```
# Singlish Emotion Classifier - Complete Pipeline with Explainability
# Install required packages first (run in separate cell):
# !pip install transformers torch shap matplotlib seaborn numpy pandas

import torch
from transformers import AutoTokenizer,
AutoModelForSequenceClassification
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from typing import List, Dict, Tuple
import warnings
warnings.filterwarnings('ignore')


class SinglishEmotionClassifier:
    def __init__(self, model_path: str = "./singlish_model"):
        """Initialize the classifier with model and tokenizer"""
        print("Loading model and tokenizer...")
        self.device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
        print(f"Using device: {self.device}")

        self.tokenizer = AutoTokenizer.from_pretrained(model_path)
        self.model =
AutoModelForSequenceClassification.from_pretrained(model_path)
        self.model.to(self.device)
        self.model.eval()

        num_labels = self.model.config.num_labels
        print(f"Model has {num_labels} output classes")

        print("\n Searching for label mappings in model config...")

        print(f"Config id2label: {getattr(self.model.config,
'id2label', 'Not found')}")
        print(f"Config label2id: {getattr(self.model.config,
'label2id', 'Not found')}")

        import json
        import os
        config_path = os.path.join(model_path, "config.json")
        if os.path.exists(config_path):
            print(f"\nReading {config_path}...")
            with open(config_path, 'r') as f:
                config_json = json.load(f)
                if 'id2label' in config_json:
                    print(f"Found id2label in config.json:
{config_json['id2label']}")
```

```
                if 'label2id' in config_json:
                    print(f"Found label2id in config.json:
{config_json['label2id']}")

        if hasattr(self.model.config, 'id2label') and not all('LABEL_'
in str(v) for v in self.model.config.id2label.values()):

            self.emotions = [self.model.config.id2label[i] for i in
range(num_labels)]
            print(f"\nUsing labels from model config:
{self.emotions}")
        elif os.path.exists(config_path):

            with open(config_path, 'r') as f:
                config_json = json.load(f)
                if 'id2label' in config_json and not all('LABEL_' in
str(v) for v in config_json['id2label'].values()):
                    self.emotions = [config_json['id2label'][str(i)]
for i in range(num_labels)]
                    print(f"\nUsing labels from config.json:
{self.emotions}")
                else:
                    print(f"Using default 7-emotion mapping based on
common emotion datasets:")
                    self.emotions = ['anger', 'fear', 'joy',
'neutral', 'sadness', 'surprise']
                    print(f"Mapping: {self.emotions}")
                    print(f"\nNote: If this mapping is incorrect,
please check your training script")
                    print(f"or provide the correct label mapping.")
        else:
            # Fallback
            self.emotions = ['anger', 'fear', 'joy', 'neutral',
'sadness', 'surprise']
        print(f"\n✓ Model loaded successfully!\n")

    def predict(self, text: str) -> Dict:
        """
        Predict emotion for a single text with confidence scores
        """
        # Tokenize
        inputs = self.tokenizer(
            text,
            return_tensors="pt",
            truncation=True,
            max_length=512,
            padding=True
        )
        inputs = {k: v.to(self.device) for k, v in inputs.items()}
```

```
        # Get predictions
        with torch.no_grad():
            outputs = self.model(**inputs)
            logits = outputs.logits
            probabilities = torch.softmax(logits, dim=-1)[0]

        predicted_idx = torch.argmax(probabilities).item()
        predicted_emotion = self.emotions[predicted_idx]
        confidence = probabilities[predicted_idx].item()

        confidence_scores = {
            emotion: prob.item()
            for emotion, prob in zip(self.emotions, probabilities)
        }

        return {
            'text': text,
            'predicted_emotion': predicted_emotion,
            'confidence': confidence,
            'all_scores': confidence_scores
        }

    def predict_batch(self, texts: List[str]) -> List[Dict]:
        """Predict emotions for multiple texts"""
        results = []
        for text in texts:
            results.append(self.predict(text))
        return results


class AttentionVisualizer:
    def __init__(self, classifier: SinglishEmotionClassifier):
        self.classifier = classifier
        self.model = classifier.model
        self.tokenizer = classifier.tokenizer
        self.device = classifier.device

    def get_attention_weights(self, text: str) -> Tuple:
        """Extract attention weights from the model"""
        # Tokenize
        inputs = self.tokenizer(
            text,
            return_tensors="pt",
            truncation=True,
            max_length=512,
            padding=True
        )
        inputs = {k: v.to(self.device) for k, v in inputs.items()}

        # Get attention weights
```

```
        with torch.no_grad():
            outputs = self.model(**inputs, output_attentions=True)
            attentions = outputs.attentions
            logits = outputs.logits
            probabilities = torch.softmax(logits, dim=-1)[0]

        # Get tokens
        tokens =
self.tokenizer.convert_ids_to_tokens(inputs['input_ids'][0])


        last_layer_attention = attentions[-1][0]
        avg_attention = last_layer_attention.mean(dim=0)


        cls_attention = avg_attention[0].cpu().numpy()

        predicted_idx = torch.argmax(probabilities).item()

        return tokens, cls_attention, predicted_idx,
probabilities.cpu().numpy()

    def visualize_attention(self, text: str, top_k: int = 15):
        """Visualize attention weights as a heatmap"""
        tokens, attention, pred_idx, probs =
self.get_attention_weights(text)
        predicted_emotion = self.classifier.emotions[pred_idx]
        confidence = probs[pred_idx]

        token_attention_pairs = [
            (token, att) for token, att in zip(tokens, attention)
            if token not in ['[CLS]', '[SEP]', '[PAD]']
        ]

        # Sort by attention weight
        token_attention_pairs.sort(key=lambda x: x[1], reverse=True)
        top_tokens = token_attention_pairs[:min(top_k,
len(token_attention_pairs))]

        # Prepare data for plotting
        labels = [t[0].replace('##', '') for t in top_tokens]
        values = [t[1] for t in top_tokens]

        # Create figure
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

        # Plot 1: Attention bar chart
        colors = plt.cm.viridis(np.linspace(0.3, 0.9, len(labels)))
        bars = ax1.barh(range(len(labels)), values, color=colors)
        ax1.set_yticks(range(len(labels)))
```

```
        ax1.set_yticklabels(labels)
        ax1.set_xlabel('Attention Weight', fontsize=12)
        ax1.set_title(f'Token Importance for "{predicted_emotion}"
(conf: {confidence:.2%})',
                      fontsize=14, fontweight='bold')
        ax1.invert_yaxis()

        # Add value labels on bars
        for i, (bar, val) in enumerate(zip(bars, values)):
            ax1.text(val, i, f' {val:.4f}', va='center', fontsize=9)

        emotions = self.classifier.emotions
        emotion_probs = probs
        colors_emotion = ['green' if i == pred_idx else 'gray' for i
in range(len(emotions))]

        bars2 = ax2.bar(emotions, emotion_probs, color=colors_emotion,
alpha=0.7)
        ax2.set_ylabel('Confidence Score', fontsize=12)
        ax2.set_title('Emotion Confidence Distribution', fontsize=14,
fontweight='bold')
        ax2.set_ylim(0, 1)

        # Add value labels on bars
        for bar, val in zip(bars2, emotion_probs):
            height = bar.get_height()
            ax2.text(bar.get_x() + bar.get_width()/2., height,
                    f'{val:.2%}', ha='center', va='bottom',
fontsize=10)

        plt.tight_layout()
        plt.show()

        return top_tokens

class GradientExplainer:
    def __init__(self, classifier: SinglishEmotionClassifier):
        self.classifier = classifier
        self.model = classifier.model
        self.tokenizer = classifier.tokenizer
        self.device = classifier.device

    def compute_integrated_gradients(self, text: str, steps: int = 50)
-> Tuple:
        """
        Compute Integrated Gradients - more accurate attribution than
simple gradients
        This shows which words actually contribute to the prediction
        """
        # Tokenize
```

```
        inputs = self.tokenizer(
            text,
            return_tensors="pt",
            truncation=True,
            max_length=512,
            padding=True
        )
        inputs = {k: v.to(self.device) for k, v in inputs.items()}

        embedding_layer = self.model.get_input_embeddings()

        original_embeddings = embedding_layer(inputs['input_ids'])

        baseline_embeddings = torch.zeros_like(original_embeddings)

        with torch.no_grad():
            outputs = self.model(inputs_embeds=original_embeddings,

attention_mask=inputs['attention_mask'])
            predicted_class = torch.argmax(outputs.logits, dim=-
1).item()

        accumulated_grads = torch.zeros_like(original_embeddings)

        for step in range(steps):
            alpha = (step + 1) / steps
            interpolated_embeddings = baseline_embeddings + alpha *
(original_embeddings - baseline_embeddings)
            interpolated_embeddings =
interpolated_embeddings.detach().requires_grad_(True)

            # Forward pass
            outputs =
self.model(inputs_embeds=interpolated_embeddings,

attention_mask=inputs['attention_mask'])

            self.model.zero_grad()
            if interpolated_embeddings.grad is not None:
                interpolated_embeddings.grad.zero_()

            outputs.logits[0, predicted_class].backward()

            # Accumulate gradients
            accumulated_grads += interpolated_embeddings.grad

        # Average the gradients
        avg_grads = accumulated_grads / steps

        integrated_grads = (original_embeddings - baseline_embeddings)
* avg_grads
```

```
        importance = integrated_grads.sum(dim=-1).squeeze().abs()

        tokens =
self.tokenizer.convert_ids_to_tokens(inputs['input_ids'][0])

        return tokens, importance.detach().cpu().numpy(),
predicted_class

    def compute_gradients(self, text: str) -> Tuple:
        """Compute simple gradient-based importance scores"""
        inputs = self.tokenizer(
            text,
            return_tensors="pt",
            truncation=True,
            max_length=512,
            padding=True
        )
        inputs = {k: v.to(self.device) for k, v in inputs.items()}

        embedding_layer = self.model.get_input_embeddings()
        input_ids = inputs['input_ids'].clone()
        embeddings = embedding_layer(input_ids)
        embeddings = embeddings.detach().requires_grad_(True)

        outputs = self.model(
            inputs_embeds=embeddings,
            attention_mask=inputs['attention_mask']
        )

        logits = outputs.logits
        predicted_class = torch.argmax(logits, dim=-1).item()

        self.model.zero_grad()
        if embeddings.grad is not None:
            embeddings.grad.zero_()

        logits[0, predicted_class].backward()
        gradients = embeddings.grad
        importance = (gradients * embeddings).sum(dim=-
1).squeeze().abs()

        tokens =
self.tokenizer.convert_ids_to_tokens(inputs['input_ids'][0])

        return tokens, importance.detach().cpu().numpy(),
predicted_class

    def visualize_integrated_gradients(self, text: str, top_k: int =
15):
```

```
        """Visualize integrated gradients - BETTER than attention for
understanding predictions"""
        tokens, importance, pred_idx =
self.compute_integrated_gradients(text)
        predicted_emotion = self.classifier.emotions[pred_idx]

        # Filter special tokens
        token_imp_pairs = [
            (token, imp) for token, imp in zip(tokens, importance)
            if token not in ['[CLS]', '[SEP]', '[PAD]']
        ]

        # Sort by importance
        token_imp_pairs.sort(key=lambda x: x[1], reverse=True)
        top_tokens = token_imp_pairs[:min(top_k,
len(token_imp_pairs))]

        labels = [t[0].replace('##', '') for t in top_tokens]
        values = [t[1] for t in top_tokens]

        # Normalize for better visualization
        values = np.array(values)
        if values.max() > 0:
            values = (values - values.min()) / (values.max() -
values.min() + 1e-8)

        plt.figure(figsize=(14, 7))
        colors = plt.cm.RdYlGn(values)  # Red (low) to Green (high)
        bars = plt.barh(range(len(labels)), values, color=colors,
edgecolor='black', linewidth=0.5)
        plt.yticks(range(len(labels)), labels, fontsize=11)
        plt.xlabel('Importance Score (Integrated Gradients)',
fontsize=13, fontweight='bold')
        plt.title(f'□ Word Contribution to "{predicted_emotion}"
Prediction\n(Higher = More Important for Decision)',
                  fontsize=15, fontweight='bold', pad=20)
        plt.gca().invert_yaxis()

        # Add value labels
        for i, (bar, val) in enumerate(zip(bars, values)):
            plt.text(val + 0.02, i, f'{val:.3f}', va='center',
fontsize=10, fontweight='bold')

        # Add grid
        plt.grid(axis='x', alpha=0.3, linestyle='--')
        plt.tight_layout()
        plt.show()

        return top_tokens
```

```
    def visualize_gradients(self, text: str, top_k: int = 15):
        """Visualize simple gradient-based importance"""
        tokens, importance, pred_idx = self.compute_gradients(text)
        predicted_emotion = self.classifier.emotions[pred_idx]

        token_imp_pairs = [
            (token, imp) for token, imp in zip(tokens, importance)
            if token not in ['[CLS]', '[SEP]', '[PAD]']
        ]

        token_imp_pairs.sort(key=lambda x: x[1], reverse=True)
        top_tokens = token_imp_pairs[:min(top_k,
len(token_imp_pairs))]

        labels = [t[0].replace('##', '') for t in top_tokens]
        values = [t[1] for t in top_tokens]

        values = np.array(values)
        if values.max() > 0:
            values = (values - values.min()) / (values.max() -
values.min() + 1e-8)

        plt.figure(figsize=(12, 6))
        colors = plt.cm.plasma(values)
        bars = plt.barh(range(len(labels)), values, color=colors)
        plt.yticks(range(len(labels)), labels)
        plt.xlabel('Normalized Importance Score', fontsize=12)
        plt.title(f'Gradient-Based Token Importance for
"{predicted_emotion}"',
                  fontsize=14, fontweight='bold')
        plt.gca().invert_yaxis()

        for i, (bar, val) in enumerate(zip(bars, values)):
            plt.text(val, i, f' {val:.3f}', va='center', fontsize=9)

        plt.tight_layout()
        plt.show()


class NarrativeExplainer:
    def __init__(self, classifier: SinglishEmotionClassifier,
                 attention_viz: AttentionVisualizer):
        self.classifier = classifier
        self.attention_viz = attention_viz

    def generate_explanation(self, text: str) -> str:
        """Generate a narrative explanation of the prediction"""
        # Get prediction
        result = self.classifier.predict(text)
```

```
        # Get attention weights
        tokens, attention, pred_idx, probs =
self.attention_viz.get_attention_weights(text)

        # Filter and sort tokens by attention
        token_attention_pairs = [
            (token, att) for token, att in zip(tokens, attention)
            if token not in ['[CLS]', '[SEP]', '[PAD]']
        ]
        token_attention_pairs.sort(key=lambda x: x[1], reverse=True)

        top_3_tokens = [t[0].replace('##', '') for t in
token_attention_pairs[:3]]

        # Get top 2 emotions
        sorted_emotions = sorted(
            result['all_scores'].items(),
            key=lambda x: x[1],
            reverse=True
        )

        # Generate narrative
        narrative = f"""
═══════════════════════════════════════════════════════
 EXPLANATION FOR: "{text}"
═══════════════════════════════════════════════════════

 PREDICTION:
   Emotion: {result['predicted_emotion'].upper()}
   Confidence: {result['confidence']:.2%}

 HOW THE MODEL THINKS:

   The model classified this text as "{result['predicted_emotion']}"
with
   {result['confidence']:.1%} confidence. Here's why:

   1. KEY INDICATORS:
      The model paid most attention to these words:
      • "{top_3_tokens[0]}" (strongest signal)
      • "{top_3_tokens[1]}"
      • "{top_3_tokens[2]}"

   2. DECISION PROCESS:
      Primary emotion: {sorted_emotions[0][0]} ({sorted_emotions[0]
[1]:.1%})
      Secondary emotion: {sorted_emotions[1][0]} ({sorted_emotions[1]
[1]:.1%})

      The model was {"very confident" if result['confidence'] > 0.7
```

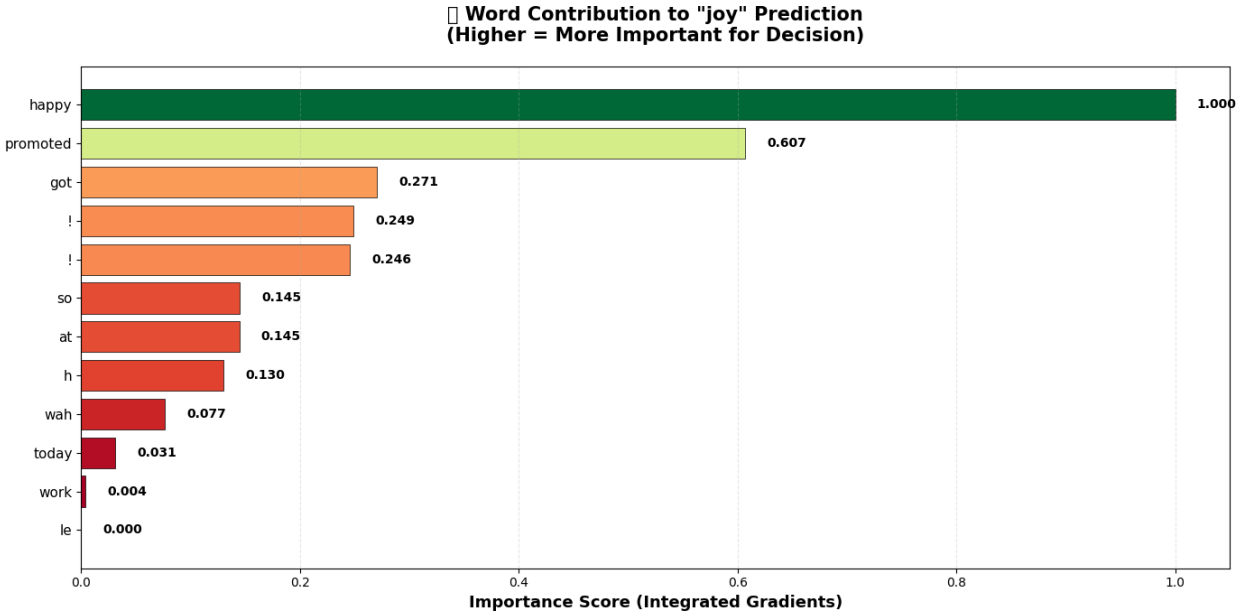
```
else "moderately confident" if result['confidence'] > 0.5 else "less
confident"}
      in this prediction, suggesting {"clear emotional signals" if
result['confidence'] > 0.7 else "mixed emotional cues"}.

   3. ALTERNATIVE INTERPRETATIONS:
      {f"The model also considered '{sorted_emotions[1][0]}'
({sorted_emotions[1][1]:.1%})" if sorted_emotions[1][1] > 0.15 else
"No strong alternative emotions detected."}

═══════════════════════════════════════════════════════════
"""
        return narrative


def main():
    # Initialize classifier
    classifier = SinglishEmotionClassifier("./singlish_model")

    # Test cases - Singlish examples
    test_cases = [
        "Wah so happy today leh! Got promoted at work!",
        "Cb this guy really piss me off sia",
        "Haiz feeling very sad... my dog passed away",
        "Alamak! You scared me lah!",
        "Wah lau this exam confirm fail already",
        "Shiok ah! Weekend finally here!",
        "Just eating lunch now lor, nothing special",
        "Why you like that?! So angry with you!",
        "Surprised sia, never expect you to come",
        "Sian half day, nothing to do at home"
    ]

    print("="*80)
    print("TESTING SINGLISH EMOTION CLASSIFIER")
    print("="*80)
    print()

    # Run predictions
    results = classifier.predict_batch(test_cases)

    table_data = []
    for r in results:
        row = {
            'Text': r['text'][:50] + '...' if len(r['text']) > 50 else
r['text'],
            'Predicted Emotion': r['predicted_emotion'],
            'Confidence': f"{r['confidence']:.2%}"
        }
        # Add all emotion scores dynamically
```

```
        for emotion, score in r['all_scores'].items():
            row[emotion.capitalize()] = f"{score:.3f}"
        table_data.append(row)

    results_df = pd.DataFrame(table_data)

    print(results_df.to_string(index=False))
    print("\n" + "="*80 + "\n")

    # Initialize explainability tools
    attention_viz = AttentionVisualizer(classifier)
    gradient_exp = GradientExplainer(classifier)
    narrative_exp = NarrativeExplainer(classifier, attention_viz)

    print("🔍 DETAILED EXPLAINABILITY ANALYSIS")
    print("="*80)

    for i, text in enumerate(test_cases[:3]):
        print(f"\nEXAMPLE {i+1}")
        print(narrative_exp.generate_explanation(text))

        print("\nINTEGRATED GRADIENTS (Most Accurate - Shows Real Word
Importance):")
        gradient_exp.visualize_integrated_gradients(text, top_k=12)

        print("\nAttention Visualization (Shows What Model Looks
At):")
        attention_viz.visualize_attention(text, top_k=10)

        print("\n" + "-"*80 + "\n")

    return classifier, attention_viz, gradient_exp, narrative_exp

# Run the pipeline
if __name__ == "__main__":
    classifier, attention_viz, gradient_exp, narrative_exp = main()

    # Example: Analyze a custom text
    print("\n" + "="*80)
    print("🎯 Try your own text!")
    print("="*80)
    custom_text = "Wah today really damn shiok leh!"
    print(f"\nAnalyzing: '{custom_text}'")
    print(narrative_exp.generate_explanation(custom_text))
    print("\n📊 Word Importance (Integrated Gradients):")
    gradient_exp.visualize_integrated_gradients(custom_text)
    print("\n👁 Attention Pattern:")
    attention_viz.visualize_attention(custom_text)
```

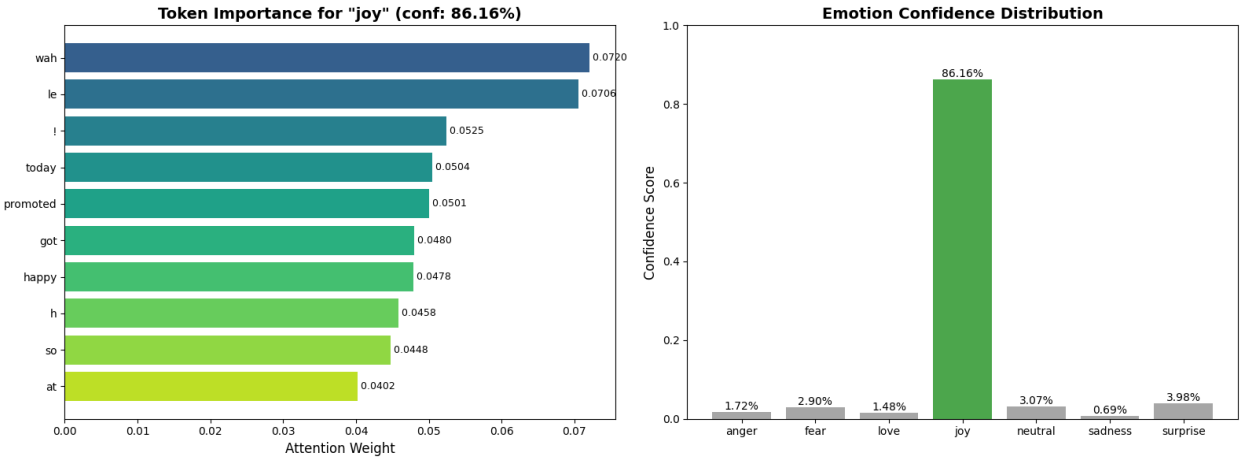
```
Loading model and tokenizer...
Using device: cuda
```

```
Model has 7 output classes

 Searching for label mappings in model config...
Config id2label: {0: 'LABEL_0', 1: 'LABEL_1', 2: 'LABEL_2', 3:
'LABEL_3', 4: 'LABEL_4', 5: 'LABEL_5', 6: 'LABEL_6'}
Config label2id: {'LABEL_0': 0, 'LABEL_1': 1, 'LABEL_2': 2, 'LABEL_3':
3, 'LABEL_4': 4, 'LABEL_5': 5, 'LABEL_6': 6}

 Reading ./singlish_model/config.json...
Found id2label in config.json: {'0': 'LABEL_0', '1': 'LABEL_1', '2':
'LABEL_2', '3': 'LABEL_3', '4': 'LABEL_4', '5': 'LABEL_5', '6':
'LABEL_6'}
Found label2id in config.json: {'LABEL_0': 0, 'LABEL_1': 1, 'LABEL_2':
2, 'LABEL_3': 3, 'LABEL_4': 4, 'LABEL_5': 5, 'LABEL_6': 6}

⚠  Only generic labels (LABEL_X) found in config
Using default 7-emotion mapping based on common emotion datasets:
Mapping: ['anger', 'fear', 'love', 'joy', 'neutral', 'sadness',
'surprise']

Note: If this mapping is incorrect, please check your training script
or provide the correct label mapping.

✓ Model loaded successfully!

================================================================================
TESTING SINGLISH EMOTION CLASSIFIER
================================================================================

                                        Text Predicted Emotion
Confidence Anger  Fear  Love   Joy Neutral Sadness Surprise
Wah so happy today leh! Got promoted at work!               joy
86.16% 0.017 0.029 0.015 0.862   0.031   0.007    0.040
           Cb this guy really piss me off sia             anger
81.64% 0.816 0.021 0.024 0.012   0.021   0.083    0.022
  Haiz feeling very sad... my dog passed away           sadness
37.84% 0.085 0.115 0.083 0.029   0.076   0.378    0.234
                   Alamak! You scared me lah!              fear
87.13% 0.016 0.871 0.027 0.030   0.017   0.016    0.022
       Wah lau this exam confirm fail already           sadness
49.66% 0.055 0.035 0.051 0.016   0.024   0.497    0.322
              Shiok ah! Weekend finally here!               joy
56.27% 0.021 0.024 0.039 0.563   0.036   0.014    0.304
   Just eating lunch now lor, nothing special           neutral
81.89% 0.014 0.018 0.022 0.052   0.819   0.047    0.029
       Why you like that?! So angry with you!             anger
83.62% 0.836 0.017 0.025 0.017   0.022   0.058    0.026
      Surprised sia, never expect you to come          surprise
```

```
86.58% 0.016 0.011 0.045 0.032   0.013   0.018    0.866
         Sian half day, nothing to do at home           sadness
45.02% 0.124 0.045 0.019 0.014   0.330   0.450    0.019

================================================================================
==========

 DETAILED EXPLAINABILITY ANALYSIS
================================================================================
==========

 EXAMPLE 1

──────────────────────────────────────────────────────────────────────
 EXPLANATION FOR: "Wah so happy today leh! Got promoted at work!"
──────────────────────────────────────────────────────────────────────

 PREDICTION:
   Emotion: JOY
   Confidence: 86.16%

 HOW THE MODEL THINKS:

   The model classified this text as "joy" with
   86.2% confidence. Here's why:

   1. KEY INDICATORS:
      The model paid most attention to these words:
      • "wah" (strongest signal)
      • "le"
      • "!"

   2. DECISION PROCESS:
      Primary emotion: joy (86.2%)
      Secondary emotion: surprise (4.0%)

      The model was very confident
      in this prediction, suggesting clear emotional signals.

   3. ALTERNATIVE INTERPRETATIONS:
      No strong alternative emotions detected.

──────────────────────────────────────────────────────────────────────


 INTEGRATED GRADIENTS (Most Accurate - Shows Real Word Importance):
```



🎨 Attention Visualization (Shows What Model Looks At):



🔍 EXAMPLE 2

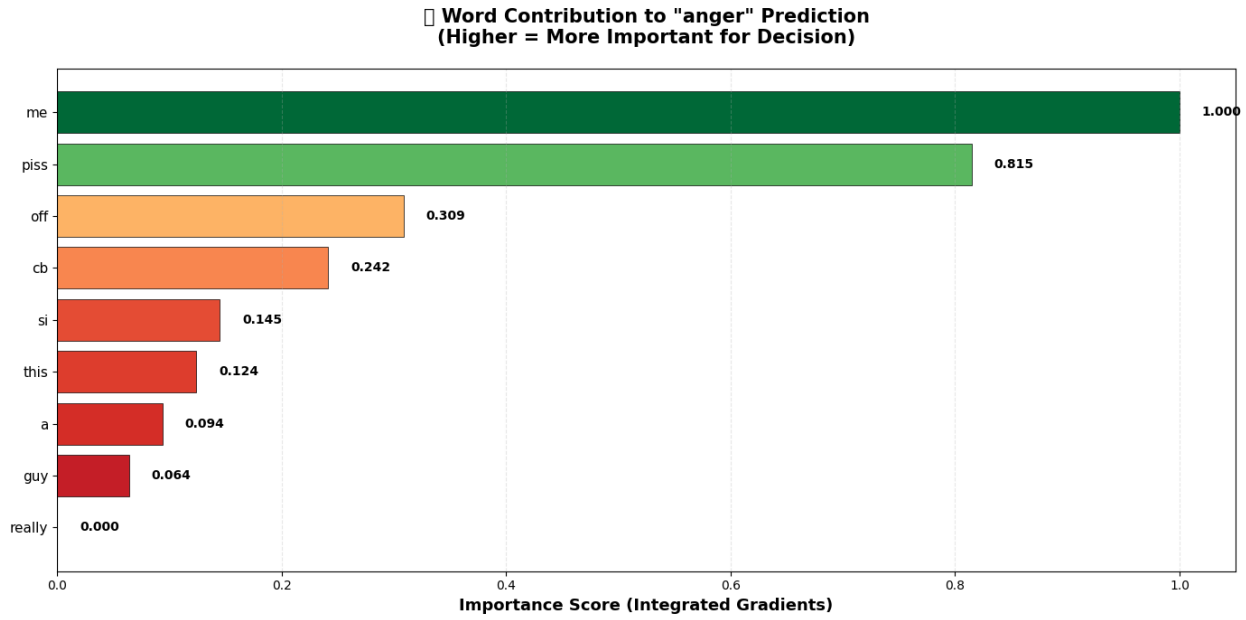
═══

🔍 EXPLANATION FOR: "Cb this guy really piss me off sia"

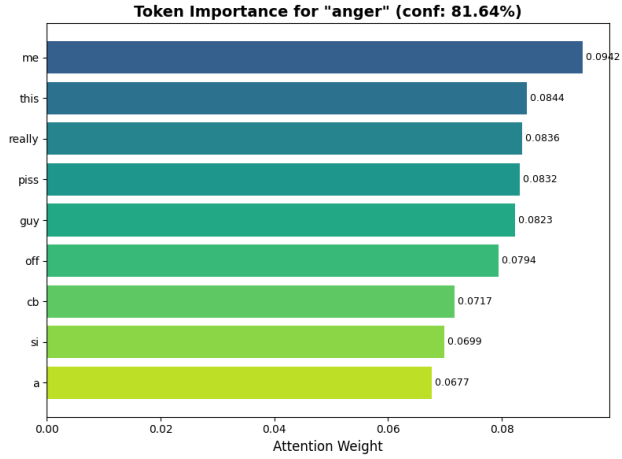
═══

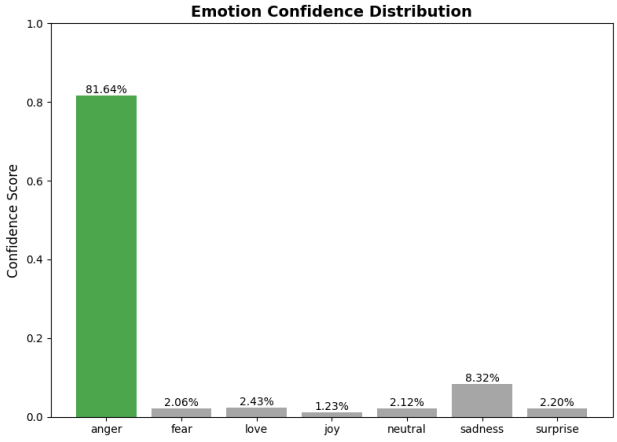
🎯 PREDICTION:
   Emotion: ANGER

```
   Confidence: 81.64%

 HOW THE MODEL THINKS:

   The model classified this text as "anger" with
   81.6% confidence. Here's why:

   1. KEY INDICATORS:
      The model paid most attention to these words:
      • "me" (strongest signal)
      • "this"
      • "really"

   2. DECISION PROCESS:
      Primary emotion: anger (81.6%)
      Secondary emotion: sadness (8.3%)

      The model was very confident
      in this prediction, suggesting clear emotional signals.

   3. ALTERNATIVE INTERPRETATIONS:
      No strong alternative emotions detected.

═══════════════════════════════════════════════════════════

 INTEGRATED GRADIENTS (Most Accurate - Shows Real Word Importance):
```




```
 Attention Visualization (Shows What Model Looks At):
```

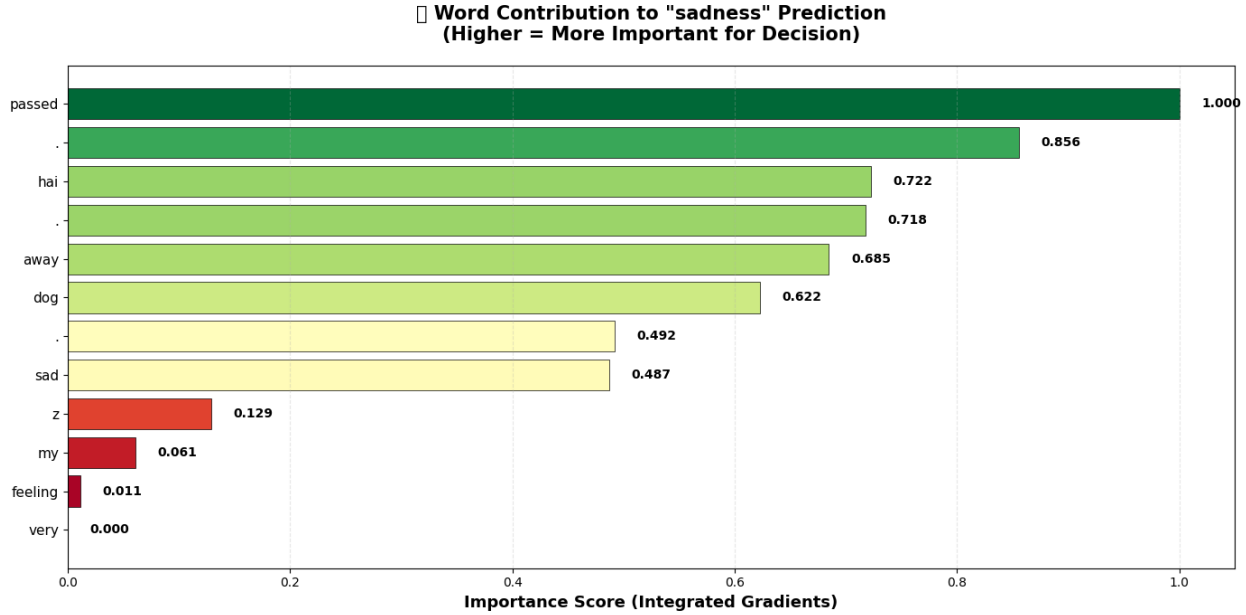




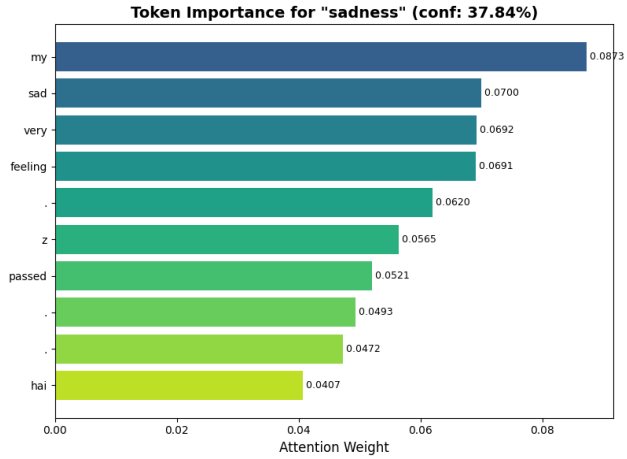

```
------------------------------------------------------------------------------
----------

 EXAMPLE 3

══════════════════════════════════════════════════════════════════════
 EXPLANATION FOR: "Haiz feeling very sad... my dog passed away"
══════════════════════════════════════════════════════════════════════

 PREDICTION:
   Emotion: SADNESS
   Confidence: 37.84%

 HOW THE MODEL THINKS:

   The model classified this text as "sadness" with
   37.8% confidence. Here's why:

   1. KEY INDICATORS:
      The model paid most attention to these words:
      • "my" (strongest signal)
      • "sad"
      • "very"

   2. DECISION PROCESS:
      Primary emotion: sadness (37.8%)
      Secondary emotion: surprise (23.4%)

      The model was less confident
      in this prediction, suggesting mixed emotional cues.

   3. ALTERNATIVE INTERPRETATIONS:
      The model also considered 'surprise' (23.4%)
```

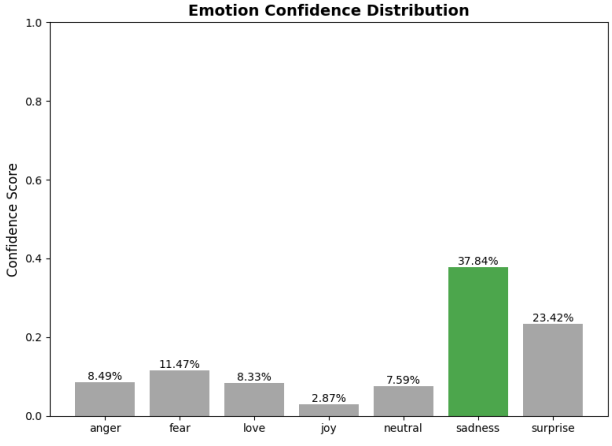
```
```

🔬 INTEGRATED GRADIENTS (Most Accurate - Shows Real Word Importance):



🎯 Attention Visualization (Shows What Model Looks At):

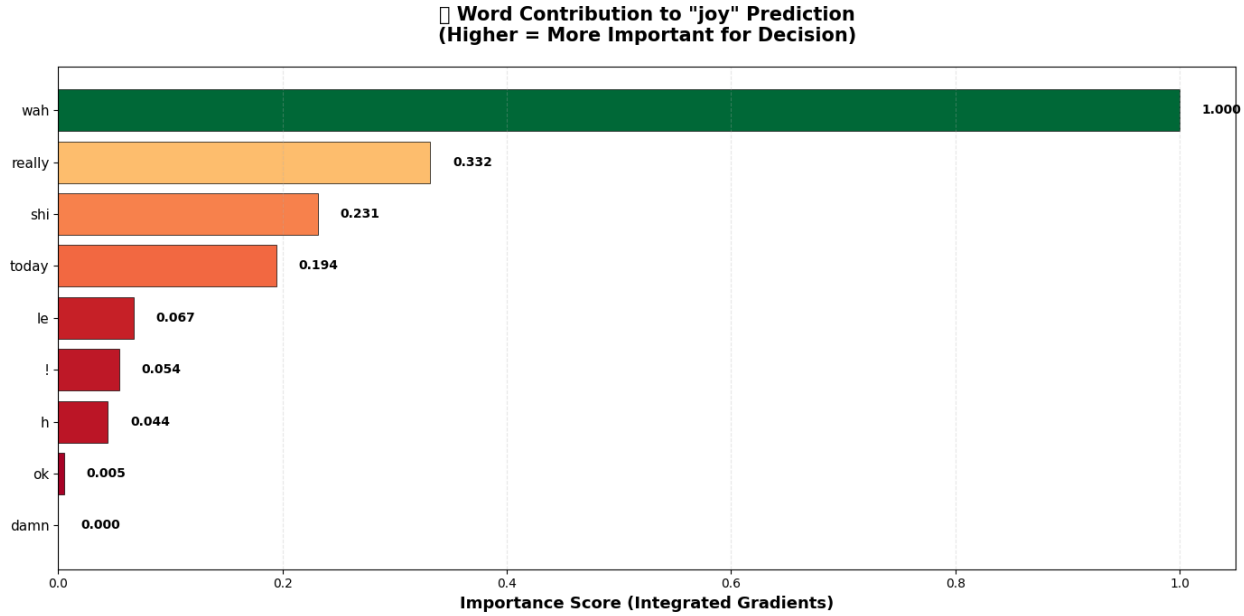




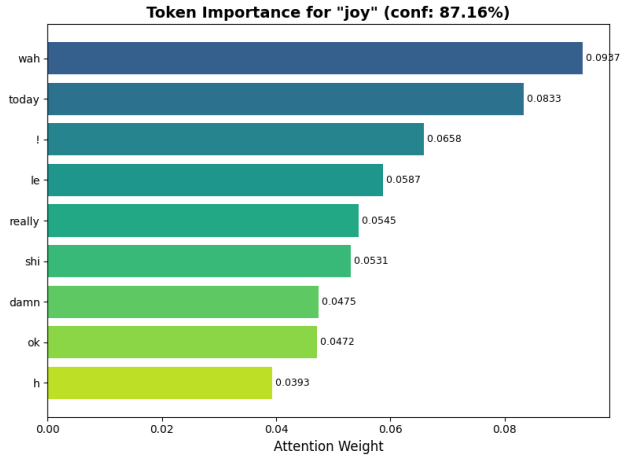

```
----------------------------------------------------------------------------------
```

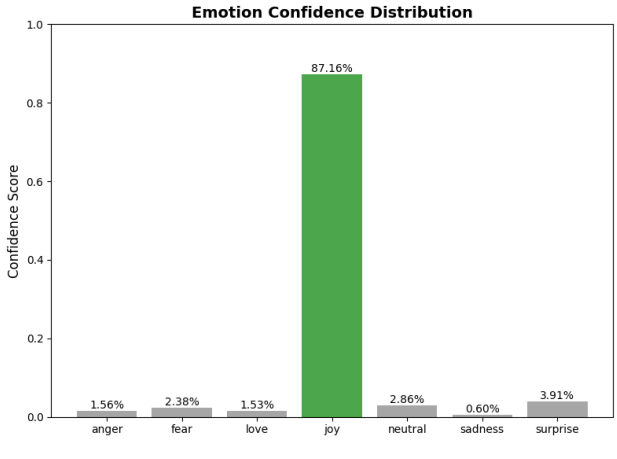
```
==================================================================================
```

```
 Try your own text!
================================================================================

Analyzing: 'Wah today really damn shiok leh!'

──────────────────────────────────────────────────────────────────────
 EXPLANATION FOR: "Wah today really damn shiok leh!"
──────────────────────────────────────────────────────────────────────

 PREDICTION:
   Emotion: JOY
   Confidence: 87.16%

 HOW THE MODEL THINKS:

   The model classified this text as "joy" with
   87.2% confidence. Here's why:

   1. KEY INDICATORS:
      The model paid most attention to these words:
      • "wah" (strongest signal)
      • "today"
      • "!"

   2. DECISION PROCESS:
      Primary emotion: joy (87.2%)
      Secondary emotion: surprise (3.9%)

      The model was very confident
      in this prediction, suggesting clear emotional signals.

   3. ALTERNATIVE INTERPRETATIONS:
      No strong alternative emotions detected.

──────────────────────────────────────────────────────────────────────


 Word Importance (Integrated Gradients):
```



Attention Pattern:






```
!pip install shap transformers torch numpy pandas scipy

Requirement already satisfied: shap in /usr/local/lib/python3.12/dist-
packages (0.50.0)
Requirement already satisfied: transformers in
/usr/local/lib/python3.12/dist-packages (4.57.2)
Requirement already satisfied: torch in
/usr/local/lib/python3.12/dist-packages (2.9.0+cu126)
Requirement already satisfied: numpy in
/usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: pandas in
/usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: scipy in
```

```
/usr/local/lib/python3.12/dist-packages (1.16.3)
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.12/dist-packages (from shap) (1.6.1)
Requirement already satisfied: tqdm>=4.27.0 in
/usr/local/lib/python3.12/dist-packages (from shap) (4.67.1)
Requirement already satisfied: packaging>20.9 in
/usr/local/lib/python3.12/dist-packages (from shap) (25.0)
Requirement already satisfied: slicer==0.0.8 in
/usr/local/lib/python3.12/dist-packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in
/usr/local/lib/python3.12/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in
/usr/local/lib/python3.12/dist-packages (from shap) (3.1.2)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.12/dist-packages (from shap) (4.15.0)
Requirement already satisfied: filelock in
/usr/local/lib/python3.12/dist-packages (from transformers) (3.20.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in
/usr/local/lib/python3.12/dist-packages (from transformers) (0.36.0)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.12/dist-packages (from transformers) (6.0.3)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.12/dist-packages (from transformers)
(2025.11.3)
Requirement already satisfied: requests in
/usr/local/lib/python3.12/dist-packages (from transformers) (2.32.4)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in
/usr/local/lib/python3.12/dist-packages (from transformers) (0.22.1)
Requirement already satisfied: safetensors>=0.4.3 in
/usr/local/lib/python3.12/dist-packages (from transformers) (0.7.0)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.12/dist-packages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in
/usr/local/lib/python3.12/dist-packages (from torch) (1.14.0)
Requirement already satisfied: networkx>=2.5.1 in
/usr/local/lib/python3.12/dist-packages (from torch) (3.6)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.12/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec>=0.8.5 in
/usr/local/lib/python3.12/dist-packages (from torch) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in
/usr/local/lib/python3.12/dist-packages (from torch) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in
```

```
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in
/usr/local/lib/python3.12/dist-packages (from torch) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in
/usr/local/lib/python3.12/dist-packages (from torch) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in
/usr/local/lib/python3.12/dist-packages (from torch) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in
/usr/local/lib/python3.12/dist-packages (from torch) (2.27.5)
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in
/usr/local/lib/python3.12/dist-packages (from torch) (3.3.20)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in
/usr/local/lib/python3.12/dist-packages (from torch) (1.11.1.6)
Requirement already satisfied: triton==3.5.0 in
/usr/local/lib/python3.12/dist-packages (from torch) (3.5.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub<1.0,>=0.34.0->transformers) (1.2.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.12/dist-packages (from numba>=0.54->shap)
(0.43.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2-
>pandas) (1.17.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch)
(1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2->torch) (3.0.3)
Requirement already satisfied: charset_normalizer<4,>=2 in
/usr/local/lib/python3.12/dist-packages (from requests->transformers)
(3.4.4)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.12/dist-packages (from requests->transformers)
(3.11)
```

```
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.12/dist-packages (from requests->transformers)
(2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.12/dist-packages (from requests->transformers)
(2025.11.12)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn->shap)
(1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn->shap)
(3.6.0)

import torch
import shap
import numpy as np
import pandas as pd
import scipy.special
from transformers import AutoTokenizer,
AutoModelForSequenceClassification
from transformers import pipeline


class SinglishReasoningExplainer:
    def __init__(self, model_path="./singlish_model"):
        """
        Initializes the model and the SHAP explainer.
        """
        print("□ Loading Model & Tokenizer...")
        self.device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")

        # Load Model & Tokenizer
        self.tokenizer = AutoTokenizer.from_pretrained(model_path)
        self.model =
AutoModelForSequenceClassification.from_pretrained(model_path)
        self.model.to(self.device)
        self.model.eval()

        # Get Labels
        self.id2label = self.model.config.id2label
        self.label2id = self.model.config.label2id
        self.labels = list(self.id2label.values())
        print(f"□ Model Loaded. Labels: {self.labels}")

        self.pipe = pipeline(
            "text-classification",
            model=self.model,
            tokenizer=self.tokenizer,
            return_all_scores=True,
```

```
            device=0 if torch.cuda.is_available() else -1
        )

        print("🔍 Initializing SHAP Explainer (this takes a
moment)...")
        self.explainer = shap.Explainer(self.pipe)
        print("✅ Explainer Ready.")

    def analyze_text(self, text):
        """
        Performs the SHAP analysis on a single text string.
        """
        # SHAP calculation
        shap_values = self.explainer([text])
        return shap_values

    def generate_narrative(self, text, shap_values):
        """
        Generates a 'Reasoning Model' style narrative based on SHAP
scores.
        """
        prediction_output = self.pipe(text)[0]
        prediction_output = sorted(prediction_output, key=lambda x:
x['score'], reverse=True)
        top_emotion = prediction_output[0]['label']
        top_conf = prediction_output[0]['score']

        token_values = shap_values[0].values[:,
self.model.config.label2id[top_emotion]]
        token_names = shap_values[0].data

        word_impacts = []
        for word, score in zip(token_names, token_values):
            if word.strip() == "": continue
            word_impacts.append((word.strip(), score))

        word_impacts.sort(key=lambda x: x[1], reverse=True)

        drivers = [w for w in word_impacts if w[1] > 0]
        resistors = [w for w in word_impacts if w[1] < 0]

        narrative = f"""
    **Singlish Reasoning Engine**
    ---
    **Input:** "{text}"
    **Prediction:** {top_emotion.upper()} (Confidence: {top_conf:.1%})

    **Reasoning Analysis:**
    The model has identified this text as **{top_emotion}**. Here is
the linguistic breakdown:
```

```
    1.  **Primary Triggers:** The strongest emotional signal comes
from the word **"{drivers[0][0]}"**.
        {f'This is supported by **"{drivers[1][0]}"**.' if
len(drivers) > 1 else ''}
        These words strongly push the probability toward
{top_emotion}.

    2.  **Contextual Nuance:**
        """

        if len(resistors) > 0 and abs(resistors[-1][1]) > 0.05:
            worst_resistor = resistors[-1]
            narrative += f"Although the overall sentiment is
{top_emotion}, the presence of **\"{worst_resistor[0]}\"** actually
works against this prediction. The model sees this word as
conflicting, but the strength of \"{drivers[0][0]}\" overrides it."
        else:
            narrative += "The sentence structure is consistent. There
are no significant conflicting words reducing the confidence."

        narrative += "\n\n**Impact Factor:**"
        narrative += f"\n• \"{drivers[0][0]}\": +{drivers[0][1]:.3f}
impact"
        if len(drivers) > 1: narrative += f"\n• \"{drivers[1][0]}\": +
{drivers[1][1]:.3f} impact"

        return narrative

    def visualize(self, shap_values):
        """
        Plots the SHAP text plot.
        """
        shap.plots.text(shap_values)

reasoning_engine = SinglishReasoningExplainer("./singlish_model")

text1 = "Cb this guy really piss me off sia"
print("\nAnalyzing Case 1...")
shap_values1 = reasoning_engine.analyze_text(text1)
print(reasoning_engine.generate_narrative(text1, shap_values1))
reasoning_engine.visualize(shap_values1)

text2 = "Wah so happy meh?"
print("\nAnalyzing Case 2...")
shap_values2 = reasoning_engine.analyze_text(text2)
print(reasoning_engine.generate_narrative(text2, shap_values2))
reasoning_engine.visualize(shap_values2)
```

```
 Loading Model & Tokenizer...
```

```
Device set to use cuda:0

 Model Loaded. Labels: ['LABEL_0', 'LABEL_1', 'LABEL_2', 'LABEL_3',
'LABEL_4', 'LABEL_5', 'LABEL_6']
 Initializing SHAP Explainer (this takes a moment)...
 Explainer Ready.

Analyzing Case 1...

/usr/local/lib/python3.12/dist-packages/transformers/pipelines/
text_classification.py:111: UserWarning: `return_all_scores` is now
deprecated,  if want a similar functionality use `top_k=None` instead
of `return_all_scores=True` or `top_k=1` instead of
`return_all_scores=False`.
  warnings.warn(


    **Singlish Reasoning Engine**
    ──────────────────────────────────────────────────
    **Input:** "Cb this guy really piss me off sia"
    **Prediction:** LABEL_0 (Confidence: 81.6%)

    **Reasoning Analysis:**
    The model has identified this text as **LABEL_0**. Here is the
linguistic breakdown:

    1.  **Primary Triggers:** The strongest emotional signal comes
from the word **"piss"**.
        This is supported by **"guy"**.
        These words strongly push the probability toward LABEL_0.

    2.  **Contextual Nuance:**
        Although the overall sentiment is LABEL_0, the presence of
**"really"** actually works against this prediction. The model sees
this word as conflicting, but the strength of "piss" overrides it.

**Impact Factor:**
• "piss": +0.569 impact
• "guy": +0.132 impact

<IPython.core.display.HTML object>


Analyzing Case 2...

    **Singlish Reasoning Engine**
    ──────────────────────────────────────────────────
    **Input:** "Wah so happy meh?"
    **Prediction:** LABEL_3 (Confidence: 84.9%)

    **Reasoning Analysis:**
```

```
    The model has identified this text as **LABEL_3**. Here is the
linguistic breakdown:

    1.  **Primary Triggers:** The strongest emotional signal comes
from the word **"happy"**.
        This is supported by **"Wah"**.
        These words strongly push the probability toward LABEL_3.

    2.  **Contextual Nuance:**
        Although the overall sentiment is LABEL_3, the presence of
**"me"** actually works against this prediction. The model sees this
word as conflicting, but the strength of "happy" overrides it.

**Impact Factor:**
• "happy": +0.428 impact
• "Wah": +0.255 impact

<IPython.core.display.HTML object>

import torch
import shap
import numpy as np
import pandas as pd
import scipy.special
from transformers import AutoTokenizer,
AutoModelForSequenceClassification
from transformers import pipeline


class SinglishReasoningExplainer:
    def __init__(self, model_path="./singlish_model"):
        """
        Initializes the model and the SHAP explainer.
        """
        print("□ Loading Model & Tokenizer...")
        self.device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")

        self.tokenizer = AutoTokenizer.from_pretrained(model_path)
        self.model =
AutoModelForSequenceClassification.from_pretrained(model_path)
        self.model.to(self.device)
        self.model.eval()

        self.id2label = self.model.config.id2label
        self.label2id = self.model.config.label2id

        self.emotion_mapping = {
            'LABEL_0': 'anger',
            'LABEL_1': 'fear',
```

```
            'LABEL_3': 'joy',
            'LABEL_4': 'neutral',
            'LABEL_5': 'sadness',
            'LABEL_6': 'surprise'
        }

        self.labels = [self.emotion_mapping.get(label, label) for
label in self.id2label.values()]
        print(f"□ Model Loaded. Labels: {self.labels}")

        self.pipe = pipeline(
            "text-classification",
            model=self.model,
            tokenizer=self.tokenizer,
            return_all_scores=True,
            device=0 if torch.cuda.is_available() else -1
        )

        print("□ Initializing SHAP Explainer (this takes a
moment)...")
        self.explainer = shap.Explainer(self.pipe)
        print("□ Explainer Ready.")

    def analyze_text(self, text):
        """
        Performs the SHAP analysis on a single text string.
        """
        shap_values = self.explainer([text])
        return shap_values

    def generate_narrative(self, text, shap_values):
        """
        Generates a 'Reasoning Model' style narrative based on SHAP
scores.
        """
        prediction_output = self.pipe(text)[0]
        # Sort by score
        prediction_output = sorted(prediction_output, key=lambda x:
x['score'], reverse=True)
        top_emotion_label = prediction_output[0]['label']
        top_conf = prediction_output[0]['score']

        top_emotion = self.emotion_mapping.get(top_emotion_label,
top_emotion_label)

        token_values = shap_values[0].values[:,
self.model.config.label2id[top_emotion_label]]
        token_names = shap_values[0].data

        word_impacts = []
```

```
        for word, score in zip(token_names, token_values):
            if word.strip() == "": continue
            word_impacts.append((word.strip(), score))

        word_impacts.sort(key=lambda x: x[1], reverse=True)
        drivers = [w for w in word_impacts if w[1] > 0]
        resistors = [w for w in word_impacts if w[1] < 0]

        narrative = f"""
 **Singlish Reasoning Engine**
```

```
**Input:** "{text}"
**Prediction:** {top_emotion.upper()} (Confidence: {top_conf:.1%})

**Reasoning Analysis:**
The model has identified this text as **{top_emotion}**. Here is the
linguistic breakdown:

1.  **Primary Triggers:** The strongest emotional signal comes from
the word **"{drivers[0][0]}"**.
    {f'This is supported by **"{drivers[1][0]}"**.' if len(drivers) >
1 else ''}
    These words strongly push the probability toward {top_emotion}.

2.  **Contextual Nuance:**
    """

        # Logic for mixed sentiments
        if len(resistors) > 0 and abs(resistors[-1][1]) > 0.05:
            worst_resistor = resistors[-1]
            narrative += f"Although the overall sentiment is
{top_emotion}, the presence of **\"{worst_resistor[0]}\"** actually
works against this prediction. The model sees this word as
conflicting, but the strength of \"{drivers[0][0]}\" overrides it."
        else:
            narrative += "The sentence structure is consistent. There
are no significant conflicting words reducing the confidence."

        narrative += "\n\n**Impact Factor:**"
        narrative += f"\n• \"{drivers[0][0]}\": +{drivers[0][1]:.3f}
impact"
        if len(drivers) > 1: narrative += f"\n• \"{drivers[1][0]}\": +
{drivers[1][1]:.3f} impact"

        return narrative

    def visualize(self, shap_values):
        """
        Plots the SHAP text plot.
        """
```

```
        shap.plots.text(shap_values)


# Initialize
reasoning_engine = SinglishReasoningExplainer("./singlish_model")

test_cases = [
    ("Anger", "Cb this guy really piss me off sia, knn so angry!"),
    ("Fear", "I takut I lose my job if this target cannot hit one."),
    ("Joy", "Shiok ah! So happy today, got promotion and bonus! Best
day ever!"),
    ("Neutral", "Just going home now lor, nothing special today."),
    ("Sadness", "Haiz so sad lah... my dog died yesterday, heart pain
sia."),
    ("Surprise", "Walao eh! Cannot believe you came back! So shocked
sia!")
]

print("\n" + "="*80)
print("TESTING ALL EMOTIONS WITH SINGLISH EXAMPLES")
print("="*80)

for i, (emotion_name, text) in enumerate(test_cases, 1):
    print(f"\n{'='*80}")
    print(f"TEST CASE {i}: {emotion_name.upper()}")
    print(f"{'='*80}")
    print(f"Input: \"{text}\"")

    shap_values = reasoning_engine.analyze_text(text)
    print(reasoning_engine.generate_narrative(text, shap_values))
    reasoning_engine.visualize(shap_values)
    print("\n")
```

```
 Loading Model & Tokenizer...

Device set to use cuda:0

 Model Loaded. Labels: ['anger', 'fear', 'LABEL_2', 'joy', 'neutral',
'sadness', 'surprise']
 Initializing SHAP Explainer (this takes a moment)...
 Explainer Ready.

================================================================================
==========
TESTING ALL EMOTIONS WITH SINGLISH EXAMPLES
================================================================================
==========

================================================================================
==========
TEST CASE 1: ANGER
```

```
==========================================================================
==========
Input: "Cb this guy really piss me off sia, knn so angry!"

/usr/local/lib/python3.12/dist-packages/transformers/pipelines/
text_classification.py:111: UserWarning: `return_all_scores` is now
deprecated,  if want a similar functionality use `top_k=None` instead
of `return_all_scores=True` or `top_k=1` instead of
`return_all_scores=False`.
  warnings.warn(


 **Singlish Reasoning Engine**
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
**Input:** "Cb this guy really piss me off sia, knn so angry!"
**Prediction:** ANGER (Confidence: 85.2%)

**Reasoning Analysis:**
The model has identified this text as **anger**. Here is the
linguistic breakdown:

1.  **Primary Triggers:** The strongest emotional signal comes from
the word **"angry"**.
    This is supported by **"piss"**.
    These words strongly push the probability toward anger.

2.  **Contextual Nuance:**
    Although the overall sentiment is anger, the presence of
**"really"** actually works against this prediction. The model sees
this word as conflicting, but the strength of "angry" overrides it.

**Impact Factor:**
• "angry": +0.324 impact
• "piss": +0.268 impact

<IPython.core.display.HTML object>




==========================================================================
==========
TEST CASE 2: FEAR
==========================================================================
==========
Input: "I takut I lose my job if this target cannot hit one."

 **Singlish Reasoning Engine**
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
**Input:** "I takut I lose my job if this target cannot hit one."
**Prediction:** FEAR (Confidence: 88.6%)
```

Reasoning Analysis:
The model has identified this text as **fear**. Here is the linguistic breakdown:

1. **Primary Triggers:** The strongest emotional signal comes from the word **"tak"**.
    This is supported by **"if"**.
    These words strongly push the probability toward fear.

2. **Contextual Nuance:**
    The sentence structure is consistent. There are no significant conflicting words reducing the confidence.

Impact Factor:
• "tak": +0.219 impact
• "if": +0.142 impact

<IPython.core.display.HTML object>

==========
TEST CASE 3: JOY
==========
Input: "Shiok ah! So happy today, got promotion and bonus! Best day ever!"

 Singlish Reasoning Engine

Input: "Shiok ah! So happy today, got promotion and bonus! Best day ever!"
Prediction: JOY (Confidence: 84.7%)

Reasoning Analysis:
The model has identified this text as **joy**. Here is the linguistic breakdown:

1. **Primary Triggers:** The strongest emotional signal comes from the word **"Best"**.
    This is supported by **"happy"**.
    These words strongly push the probability toward joy.

2. **Contextual Nuance:**
    The sentence structure is consistent. There are no significant conflicting words reducing the confidence.

Impact Factor:

```
• "Best": +0.247 impact
• "happy": +0.175 impact

<IPython.core.display.HTML object>



================================================================================
TEST CASE 4: NEUTRAL
================================================================================
Input: "Just going home now lor, nothing special today."

 **Singlish Reasoning Engine**
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
**Input:** "Just going home now lor, nothing special today."
**Prediction:** NEUTRAL (Confidence: 66.6%)

**Reasoning Analysis:**
The model has identified this text as **neutral**. Here is the
linguistic breakdown:

1.  **Primary Triggers:** The strongest emotional signal comes from
the word **"Just"**.
    This is supported by **"home"**.
    These words strongly push the probability toward neutral.

2.  **Contextual Nuance:**
    Although the overall sentiment is neutral, the presence of
**"today"** actually works against this prediction. The model sees
this word as conflicting, but the strength of "Just" overrides it.

**Impact Factor:**
• "Just": +0.185 impact
• "home": +0.092 impact

<IPython.core.display.HTML object>



================================================================================
TEST CASE 5: SADNESS
================================================================================
Input: "Haiz so sad lah... my dog died yesterday, heart pain sia."

 **Singlish Reasoning Engine**
```

Input: "Haiz so sad lah... my dog died yesterday, heart pain sia."
Prediction: SADNESS (Confidence: 51.8%)

Reasoning Analysis:
The model has identified this text as **sadness**. Here is the linguistic breakdown:

1. **Primary Triggers:** The strongest emotional signal comes from the word **"sad"**.
    This is supported by **"Hai"**.
    These words strongly push the probability toward sadness.

2. **Contextual Nuance:**
    The sentence structure is consistent. There are no significant conflicting words reducing the confidence.

Impact Factor:
• "sad": +0.243 impact
• "Hai": +0.058 impact

<IPython.core.display.HTML object>

==========
TEST CASE 6: SURPRISE
==========
Input: "Walao eh! Cannot believe you came back! So shocked sia!"

 Singlish Reasoning Engine

Input: "Walao eh! Cannot believe you came back! So shocked sia!"
Prediction: SURPRISE (Confidence: 86.5%)

Reasoning Analysis:
The model has identified this text as **surprise**. Here is the linguistic breakdown:

1. **Primary Triggers:** The strongest emotional signal comes from the word **"shocked"**.
    This is supported by **"believe"**.
    These words strongly push the probability toward surprise.

2. **Contextual Nuance:**
    The sentence structure is consistent. There are no significant conflicting words reducing the confidence.

```
**Impact Factor:**
• "shocked": +0.385 impact
• "believe": +0.150 impact

<IPython.core.display.HTML object>
```