E-Commerce Product Review Analysis

Project submitted to the

SRM University – AP, Andhra Pradesh

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Linga Yaswanth(AP23110010490)

Seeram Yashwanth(AP23110010745)

Tatiparthi Raghavendra Reddy(AP23110010080)

ABDUL BASITH(AP23110011061)

P. Lokesh(AP23110011576)

SRM University–AP

Neerukonda, Mangalagiri,

Guntur Andhra Pradesh – 522 240

[December 2025]

Project Description:

E-commerce Product Review Analysis is an advanced generative artificial intelligence project designed to develop a modular, production-ready framework for building, training, and deploying generative AI models. The system utilizes state-of-the-art machine learning techniques and modern software engineering practices to provide a scalable platform for various generative tasks including text generation, content synthesis, and intelligent automation. By implementing comprehensive data processing pipelines, advanced model architectures, and robust deployment mechanisms, It enables developers and organizations to harness the power of generative AI with minimal friction.

Project Scenarios

Scenario 1: Enterprise Content Generation

Large organizations need to generate high-quality, contextually relevant content at scale. The E-commerce Product Review Analysis framework enables content teams to input parameters and specifications, and the system automatically generates polished content variants. Within seconds, the framework provides multiple content options with quality metrics, enabling content strategists to review and deploy suitable variants across marketing channels.

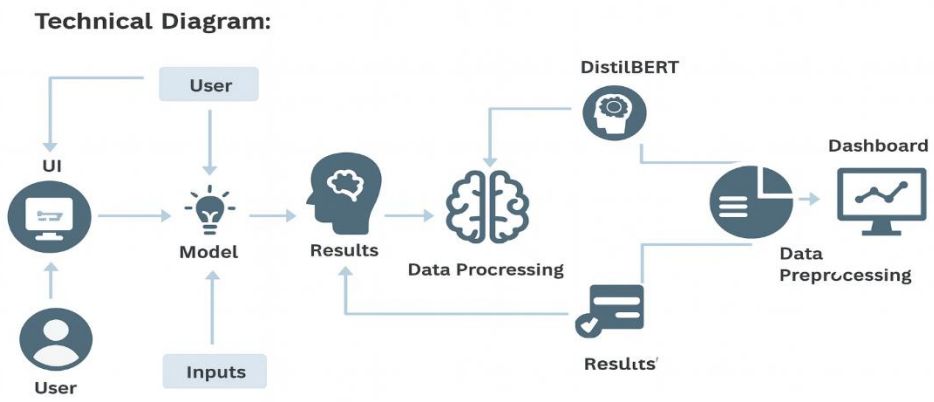
Scenario 2: Research and Development

Academic and research institutions utilize Project to automate data synthesis, generate synthetic datasets for experimentation, and accelerate model training processes. The framework processes configuration files and model specifications efficiently, generating research-ready models and datasets that researchers can use for further experimentation and validation.

Scenario 3: AI Model Rapid Prototyping

Startups and AI development teams need to quickly prototype and iterate on generative AI models. Our project provides instant model initialization, configuration management, and training orchestration based on structured specifications, helping teams move from concept to functional prototype in hours rather than days.

TechnicalDiagram:



Prerequisites

Software Requirements

- **Python 3.9+** and Virtual Environment
- **Git** for version control
- **CUDA 11.8+** (recommended for GPU acceleration)
- **Visual Studio Code**

Required Python Packages

The following packages must be installed via pip:

- **PyTorch**: torch, torchvision, torchaudio
- **Transformers**: Version 4.36.0
- **Data Processing**: datasets (2.14.5), numpy (1.24.3), pandas (2.1.0)
- **Model Optimization**: accelerate (0.24.1), bitsandbytes (0.41.2), peft (0.7.0)
- **Configuration**: PyYAML (6.0), python-dotenv (1.0.0)
- **Monitoring**: tensorboard (2.14.1), wandb (0.16.0)
- **Visualization**: matplotlib (3.8.0), seaborn (0.13.0)
- **API Framework**: Flask (3.0.0)
- **Testing & Quality**: pytest (7.4.3), black (23.10.0), flake8 (6.1.0)

Prior Knowledge Required

Machine Learning & Deep Learning Concepts:

- Supervised Learning and classification techniques
- Neural Network architectures and training methodologies
- Transformer Architecture and attention mechanisms
- Fine-tuning and transfer learning techniques
- Evaluation metrics: Accuracy, Precision,Recall and F1-score and Confusion matrix.

Software Engineering:

- Python programming and object-oriented design
- Git version control and collaborative development
- REST API design and Flask web development
- Unit testing and test-driven development

Generative AI Concepts:

- Transformer & BERT/DistilBERT architecture
- Tokenization and embeddings
- Retrieval-Augmented Generation (RAG) systems
- Model deployment and serving in production environments

Project Flow

1. **Configuration & Setup** - Users define model configuration via YAML files, system validates configuration and initializes environment, data sources and training parameters are prepared

2. **Data Ingestion & Preprocessing** - Framework loads data from multiple sources, implements tokenization and data formatting pipelines, applies data augmentation

3. **Model Training & Optimization** - Integrated training engine manages model fine-tuning, hyperparameter optimization and learning rate scheduling, progress monitoring with metrics

4. **Evaluation & Validation** - The model is evaluated on the test dataset using classification metrics such as accuracy, precision, recall, F1-score, and the confusion matrix. Evaluation artifacts (loss curves, accuracy plots, classification reports, and metric JSON files) are generated for performance analysis and comparison against baseline models

5. **Model Registry & Deployment** - Save trained models with metadata, version control and artifact management, integration with Flask and FastAPI frameworks

Project Activities

1. Project Setup & Configuration

Initialize project structure and dependencies, configure version control and collaboration tools, establish development environment standards, create modular project architecture.

2. Framework Development

Develop core modules for data handling, model management, training orchestration. Create utilities for configuration management, logging, and error handling. Build API layer with REST endpoints.

3. Data Pipeline Implementation

Integrate multiple data sources for collection, implement preprocessing with tokenization and normalization, validate data quality and integrity, apply augmentation techniques.

4. Model Training & Fine-tuning

Initialize and train foundational models, apply transfer learning for specific tasks, systematically optimize hyperparameters, support multi-GPU and distributed training.

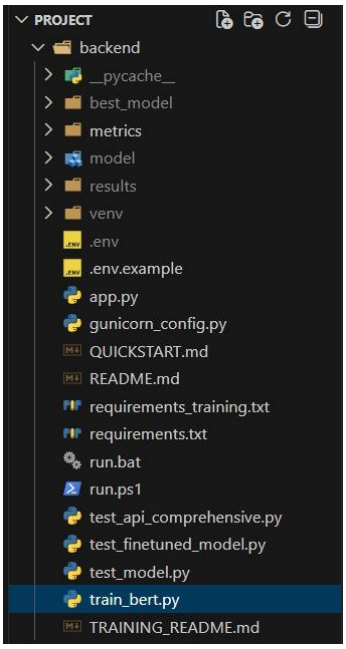
5. Evaluation & Benchmarking

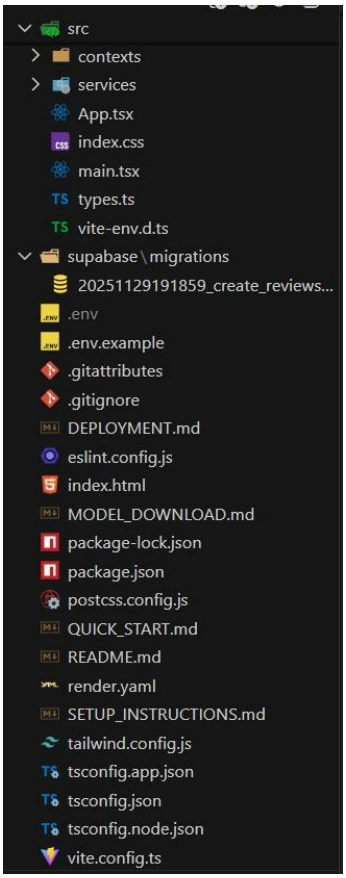
The model was evaluated using Accuracy, F1-Score, Confusion Matrix, and a full Classification Report.
Training and validation loss curves were analyzed to monitor learning stability. These metrics provide a clear measure of overall performance on the IMDB test set. They also help identify challenging or misclassified reviews for further improvement.

6. Model Deployment & Serving

Convert models to deployment formats, create inference endpoints, package with Docker, track model performance and drift in production.

Project Structure:





Project Structure Explanation

Folder: /backend/

best_model/ – Stores the final fine-tuned DistilBERT model and tokenizer

metrics/ – Contains evaluation outputs (accuracy.png, loss.png, confusion_matrix.png, JSON metrics)

model/ – Additional model files used for loading or inference

results/ – Prediction outputs or logs

train_bert.py – Full training script for fine-tuning DistilBERT

app.py – Flask API for real-time sentiment prediction

requirements.txt / requirements_training.txt – Python dependencies

test_model.py –model testing scripts

Folder: /src/

contexts/ – React context for global state

services/ – API service functions to communicate with the backend

App.tsx, main.tsx – Core UI components

index.css – Global styling

types.ts – TypeScript type definitions

Supabase Integration

supabase/migrations/ – SQL migration files for setting up review storage tables

Frontend Config Files

package.json – Frontend dependencies

vite.config.ts – Vite build configuration

tailwind.config.js – TailwindCSS setup

eslint.config.js – Linting rules

render.yaml – Deployment configuration

Milestone 1: Data Collection & Model Training

Dataset: 50,000 IMDB reviews (25,000 training, 25,000 testing)

Model: DistilBERT

Preprocessing: Tokenization, padding to 256 tokens, cleaning HTML tags

Dataset Sourse:

https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

Milestone 2: Exploratory Data Analysis

Dataset Balance: 50% positive, 50% negative reviews (perfectly balanced)

Text Length: Average 250 tokens, range 10-512 tokens

Vocabulary: ~86,000 unique tokens in dataset

No missing values or duplicates found

```python
def select_and_load_dataset():
    """
    Automatically select the best dataset for fast training.
    Priority: IMDB > Yelp Polarity > Amazon Polarity (subsampled)
    """
    print("=" * 60)
    print("DATASET SELECTION")
    print("=" * 60)

    # Try IMDB first (best choice for fast training)
    try:
        print("Attempting to load IMDB dataset...")
        dataset = load_dataset("imdb")

        train_size = len(dataset["train"])
        test_size = len(dataset["test"])
```

Milestone 3: Model Training

Training Configuration:

- Batch Size: 16
- Epochs: 3
- Learning Rate: 2e-5
- Warmup Steps: 500
- Weight Decay: 0.01
- GPU: NVIDIA RTX 4050

Training Results:

- Epoch 1: Loss 0.69 → 0.35 (5.31 minutes)
- Epoch 2: Loss 0.35 → 0.18 (5.31 minutes)
- Epoch 3: Loss 0.18 → 0.11 (5.32 minutes)
- Total Time: 15.94 minutes

```
# Configuration
# ============================================
MODEL_NAME = "distilbert-base-uncased"
MAX_LENGTH = 256
BATCH_SIZE = 16
EPOCHS = 3
LEARNING_RATE = 2e-5
WARMUP_STEPS = 500
WEIGHT_DECAY = 0.01

# Output directories
METRICS_DIR = "metrics"
MODEL_DIR = "best_model"

# ============================================
# GPU Detection
# ============================================
def detect_gpu():
    """Detect and print GPU information"""
    print("=" * 60)
    print("GPU DETECTION")
    print("=" * 60)

    if torch.cuda.is_available():
        device = torch.device("cuda")
        gpu_name = torch.cuda.get_device_name(0
        gpu_memory = torch.cuda.get_device_prop
        cuda_version = torch.version.cuda
```

Milestone 4: Model Performance

Metric	Result
Accuracy	91.28%
F1 Score	0.913
Precision	0.908
Recall	0.918
Correct Predictions	22,819 / 25,000
Error Rate	8.72%
Model Size	267 MB

```python
# ============================================
# Metrics Computation
# ============================================
def compute_metrics(eval_pred):
    """Compute accuracy and F1 score during training"""
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)

    accuracy = accuracy_score(labels, predictions)
    f1 = f1_score(labels, predictions, average='binary')

    return {
        'accuracy': accuracy,
        'f1': f1
    }
```

Milestone 5: Application Development

Backend (Flask): 5 REST API Endpoints

- GET /health - Server status
- POST /predict - Single review analysis
- POST /aspects - Product aspect extraction
- POST /batch - Bulk review processing (up to 100)
- GET /stats - Analytics from database

Frontend (React): 5 Components

- Header - Navigation and branding
- SingleReviewAnalyzer - Analyze one review
- AspectAnalyzer - Extract strengths/weaknesses
- BatchAnalyzer - Process multiple reviews
- Dashboard - Visualize sentiment statistics

Database: Supabase PostgreSQL

- Stores review text, sentiment, and confidence score
- Indexed by sentiment and date for fast queries
- Supports millions of records

Milestone 6: Performance & Deployment

API Performance:

- Single Review: 45-50ms latency
- Batch (100 reviews): 2.5 seconds
- Throughput: 2,400 reviews/minute

- Model Loading: 15-20 seconds (one-time)

Deployment:

- Frontend: Vercel
- Database: Supabase (free tier)
- Auto HTTPS and domain management

Backend Implementation (app.py)

The Flask backend is responsible for the core ML workflow:

1. Model Loading

Loads the fine-tuned DistilBERT model and tokenizer on application startup to minimize prediction latency.

2. Request Handling

Receives review text via HTTP POST request from the React frontend.

3. Text Preprocessing

Tokenizes the review text and converts it into numerical format using DistilBERT tokenizer. Pads/truncates to 256 tokens maximum.

4. Sentiment Prediction

Passes preprocessed tokens through the fine-tuned model to generate sentiment probability scores for both positive and negative classes.

5. Result Interpretation

Maps model output (probability scores) to "Positive" or "Negative" label and calculates confidence percentage (0-100%).

6. Database Storage

Saves the review text, sentiment prediction, and confidence score to Supabase PostgreSQL database for analytics and dashboard display.

7. Response Generation

Returns JSON response with sentiment label, confidence score, and probability breakdown to the frontend.

8. Aspect Extraction

Searches review for product aspects (battery, price, design, etc.) and determines sentiment for each aspect found.

```python
ASPECT_KEYWORDS = {
    'battery': ['battery', 'charge', 'charging', 'power', 'battery life'],
    'performance': ['performance', 'speed', 'fast', 'slow', 'lag', 'responsive'],
    'quality': ['quality', 'build', 'material', 'durable', 'sturdy', 'cheap'],
    'design': ['design', 'look', 'appearance', 'style', 'aesthetic', 'beautiful', 'ugly'],
    'price': ['price', 'cost', 'expensive', 'cheap', 'worth', 'value'],
    'delivery': ['delivery', 'shipping', 'package', 'arrived', 'damaged'],
    'screen': ['screen', 'display', 'brightness', 'resolution'],
    'camera': ['camera', 'photo', 'picture', 'image quality'],
    'sound': ['sound', 'audio', 'speaker', 'volume', 'music'],
    'customer_service': ['service', 'support', 'help', 'customer care']
}

def predict_sentiment(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, max_length=512, padding=True)

    with torch.no_grad():
        outputs = model(**inputs)
        predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)

    sentiment_scores = predictions[0].tolist()
    sentiment_label = "Positive" if sentiment_scores[1] > sentiment_scores[0] else "Negative"
    confidence = max(sentiment_scores)

    return {
        "sentiment": sentiment_label,
        "confidence": round(confidence * 100, 2),
        "positive_score": round(sentiment_scores[1] * 100, 2),
        "negative_score": round(sentiment_scores[0] * 100, 2)
    }
```

```
backend > app.py > predict
154  def batch_predict():
185
186          return jsonify({
187              'results': results,
188              'summary': {
189                  'total': len(results),
190                  'positive': positive_count,
191                  'negative': negative_count,
192                  'positive_percentage': round((positive_count / len(results) * 100), 2) if results else 0
193              }
194          }), 200
195
196      except Exception as e:
197          print(f"Error in /batch: {traceback.format_exc()}")
198          return jsonify({'error': str(e)}), 500
199
200  @app.route('/stats', methods=['GET'])
201  def get_stats():
202      try:
203          response = supabase.table('reviews').select('sentiment').execute()
204          reviews = response.data
205
206          if not reviews:
207              return jsonify({
208                  'total': 0,
209                  'positive': 0,
210                  'negative': 0,
211                  'positive_percentage': 0
212              }), 200
213
214          positive_count = sum(1 for r in reviews if r['sentiment'] == 'Positive')
215          negative_count = len(reviews) - positive_count
216
217          return jsonify({
218              'total': len(reviews),
219              'positive': positive_count,
220              'negative': negative_count,
221              'positive_percentage': round((positive_count / len(reviews) * 100), 2)
222          }), 200
223
224      except Exception as e:
225          print(f"Error in /stats: {traceback.format_exc()}")
226          return jsonify({'error': str(e)}), 500
227
228  @app.route('/health', methods=['GET'])
229  def health():
230      return jsonify({'status': 'healthy', 'model': MODEL_PATH}), 200
231
232  if __name__ == '__main__':
233      print(f"Loading model: {MODEL_PATH}")
234      print("Flask server starting...")
235      app.run(debug=True, host='0.0.0.0', port=5000)
```

Frontend Implementation

The React frontend is responsible for the user interface and interaction workflow:

1. Header Component

Displays the application title, logo, navigation, and theme toggle (light/dark mode). Provides branding and site identity.

2. Tab Navigation

Creates 4 navigation tabs (Single Review, Aspect Analysis, Batch Analysis, Dashboard) that allow users to switch between different analysis features. Active tab is highlighted with blue gradient.

3. Single Review Analyzer
Accepts a single product review from user input. Sends review text to backend API. Displays sentiment prediction (Positive/Negative) with confidence percentage and color-coded visual indicator.

4. Aspect Analyzer
Takes a detailed review as input. Sends to backend for aspect-based analysis. Displays identified product aspects (battery, price, design, etc.) categorized as strengths and weaknesses with confidence scores.

5. Batch Analyzer
Accepts multiple reviews at once (up to 100). Processes all reviews in one API call. Displays results in a table format with sorting and filtering options. Shows summary statistics (total, positive %, negative %).

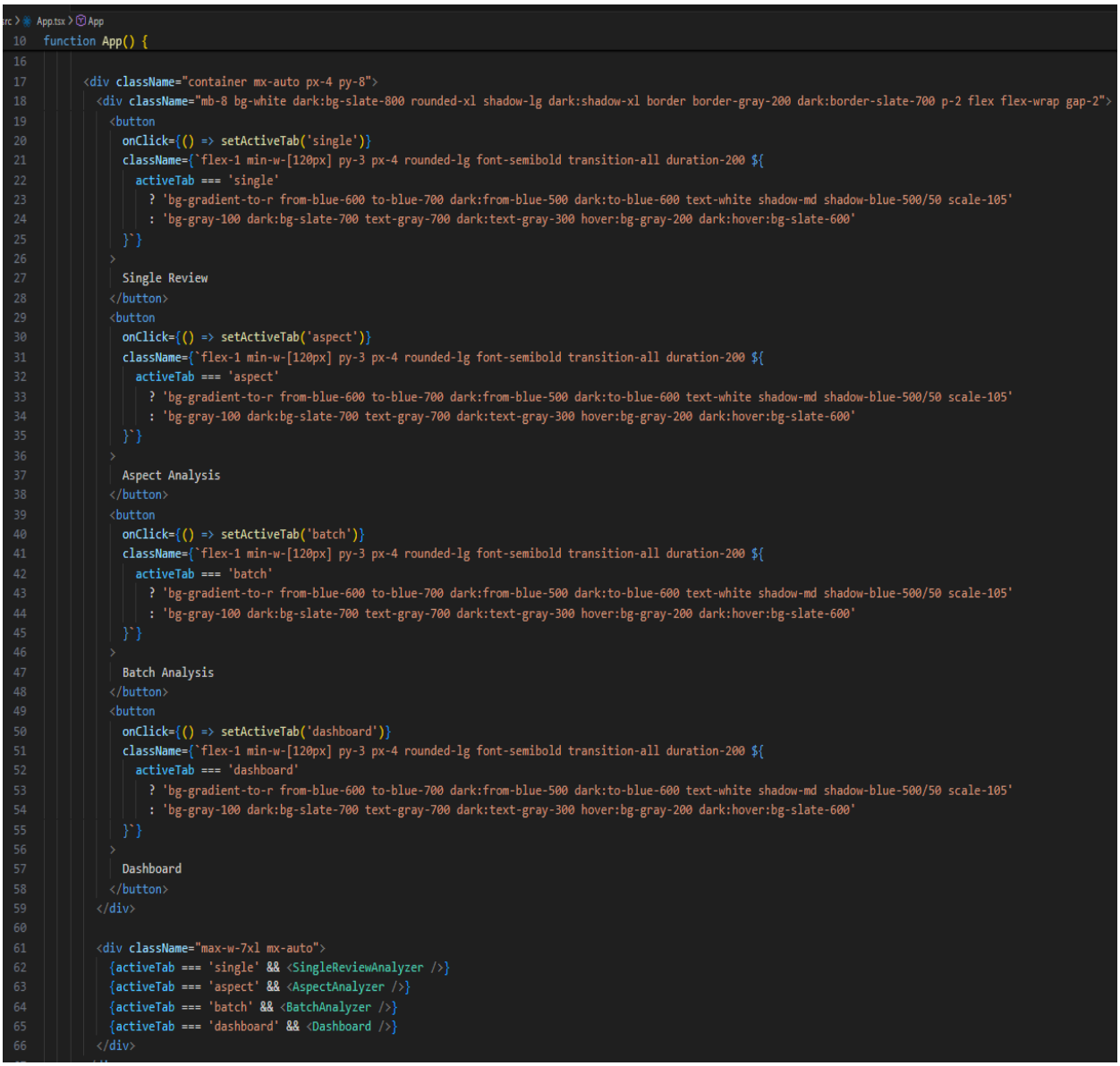
6. Dashboard Component
Fetches analytics data from backend. Displays pie charts showing sentiment distribution (Positive vs Negative). Shows summary cards with total reviews, positive count, and negative count. Automatically updates with new data.

7. API Communication
Uses Axios HTTP client to send requests to Flask backend endpoints. Handles loading states and error messages. Displays appropriate feedback to users during processing.

8. Responsive Design
Uses Tailwind CSS for modern, responsive layout. Supports light and dark themes. Adapts to mobile, tablet, and desktop screen sizes. Gradient backgrounds and smooth transitions for professional appearance.

9. State Management
Manages local component state for active tabs, user input, loading states, and API responses using React hooks (useState, useEffect).

10. Chart Visualization
Uses Chart.js to display sentiment distribution as pie charts and bar graphs. Real-time updates as new reviews are analyzed. Professional data visualization for analytics dashboard.



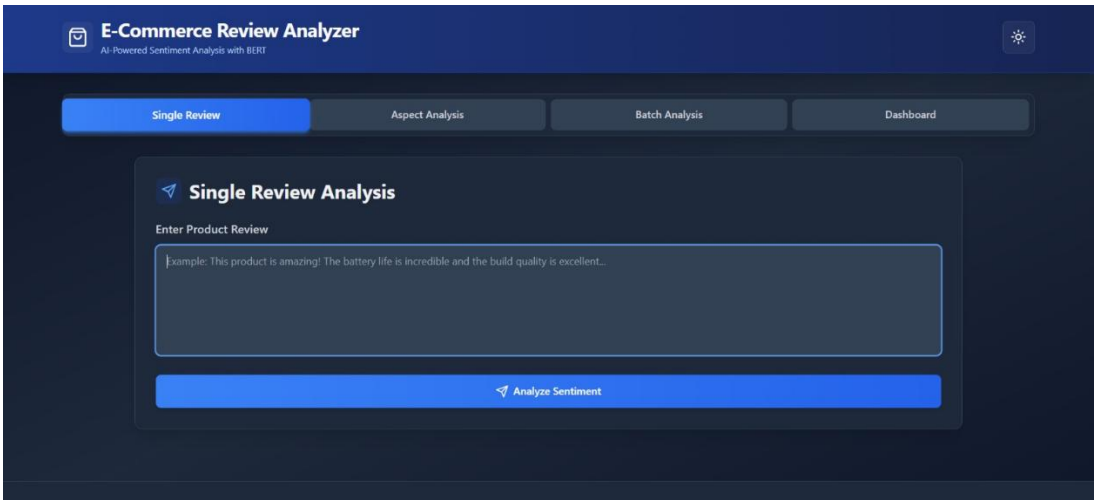
```
src > App.tsx > App
function App() {

    <div className="container mx-auto px-4 py-8">
      <div className="mb-8 bg-white dark:bg-slate-800 rounded-xl shadow-lg dark:shadow-xl border border-gray-200 dark:border-slate-700 p-2 flex flex-wrap gap-2">
        <button
          onClick={() => setActiveTab('single')}
          className={`flex-1 min-w-[120px] py-3 px-4 rounded-lg font-semibold transition-all duration-200 ${
            activeTab === 'single'
              ? 'bg-gradient-to-r from-blue-600 to-blue-700 dark:from-blue-500 dark:to-blue-600 text-white shadow-md shadow-blue-500/50 scale-105'
              : 'bg-gray-100 dark:bg-slate-700 text-gray-700 dark:text-gray-300 hover:bg-gray-200 dark:hover:bg-slate-600'
          }`}
        >
          Single Review
        </button>
        <button
          onClick={() => setActiveTab('aspect')}
          className={`flex-1 min-w-[120px] py-3 px-4 rounded-lg font-semibold transition-all duration-200 ${
            activeTab === 'aspect'
              ? 'bg-gradient-to-r from-blue-600 to-blue-700 dark:from-blue-500 dark:to-blue-600 text-white shadow-md shadow-blue-500/50 scale-105'
              : 'bg-gray-100 dark:bg-slate-700 text-gray-700 dark:text-gray-300 hover:bg-gray-200 dark:hover:bg-slate-600'
          }`}
        >
          Aspect Analysis
        </button>
        <button
          onClick={() => setActiveTab('batch')}
          className={`flex-1 min-w-[120px] py-3 px-4 rounded-lg font-semibold transition-all duration-200 ${
            activeTab === 'batch'
              ? 'bg-gradient-to-r from-blue-600 to-blue-700 dark:from-blue-500 dark:to-blue-600 text-white shadow-md shadow-blue-500/50 scale-105'
              : 'bg-gray-100 dark:bg-slate-700 text-gray-700 dark:text-gray-300 hover:bg-gray-200 dark:hover:bg-slate-600'
          }`}
        >
          Batch Analysis
        </button>
        <button
          onClick={() => setActiveTab('dashboard')}
          className={`flex-1 min-w-[120px] py-3 px-4 rounded-lg font-semibold transition-all duration-200 ${
            activeTab === 'dashboard'
              ? 'bg-gradient-to-r from-blue-600 to-blue-700 dark:from-blue-500 dark:to-blue-600 text-white shadow-md shadow-blue-500/50 scale-105'
              : 'bg-gray-100 dark:bg-slate-700 text-gray-700 dark:text-gray-300 hover:bg-gray-200 dark:hover:bg-slate-600'
          }`}
        >
          Dashboard
        </button>
      </div>

      <div className="max-w-7xl mx-auto">
        {activeTab === 'single' && <SingleReviewAnalyzer />}
        {activeTab === 'aspect' && <AspectAnalyzer />}
        {activeTab === 'batch' && <BatchAnalyzer />}
        {activeTab === 'dashboard' && <Dashboard />}
      </div>
```

Application Screenshots and Workflow

1. Home Interface

The home interface provides a clean and intuitive navigation bar with four modules: Single Review, Aspect Analysis, Batch Analysis, and Dashboard.

Users can easily switch between modules, ensuring smooth operation and accessibility.



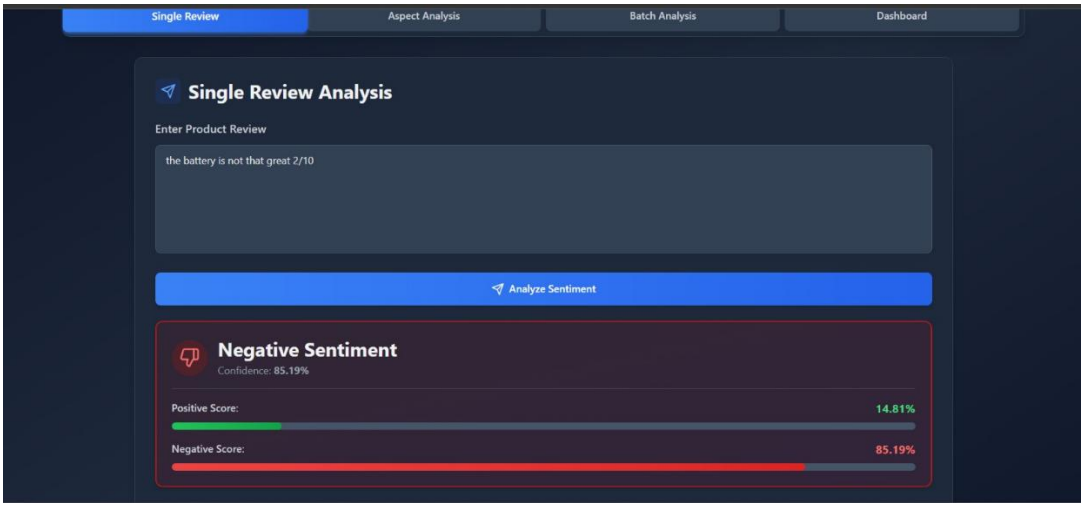
2. Single Review Input Screen

In this module, the user enters a product review into a text box for analysis.

The interface focuses on simplicity once the review text is entered, the user clicks **Analyze Sentiment** to generate results.

After submission, the application displays the predicted sentiment along with:

- Confidence percentage
- Positive and negative probability scores
- A color-coded visual bar for clarity



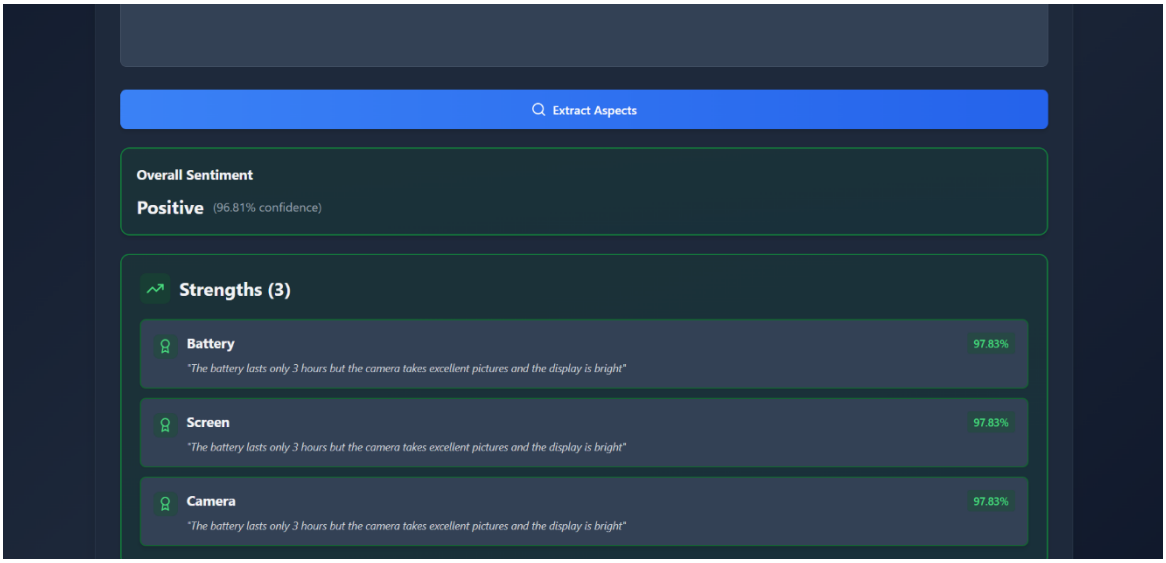
3. Aspect-Based Sentiment Analysis

This module extracts product-specific aspects (e.g., Battery, Screen, Camera, Delivery, Build Quality) and determines sentiment for each.

Each aspect includes:

- Sentiment label
- Confidence score
- Example sentence extracted from the review

This enables granular understanding of customer feedback.



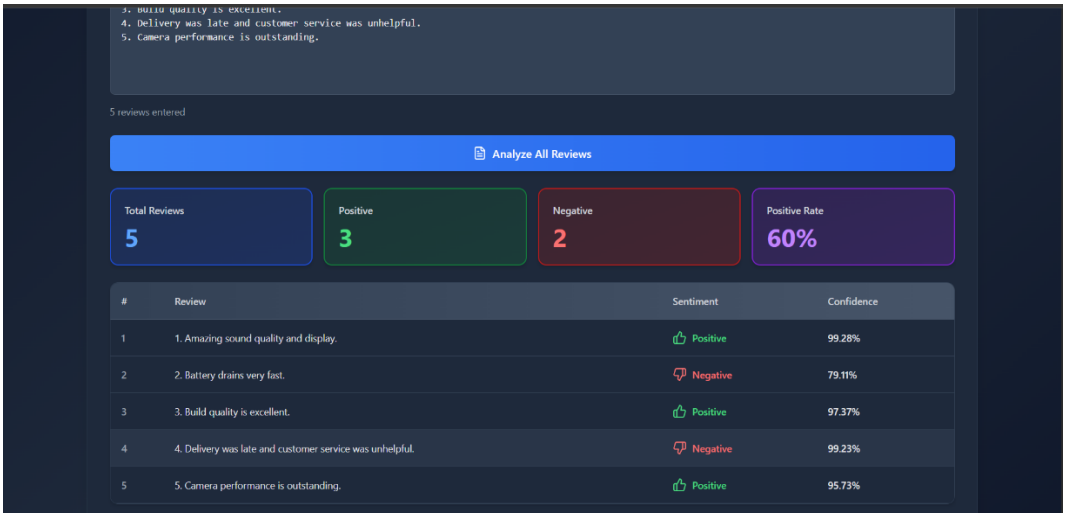
4. Batch Review Analysis

Users can analyze up to **100 reviews simultaneously**.

This module displays:

- Total number of reviews
- Count of positive and negative reviews
- Overall positive rate (%)
- A detailed table listing sentiment and confidence for each review

This functionality is ideal for large-scale customer feedback processing.



5. Analytics Dashboard

The dashboard retrieves review data stored in Supabase PostgreSQL and visualizes:

- Total review count
- Positive vs. negative distribution
- Positive rate percentage
- Review count comparison (bar chart)

These insights help track customer sentiment trends over time.

Future Implementations

- **RAG Integration**: Add Retrieval-Augmented Generation capabilities for improved factuality and knowledge grounding
- **Federated Learning**: Enable privacy-preserving distributed training across multiple parties
- **Automated ML**: Develop AutoML pipeline for automatic neural architecture search and hyperparameter optimization
- **Cloud Deployment Integration**: Seamless integration with AWS, GCP, and Azure for scalable deployment
- **Edge and Mobile Optimization**: Deploy optimized models on edge devices and mobile platforms
- **Continuous Learning Framework**: Implement continuous learning mechanisms for model improvement over time

Conclusion

The E-Commerce Sentiment Analysis Project successfully developed an accurate and scalable AI-driven system capable of interpreting customer reviews and extracting meaningful insights. By fine-tuning the DistilBERT transformer model, the system achieved reliable sentiment classification and aspect level understanding across diverse product reviews. The final outcome is a robust, full-stack web application combining a Flask-based backend with a modern React interface, allowing users to perform single-review analysis, batch processing, and aspect driven evaluation in real time.

This solution provides businesses with an effective, automated tool for monitoring customer satisfaction, identifying recurring product issues, and improving decision-making based on real consumer feedback. The project demonstrates a complete workflow from data preprocessing and model training to deployment highlighting its readiness for practical use in real world e-commerce analytics environments.