



Sentiment Analysis & Semantic Search System

Using DistilBERT + ChromaDB (Containerized with Docker)

Course: Professional Training (Intermediate) | Instructor: Ahmed Zahir

Student Name: Nusrat Suraia - 20233120064

Problem & Motivation

- **The Problem:**

- Businesses/researchers need to *analyze text* (reviews, feedback) at scale
- Traditional tools:
 - *Keyword search misses "meaning"* (great quality ≠ excellent build)
 - Manual analysis is *slow*
 - Deployment is *messy* ("it works on my machine")

- **Why This Project:**

- Solve 2 tasks in 1:
 1. Sentiment classification (positive/negative)
 2. Semantic search (find similar texts)
- Make it easy to use/deploy (no tech expertise needed!)

System Design

- **Frontend: Gradio UI:** Handles user input (text/batch/file) & displays results

The screenshot shows the Gradio UI for a sentiment analysis application. At the top, there's a navigation bar with tabs: Single Text Analysis (selected), Batch Analysis, File Upload, Semantic Search, Dashboard, and About & Help. Below the navigation, on the left, is a section titled "Enter Text" with a placeholder "Enter text to analyze its sentiment" and a text input area containing the placeholder "Type your text here... Example: 'I love this amazing product!'". On the right, there's a section titled "Sentiment Analysis Result" with a small "..." button. At the bottom left, a large blue button says "Analyze Sentiment". Below this button is a section titled "Try these examples:" with three text boxes containing sample text: "I love this product! It's amazing and works perfectly.", "The service was terrible and the staff was rude.", and "Good value for money, but could be better."

System Design

- **Backend: FastAPI** : Connects UI to models/ChromaDB via APIs (/search/semantic)

Sentiment Analysis API 0.1.0 OAS 3.1

[/openapi.json](#)

default

POST /search/semantic Semantic Search

POST /embeddings/batch Generate Embeddings

GET /health Health Check

Schemas

BatchRequest > Expand all object

HTTPValidationError > Expand all object

QueryRequest > Expand all object

ValidationError > Expand all object

The image shows a screenshot of the FastAPI documentation interface. At the top, it displays the title "Sentiment Analysis API" with version "0.1.0" and OpenAPI Specification "OAS 3.1". Below the title, there is a link to "/openapi.json". The main content is divided into sections: "default" and "Schemas". The "default" section lists three endpoints: a POST endpoint for semantic search at "/search/semantic", a POST endpoint for generating embeddings at "/embeddings/batch", and a GET endpoint for health checks at "/health". Each endpoint entry has a small dropdown arrow icon on the right. The "Schemas" section contains four schema definitions: "BatchRequest", "HTTPValidationError", "QueryRequest", and "ValidationError", each with a "Expand all" link and an "object" type indicator.

System Design

- **Model Layer** :- DistilBERT (sentiment classification)
 - Sentence-BERT (embedding generation)
- **Batch Analysis Code** (sentiment_pipeline.py) :

```
def batch_analyze(self, texts: List[str]) -> List[Dict[str, Any]]:  
    """Analyze sentiment for multiple texts"""  
    results = []  
    for text in texts:  
        result = self.analyze_sentiment(text)  
        results.append(result)  
    return results
```

System Design

- **Data Layer: ChromaDB** : Stores embeddings + metadata (sentiment, confidence) for semantic search

The screenshot shows the API documentation for the **chroma-frontend** version 1.0.0, based on OAS 3.1. The interface includes a header with the title, a version, and an OpenAPI JSON link. A green "Authorize" button with a lock icon is located in the top right corner. The main content area displays the **default** endpoint group, which contains several API endpoints:

- GET /api/v2/auth/identity** Retrieves the current user's identity, tenant, and databases.
- GET /api/v2/collections/{crn}** Retrieves a collection by Chroma Resource Name.
- GET /api/v2/healthcheck** Health check endpoint that returns 200 if the server and executor are ready
- GET /api/v2/heartbeat** Heartbeat endpoint that returns a nanosecond timestamp of the current time.
- GET /api/v2/pre-flight-checks** Pre-flight checks endpoint reporting basic readiness info.
- POST /api/v2/reset** Reset endpoint allowing authorized users to reset the database.
- POST /api/v2/tenants** Creates a new tenant.
- GET /api/v2/tenants/{tenant_name}** Returns an existing tenant by name.
- PATCH /api/v2/tenants/{tenant_name}** Updates an existing tenant by name.
- GET /api/v2/tenants/{tenant}/databases** Lists all databases for a given tenant.

ML Model (DistilBERT)

- **Why DistilBERT?**

- Lightweight (40% smaller than BERT) → fast for beginners' computers
- Pre-trained on sentiment data (SST-2 dataset) → no custom training!

- **Code Snippet** (sentiment_pipeline.py)

```
def analyze_sentiment(self, text: str) -> Dict[str, Any]:  
    """Analyze sentiment of a single text"""  
    if not text.strip():  
        return {  
            "text": text,  
            "sentiment": "NEUTRAL",  
            "confidence": 0.5,  
            "probabilities": {"NEGATIVE": 0.5, "POSITIVE": 0.5}  
        }  
  
    cleaned_text = self.clean_text(text)  
  
    # Tokenize and prepare input  
    inputs = self.tokenizer(  
        cleaned_text,  
        return_tensors="pt",  
        truncation=True,  
        padding=True,  
        max_length=512  
    )
```

Embeddings + Vector DB (ChromaDB)

- **Embedding Generation** (Sentence-BERT)

- Code:

```
self.embedding_model = SentenceTransformer("all-MiniLM-L6-v2")
```

- Output:

- 384-dimensional vector (captures text meaning)

- **ChromaDB** (chroma_manager.py):

- Stores embeddings + metadata (sentiment, confidence)

- Semantic search code:

- ```
collection.query(query_texts=[query], n_results=5)
```

# Live Demo Screenshots

- Single Text Analysis

 **Sentiment Analysis with DistilBERT**

Analyze text sentiment using DistilBERT transformer model and store results in ChromaDB vector database.

Single Text Analysis    Batch Analysis    File Upload    Semantic Search    Dashboard    About & Help

**Enter Text**  
Enter text to analyze its sentiment  
  
I love this product! It's amazing and works perfectly.

**Analyze Sentiment**

**Sentiment Analysis Result**

```
{
 text: "I love this product! It's amazing and works perfectly.",
 cleaned_text: "i love this product! it s amazing and works perfectly.",
 sentiment: "POSITIVE",
 confidence: 0.9998792409896851,
 probabilities: {
 negative: 0.00012071349920006469,
 positive: 0.9998792409896851
 }
}
```

Try these examples:

I love this product! It's amazing and works perfectly.  
The service was terrible and the staff was rude.  
Good value for money, but could be better.

# Live Demo Screenshots

- Semantic Search

 **Sentiment Analysis with DistilBERT**

Analyze text sentiment using DistilBERT transformer model and store results in ChromaDB vector database.

Single Text Analysis   Batch Analysis   File Upload   **Semantic Search**   Dashboard   About & Help

ChromaDB Search

Search Query: product

Number of Results: 1

Search in ChromaDB

Search Results:

```
{
 query: "product",
 results: [
 0: {
 document: "I love this product.",
 metadata: {
 cleaned_text: "i love this product.",
 confidence: 0.9998775720596313,
 sentiment: "POSITIVE",
 source: "sentiment_analysis"
 },
 distance: 1.0655876515912313,
 similarity_score: -0.06558765159123126
 }
 total_found: 1
}
```

# Results & Metrics

- Batch Analysis

 Sentiment Analysis with DistilBERT

Analyze text sentiment using DistilBERT transformer model and store results in ChromaDB vector database.

Single Text Analysis   Batch Analysis   File Upload   Semantic Search   Dashboard   About & Help

Enter Multiple Texts (one per line)

```
I love this product.
But i don't have money to buy.
Still i am happy that i have another
```

Analyze Batch

(+) Batch Analysis Results

```
{
 summary: {
 total_texts: 3,
 positive_count: 2,
 negative_count: 1,
 positive_percentage: 66.66666666666666,
 negative_percentage: 33.33333333333333,
 average_confidence: 0.9997107783953348
 },
 sample_results: [
 0: {
 text: "I love this product.",
 cleaned_text: "i love this product.",
 sentiment: "POSITIVE",
 confidence: 0.9998775720596313,
 probabilities: {
 negative: 0.00012246129335835576,
 positive: 0.9998775720596313
 }
 },
 1: {
 text: "But i don't have money to buy.",
 cleaned_text: "but i don t have money to buy.",
 sentiment: "NEGATIVE",
 confidence: 0.9993690848350525,
 probabilities: {
 negative: 0.9993690848350525,
 positive: 0.0006309234886430204
 }
 },
 2: {
 text: "Still i am happy that i have another",
 cleaned_text: "still i am happy that i have another",
 sentiment: "POSITIVE",
 confidence: 0.9998856782913208,
 probabilities: {
 negative: 0.00011426611308706924,
 positive: 0.9998856782913208
 }
 }
]
}
```

# Results & Metrics

- Dashboard Stats

## 🎯 Sentiment Analysis with DistilBERT

Analyze text sentiment using DistilBERT transformer model and store results in ChromaDB vector database.

Single Text Analysis   Batch Analysis   File Upload   Semantic Search   **Dashboard**   About & Help

Refresh Dashboard

**System Statistics**

```
{
 collection_stats: {
 collection_name: "sentiment_data",
 document_count: 10,
 status: "active"
 },
 application_status: "Running",
 api_status: "Available",
 model_loaded: "DistilBERT",
 timestamp: "2025-12-07 10:02:16"
}
```

# Challenges-Solutions & Summary

- **Challenges I Faced (& Fixed!)**

1. ChromaDB Connection Errors:

- Solution: Added Docker health checks (wait for DB to load)

2. Slow Inference:

- Solution: Used DistilBERT + disabled gradient calculation

3. Deployment Mess:

- Solution: Docker (no more "it works on my machine")

- **Summary**

- Built a containerized sentiment + semantic search system

- Used: DistilBERT, ChromaDB, Gradio, Docker

- Easy to use: 1-click deployment, no tech expertise

- Next Steps: Add neutral sentiment, multi-language support