Sentiment Analysis & Semantic Search System

Using DistilBERT + ChromaDB (Containerized with Docker)

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

Student Name: Nusrat Suraia - 20233120064

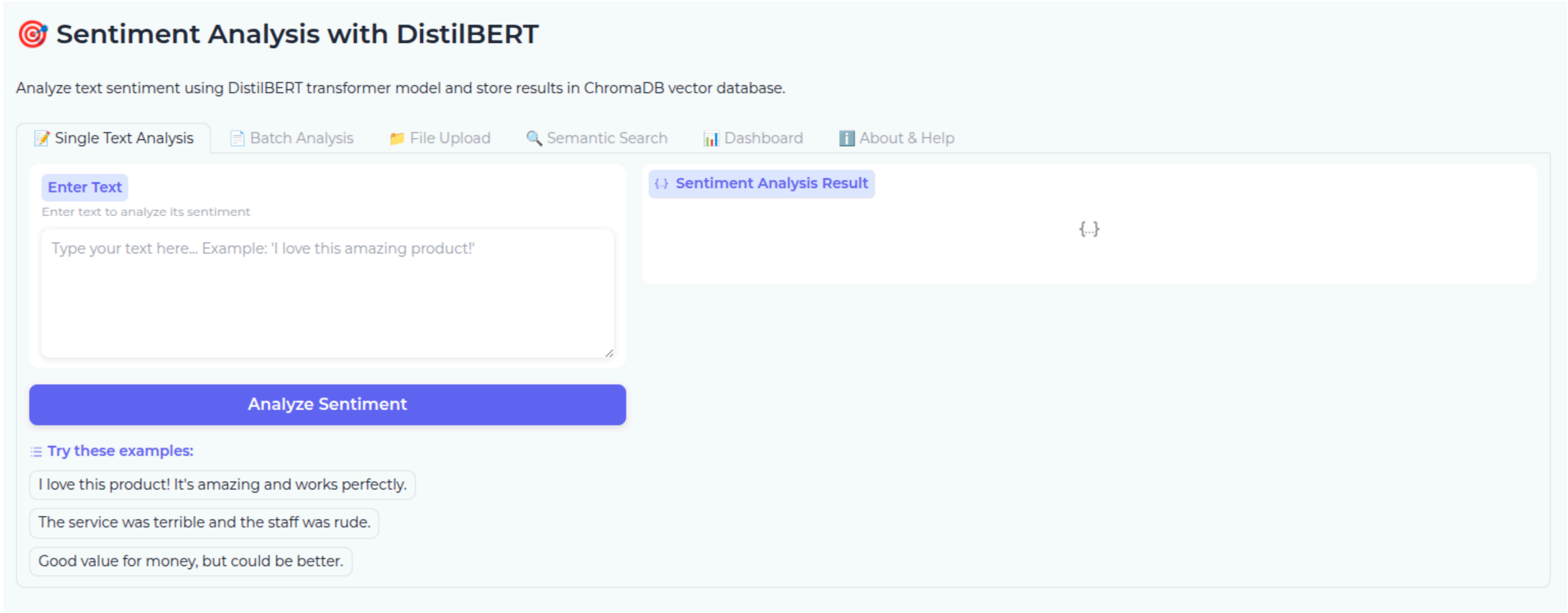
Problem & Motivation

- **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!)

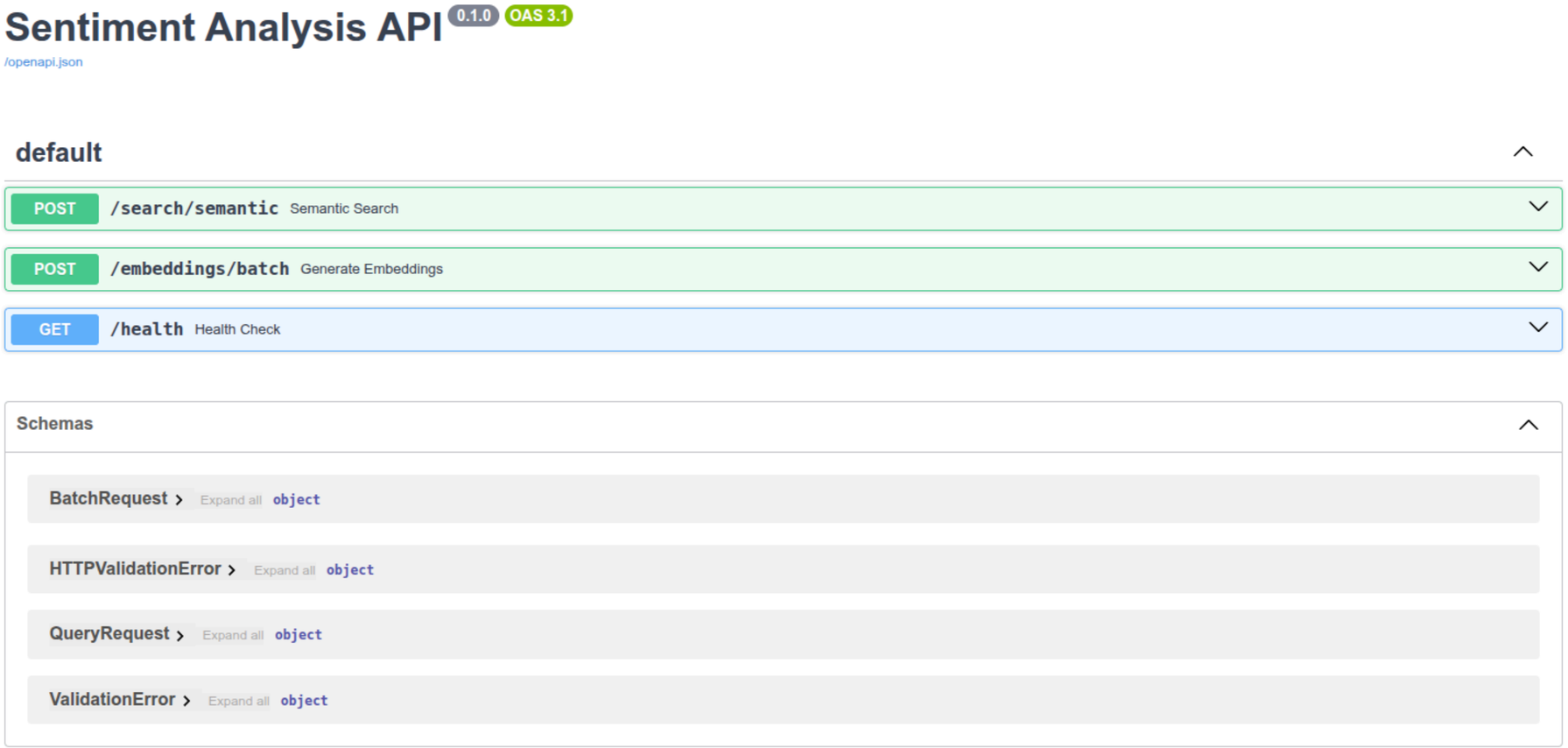
System Design

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



System Design

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



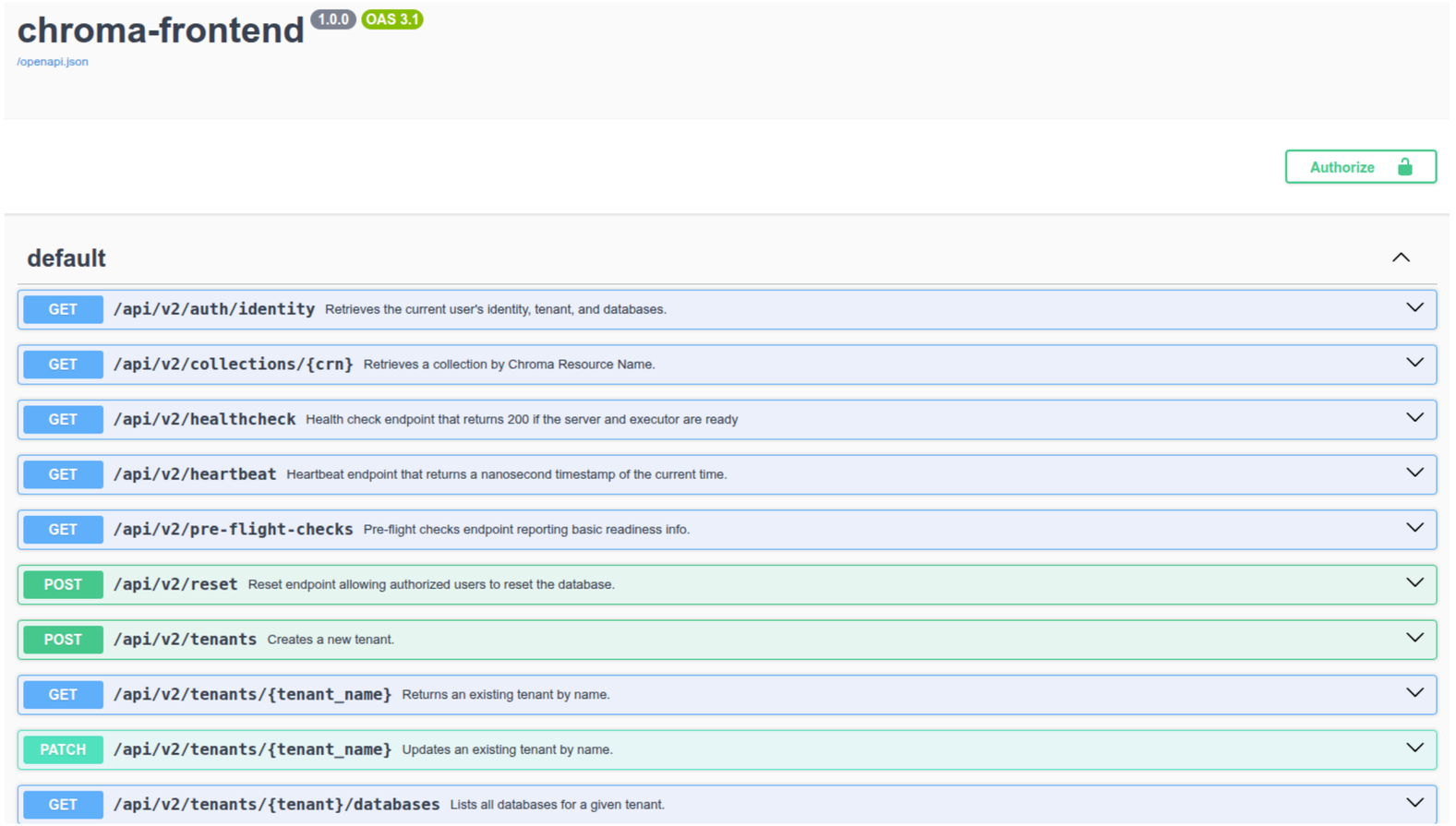
System Design

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

```python
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



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)

```python
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:

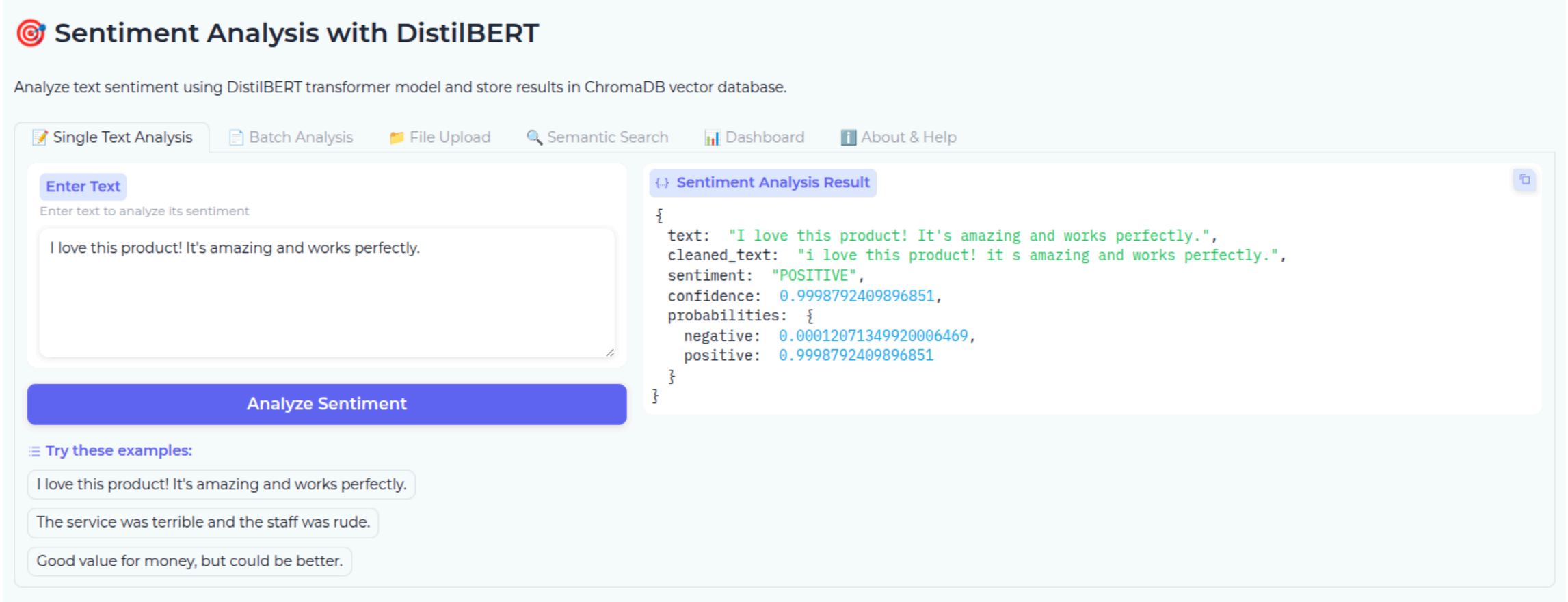
    384-dimensional vector (captures text meaning)

- **ChromaDB** (chroma_manager.py):
  - Stores embeddings + metadata (sentiment, confidence)
  - Semantic search code:

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

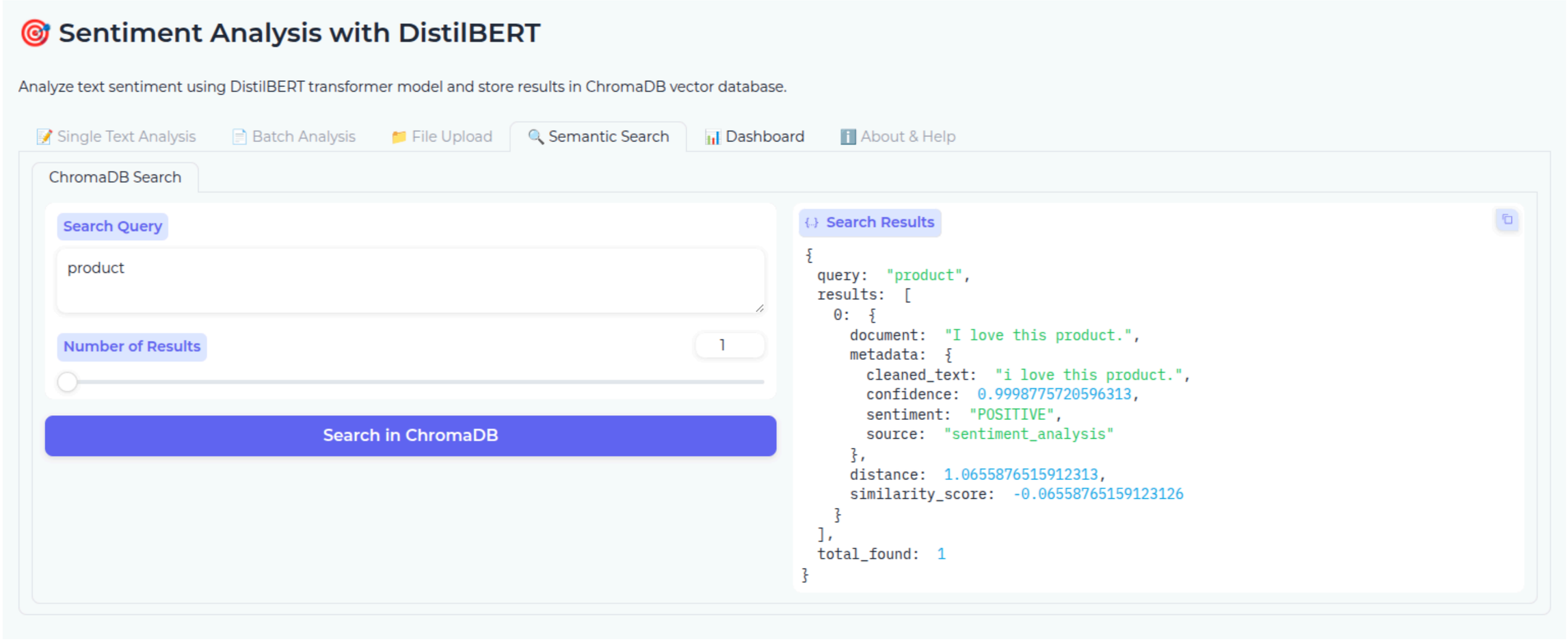
Live Demo Screenshots

- **Single Text Analysis**



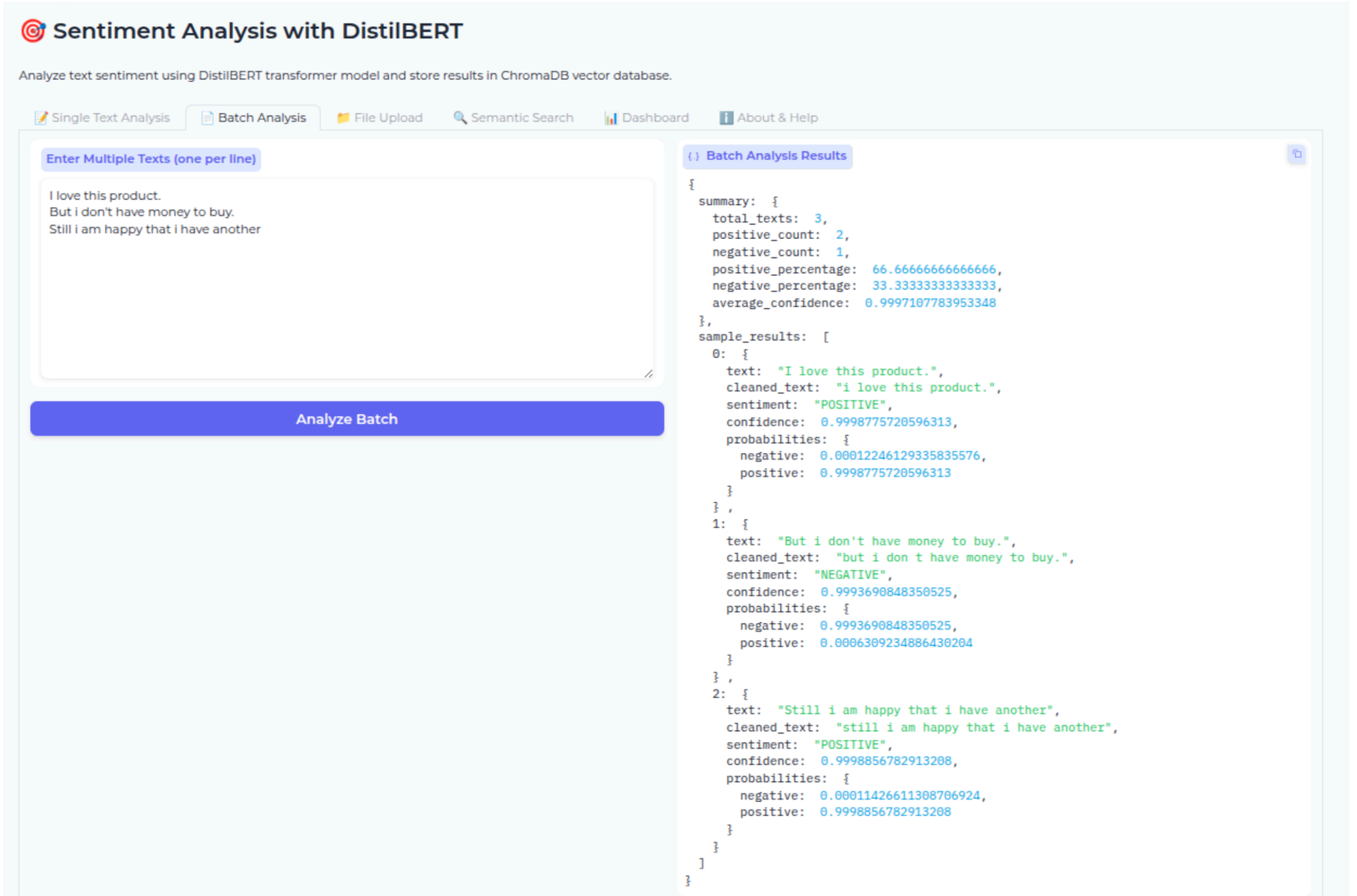
Live Demo Screenshots

- **Semantic Search**



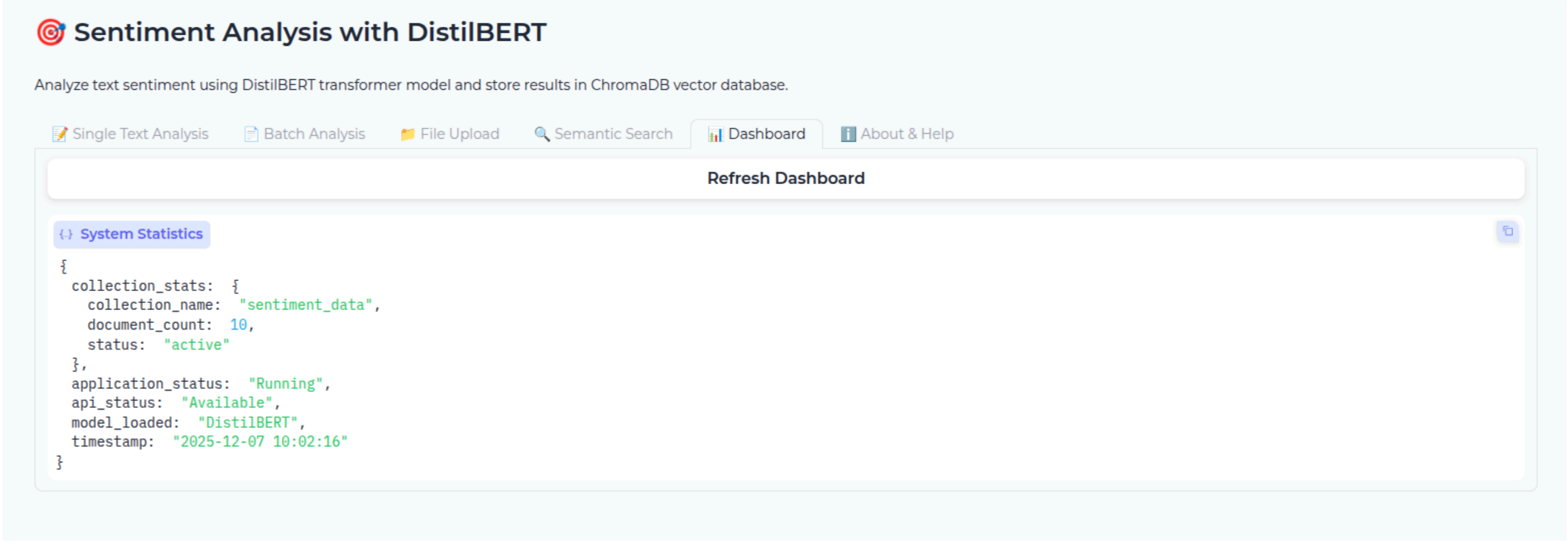
Results & Metrics

- **Batch Analysis**



Results & Metrics

- **Dashboard Stats**



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