to the pasta. toss to coat. add the ba
sil and basil. cover with tomato sauce and season with basil. serve in a large saucepan
over medium-high heat, stirring occasionally, until the sauce is golden brown, about 3 m
inutes. serve on a large plate. toss with remaining basil and tomato sauce. serve with b
asil and toss with basil, tomato sauce, salt, and pepper, tossing occasionally. add basi
l and pepper to the saucepan. serve. garnish with 1/2 cup of basil and sprinkle with sal
t, pepper, and 1/4 cup of mo
--------------------------------------------------------------------------------

Test 3
Input: Generate an Indulgent recipe using: chocolate, cream, butter, vanilla, eggs
Generated Recipe: Recipe: Instructions: preheat oven to 350°f. whisk together chocolate,
butter, vanilla, and eggs in a large bowl. whisk until smooth, about 2 minutes. add egg
s, sugar, and vanilla. stir until smooth. whisk in vanilla, eggs, and chocolate mixture.
add vanilla and eggs and whisk until combined. pour in the chocolate mixture, stirring o
ften, until smooth and smooth. let cool, about 5 minutes. stir in eggs and vanilla, stir
ring occasionally, until the mixture is smooth. add milk and vanilla and stir to combin
e. add chocolate and vanilla mixture and stir until blended. whisk again, stirring const
antly, until combined, about 10 minutes. transfer chocolate mixture to a bowl and pour o
ver the remaining chocolate mixture
```

----

7. Evaluation with ROUGE Scores

In [27]:

```
from rouge_score import rouge_scorer

def simple_evaluation(model, tokenizer, val_df, device, num_samples=10):
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)
    rouge1_scores = []
    rouge2_scores = []
    rougeL_scores = []

    model.eval()
    sample_indices = np.random.choice(len(val_df), min(num_samples, len(val_df)), replac

    print(f'Evaluating on {len(sample_indices)} samples...\n')

    for idx in sample_indices:
        input_text = val_df.iloc[idx]['input_text']
        reference_text = val_df.iloc[idx]['target_text']

        inputs = tokenizer(input_text, return_tensors='pt', max_length=128, truncation=T

        with torch.no_grad():
            generated_ids = model.generate(**inputs, max_length=150, num_beams=4, early_

        prediction = tokenizer.decode(generated_ids[0], skip_special_tokens=True)

        try:
            scores = scorer.score(reference_text, prediction)
            rouge1_scores.append(scores['rouge1'].fmeasure)
            rouge2_scores.append(scores['rouge2'].fmeasure)
            rougeL_scores.append(scores['rougeL'].fmeasure)
        except:
            pass

    results = {
        'ROUGE-1': np.mean(rouge1_scores),
        'ROUGE-2': np.mean(rouge2_scores),
        'ROUGE-L': np.mean(rougeL_scores)
    }

    return results

eval_results = simple_evaluation(finetuned_model, finetuned_tokenizer, val_df, device, n

print('Evaluation Results:')
print(f'ROUGE-1: {eval_results["ROUGE-1"]:.4f}')
print(f'ROUGE-2: {eval_results["ROUGE-2"]:.4f}')
print(f'ROUGE-L: {eval_results["ROUGE-L"]:.4f}')

overall_rouge = (eval_results['ROUGE-1'] + eval_results['ROUGE-2'] + eval_results['ROUGE
print(f'\nOverall ROUGE: {overall_rouge:.4f}')
```

```
Evaluating on 10 samples...
```

```
Evaluation Results:
ROUGE-1: 0.3340
ROUGE-2: 0.0986
ROUGE-L: 0.1897

Overall ROUGE: 0.2074
```

8. Safety Features and Model Card

Allergen Detection

In [28]:

```
import re
from datetime import datetime

ALLERGENS = ['peanut', 'milk', 'egg', 'soy', 'fish', 'shellfish', 'wheat', 'gluten', 'se
```

In [29]:

```
def detect_allergens(text):
    found = [a for a in ALLERGENS if re.search(rf'\b{a}', text.lower())]
    return found
```

In [30]:

```
def safety_check(recipe_text):
    allergens = detect_allergens(recipe_text)
    if allergens:
        print(f'Warning: Contains allergens: {allergens}')

    if any(bad in recipe_text.lower() for bad in ['kill', 'poison', 'suicide']):
        print('Unsafe content detected! Review required.')

    return allergens
```

In [31]:

```
sample_recipe = "Recipe: Peanut Butter Cookies. Instructions: Mix peanut butter, eggs, s
allergens = safety_check(sample_recipe)
print(f'Detected allergens: {allergens}')
```

```
Warning: Contains allergens: ['peanut', 'egg']
Detected allergens: ['peanut', 'egg']
```

Prediction Logging

In [32]:

```
def log_prediction(input_text, output_text, allergens=None):
    os.makedirs('logs', exist_ok=True)
    entry = {
        'timestamp': datetime.now().isoformat(),
        'input': input_text,
        'output': output_text,
        'allergens': allergens or []
    }

    with open('logs/predictions.jsonl', 'a') as f:
        f.write(json.dumps(entry) + '\n')
    print('Logged prediction for monitoring.')
```

```
log_prediction(
    "Generate a Healthy recipe using: chicken, broccoli",
    "Recipe: Grilled Chicken with Broccoli...",
    allergens=[]
)
```

```
Logged prediction for monitoring.
```

Model Card

In [33]:

```
def create_model_card(name='AppetIte-BART', version='v1.0'):
    card = f"""# Model Card: {name}

## Model Information
- **Version:** {version}
- **Base Model:** facebook/bart-base
- **Purpose:** Generate recipes from ingredients
- **Training Data:** Curated AppetIteDataset.csv (100 samples)

## Intended Use
- Input: List of ingredients and desired category (Healthy, Quick Meals, Indulgent, Fami
- Output: Recipe name and cooking instructions

## Limitations & Risks
- May generate recipes containing common allergens
- Limited to patterns seen in training data
- Not a substitute for professional dietary advice

## Mitigation Strategies
- Allergen detection filter implemented
- User feedback collection for improvement
- Manual review for edge cases

## Contact
Project Maintainer: Sharath

## License
Educational use only
"""

    with open('MODEL_CARD.md', 'w') as f:
        f.write(card)
    print('Model Card created: MODEL_CARD.md')

create_model_card()
```

```
Model Card created: MODEL_CARD.md
```

9. Complete Pipeline Function

This function combines everything for easy recipe generation with safety checks:

In [34]:

```
def generate_recipe_with_safety(ingredients, category='Healthy', model=None, tokenizer=N
    if model is None:
```

```python
        model = finetuned_model
    if tokenizer is None:
        tokenizer = finetuned_tokenizer

    input_text = f"Generate a {category} recipe using: {ingredients}"

    inputs = tokenizer(input_text, return_tensors='pt', max_length=128, truncation=True)

    model.eval()
    with torch.no_grad():
        generated_ids = model.generate(
            **inputs,
            max_length=150,
            num_beams=5,
            no_repeat_ngram_size=3,
            early_stopping=True,
            temperature=0.8,
            do_sample=True
        )

    recipe_text = tokenizer.decode(generated_ids[0], skip_special_tokens=True)

    allergens = detect_allergens(recipe_text)

    log_prediction(input_text, recipe_text, allergens)

    result = {
        'input': input_text,
        'recipe': recipe_text,
        'category': category,
        'allergens': allergens,
        'timestamp': datetime.now().isoformat()
    }

    return result

print('\n' + '='*80)
print('COMPLETE RECIPE GENERATION PIPELINE')
print('='*80 + '\n')

test_cases = [
    ('chicken, rice, vegetables', 'Healthy'),
    ('pasta, cheese, tomato sauce', 'Quick Meals'),
    ('chocolate, cream, strawberries', 'Indulgent')
]

for ingredients, category in test_cases:
    result = generate_recipe_with_safety(ingredients, category)

    print(f"Input: {ingredients}")
    print(f"Category: {category}")
    print(f"Generated Recipe: {result['recipe']}")
    if result['allergens']:
        print(f"Allergens: {result['allergens']}")
    print('-' * 80)
    print()
```

```
================================================================================
COMPLETE RECIPE GENERATION PIPELINE
```

```
================================================================================

Logged prediction for monitoring.
Input: chicken, rice, vegetables
Category: Healthy
Generated Recipe: Recipe: 1. Instructions: preheat oven to 350°f. add chicken, rice, and
rice to a large bowl and stir until cooked through. add rice, chicken, and chicken to th
e bowl. stir until rice is tender, about 1/2 cup. cover with foil and let cool, stirring
occasionally, until the chicken is tender. transfer to a plate and serve with rice and c
hicken. refrigerate until ready to serve. transfer chicken to a bowl and serve on a rimm
ed baking sheet. place chicken on a baking sheet and cover with plastic wrap. serve chic
ken and rice in a small bowl over medium-high heat. let cool slightly, stirring often, u
ntil chicken is golden brown, about 2 to 3 minutes.
--------------------------------------------------------------------------------

Logged prediction for monitoring.
Input: pasta, cheese, tomato sauce
Category: Quick Meals
Generated Recipe: Recipe: 1. Instructions: preheat oven to 350°f. add pasta, cheese, and
tomato sauce to a large bowl and mix until smooth. add cheese, tomato sauce, and salt. s
eason with salt and pepper and season with pepper and pepper. serve on a large plate. re
frigerate until cheese is tender, about 15 minutes. serve in a small bowl over medium-hi
gh heat. cover with plastic wrap and refrigerate for at least 30 minutes. add the pasta
and cheese mixture to the pasta mixture. toss to coat with salt, pepper, salt, and peppe
r, stirring often. serve with cheese and pepper in a large saucepan over medium heat, st
irring occasionally, until the pasta is golden brown. serve immediately.
--------------------------------------------------------------------------------

Logged prediction for monitoring.
Input: chocolate, cream, strawberries
Category: Indulgent
Generated Recipe: Recipe: 1. Instructions: preheat oven to 350°f. whisk together chocola
te, strawberries, vanilla, and vanilla until smooth, about 2 minutes. add strawberries,
cream, and strawberries to a bowl and mix until smooth. transfer to a large bowl. cover
with plastic wrap and refrigerate until firm, about 10 minutes. let cool, stirring occas
ionally, to remove excess. stir in strawberries and vanilla. add chocolate and vanilla,
stirring often, and stir until smooth and smooth. add the strawberries and strawberries.
pour in the chocolate mixture. stir until the strawberries are golden brown, about 3 min
utes. transfer the strawberries to the bowl and toss in the remaining strawberries. whis
k in the cream and vanilla mixture. pour the strawberries over the
--------------------------------------------------------------------------------
```