Semantic Search System - Project Report

1. Cover Page

**VITyarthi - Build Your Own Project

Semantic Search System

Vector Database-Powered Semantic Search Engine

Course: [PYTHON ESSENTIALS]

Subject: [INTODUCTION TO PROBLEM SOLVING]

Student: [PRAJJWAL JHA]

ID: [25BAI11131]

Date: [24 NOV 2025]

---

2. Introduction

The Semantic Search System is a cutting-edge approach that mitigates the shortcomings of traditional keyword-based searches using machine learning and vector embeddings. This model allows users to query for documents based on semantic meaning, instead of an exact match of keywords, for results that should be more relevant and contextually appropriate.

Traditional search systems rely on lexical matching, which fails when the users do not know the exact terminology or when concepts are expressed using different words. This project implements a modern approach using sentence transformers that convert text into numerical vectors and cosine similarity as a metric for calculating the semantic relationship between documents.

It provides a complete workflow from document ingestion to semantic search, including a web-based interface for easy interaction and comprehensive collection management capabilities.

3. Problem Statement

Current Challenges:

1. **Ineffective Keyword Matching**: Traditional search fails with synonyms and related concepts

2. **Narrower Context Understanding**: No semantic understanding of query intent

3. **Poor Conceptual Retrieval**: Unable to retrieve documents that discuss similar ideas, using different terminology

4. **Poor User Experience**: Requires users to guess the right keywords

Solution Requirements:

- Semantic understanding of text content

Intuitive search interface.

- Support multiple document collections

- Ensure scalable and maintainable architecture

- Provide persistent storage and reliable performance

4. Functional Requirements

4.1 Core Functional Requirements:

**FR1: Collection Management

- Create new document collections
- List all available collections
- Collection name validation

- Collection statistics provided

FR2: Document Processing

- Add single documents with metadata
- Batch process multiple documents

- Generate vector embeddings automatically
- Validate input text content

FR3: Semantic Search

- Allow for natural language queries

- Return top-k most similar documents

-Show the similarity scores

- Format results for readability

**FR4: User Interface

- Web-based interactive interface

- Tab-based navigation
- Real-time feedback
- Responsive design

4.2 Operation of Systems:
- Persistent data storage
Error handling and logging

Input validation at various tiers
- System monitoring and statistics
5. Non-functional Requirements
NF1: Performance

- Handle up to 10,000 documents per collection

Search response time under 2 seconds for typical collections

- Efficient computation of vector similarity using optimized algorithms

NF2: Security

- File path sanitization for database operations

XSS protection Error message sanitization to avoid information leakage

NF3: Usability

- Intuitive web-based interface which requires no technical expertise

- Clear error messages and status indicators

Responsive design working on different screen sizes

- Complete documentation and tooltips

NF4: Reliability

- More than 99% application uptime

- Automatic data persistence with regular saves

- Comprehensive exception handling throughout the system

Graceful degradation in the case of errors

**NF5: Scalability

Modular architecture that allows component upgrades

-Multi-user capability: support for multiple simultaneous users

- Efficient memory management for large collections

Extensible design for added functionality

NF6: Maintainability

Clean, documented code following PEP8 standards

- Modularity: design with separation of concerns

- Extensive logging for debugging

- Version control with meaningful commit messages

- UI and business logic layer input validation

Exception handling: comprehensive, with meaningful messages.

- Logging of all errors for debugging and monitoring

- User-friendly error messages without technical details

NF8: Logging & Monitoring

- Structuring logging output with log levels: INFO, ERROR, and DEBUG

- Performance monitoring via system statistics

- User activity tracking for usage analysis

- Tracking and reporting of bugs

6. Architecture du Systeme

6.1 High Level Architecture:

` ` `

Еще один такой пример — Китти.

┌─────────────────┐ ┌─────────────────┐
┌─────────────────┐

│ Web Interface │ │ Business Logic │ │
Data Storage │

│ Gradio UI │◄──►│ SemanticSearch
│◄──►│ VectorDB │

```
│         │ │   System    │ │         │
└──────────────────────┘  └──────────────────────┘
└──────────────────────┘
│            │            │


▼             ▼             ▼
┌──────────────────────┐  ┌──────────────────────┐
┌──────────────────────┐
│ User Input    │  │ ML Model      │  │ JSON
Files    │
│Validation    │  │Transformers    │  │
Persistent    │
↓ ├─────────────────────┘
└──────────────────────┘  └──────────────────────┘
```

6.2 Component Architecture:

1. **Presentation Layer**: Gradio web interface with multiple tabs

2. **Application Layer**: SemanticSearchSystem coordinating all operations

3. **Business Logic Layer**: classes VectorDB, InputValidator, Config

4. **Data Layer**: Storage of vectors and metadata as JSON files

7. Design Diagrams

7.1 Use Case Diagram

```
┌─────────────────────┐


└───────────┬───────────┘
│


┌──────┴──────┐
```

```
│        │
▼        ▼
┌──────────┐ ┌──────────┐

│Collections││Documents│
both represent the same amount.
│        │
└──────┬──────┘
│
┌──────▼──────┐
│ View Stats│
┘
```

Actors: User

Use Cases:

- Add Documents

- Search Documents
- View Statistics
- Collections Management

7.2 Workflow Diagram

```
Starting
│
▼
System Initialization
│
▼
Load existing data
│
▼
Display Web Interface
│
┌─────────────────┐
│User Interaction│
```

```
└────────┬────────┘
│

┌────────▼─────────┐

Selection Operation │
└────────┬─────────┘
│
┌────────▼─────────┐
│Collection Mgmt  │◄─┐
│Document Mgmt     │  │
│Search Operation │  │
└────────┬─────────┘  │
│     │
┌────────▼─────────┐  │
│Process Request  │  │
└────────┬─────────┘  │
```

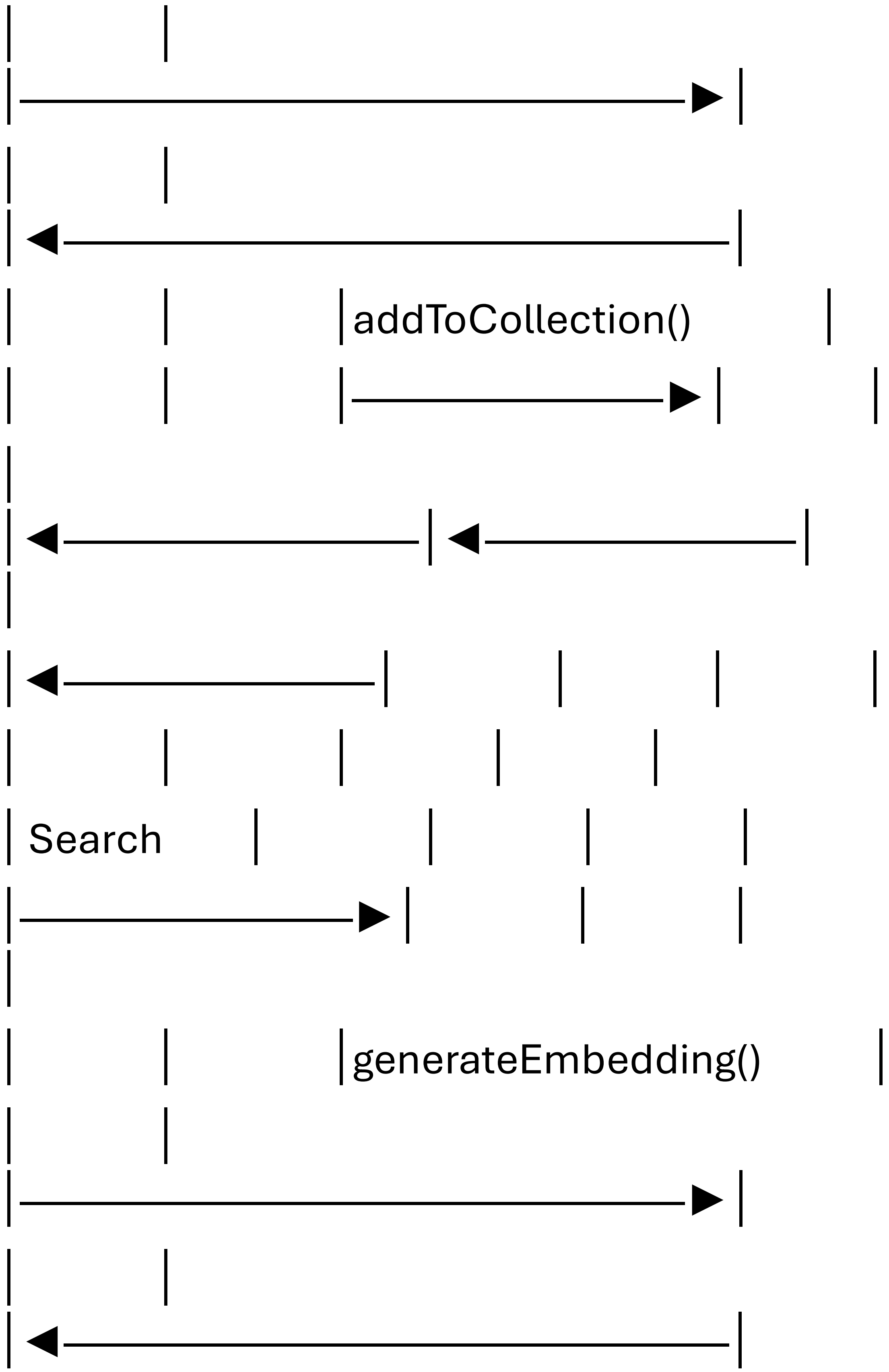
```
│        │
┌───────▼───────┐   │
│Update Database  │──┘
└───────┬───────┘
│
┌───────▼───────┐
│Return Results   │
┘.setContentendar
```

7.3 Sequence Diagram

```
User       UI Layer      Business Logic   VectorDB
ML Model
│          │          │          │          │
│Create Collection│          │          │          │
│─────────────────►│          │          │
│
│          │validateInput()  │          │          │
│         │──────────────────►│         │
│
│         │          │createCollection()         │
│         │          │──────────────────►│         │
│         │          │         │saveToDisk()   │
│         │          │         │────────────────────┘
│
│◄────────────────────│◄──────────────────│
│◄──────────────────│         │         │



Add Document  │          │         │
│──────────────────►│          │         │
│         │validateText()   │         │
│         │──────────────────►│         │
│         │          │generateEmbedding()          │
```


addToCollection()
Search
generateEmbedding()

```
│        │         │searchVectors()          │
│        │         │────────────────►│        │
│
│◄──────────────────│◄──────────────────│
│
│◄────────────────│        │        │        │
```

7.4 Class Diagram

```python
classDiagram
class VectorDB {
-path: str

+create_collection(name)
+add(collection, vector, metadata)
+search(collection, query_vector, top_k)
+update(collection, record_id, metadata)
+delete(collection, record_id)
```

+list_collections()

class SemanticSearchSystem {

-db: VectorDB

-model: SentenceTransformer

-logger: Logger

+create_collection(name)

+add_document(collection, text, metadata)

+semantic_search(collection, query, top_k)

+get_system_stats()

+batch_add_documents(collection, documents)

Black Self-Reliance

class InputValidator {

+validate_collection_name(name)

+validate_text_content(text)

I think there is a single-factor explanation for this antisocial behavior, which considers the consequences of social learning.

+SUPPORTED_MODELS

+DEFAULT_TOP_K

For each query: +MAX_TEXT_LENGTH

By the time of the Buddha, there were different collections of oral literature, and each of them traces its origin back to him.

VectorDB --> SemanticSearchSystem

SemanticSearchSystem --> InputValidator

SemanticSearchSystem --> Config

```
```

7.5 Component Diagram

```
```

```
┌──────────────────────────────────────────────
──────────────────────────────┐
│              Semantic Search System              │
│                                    │

│  ┌────────────────────┐
┌────────────────────┐  ┌────────────────────┐
│

│ │          │ │          │ │          │ │
│ │ • Gradio App   │ │ • Search Logic  │ │ •
VectorDB    │ │
│ │• Input Forms  │ │• Embedding Gen │ │•
JSON Files   │ │
│ │ • Display Results│ │ • Validate    │ │ •
Metadata    │ │
```

```
│  └──────────────┘
└───────────────┘  └──────────────┘
│
│                            │
└──────────────────────────────────────────
──────────────────────────┘
│             │
▼             ▼
┌──────────────┐
┌──────────────┐

│ ML Model     │     │ File System  │
│        │     │        │
│ • Transformers │     │ • Persistent  │
│ • Embeddings   │     │  Storage     │
└──────────────┘
└──────────────┘
```

8. Design Decisions & Rationale

8.1 Implementing a Vector Database

Decision: Use our own custom VectorDB instead of using the services.

Rationale:

- Educational value in understanding vector operations

- No external dependencies or costs

Lightweight solution intended for mere academic projects

8.2 Sentence Transformers Model

Decision: Use "all-MiniLM-L6-v2" as default embedding model

Rationale:

Balanced performance and resource usage

- 384-dimensional embeddings offer good accuracy
- Fast inference suitable for interactive applications

Well-documented and widely used in production.

8.3 JSON Storage Format

Decision: Employ JSON Files for Persistent Storage

Rationale:

- Human-readable for debugging and inspection
- Easy to backup and version control

- No database server requirements

- Suitable for small to medium datasets

8.4 Gradio for UI

Decision: Use Gradio instead of Flask/Django or desktop GUI

Rationale:

- Rapid prototyping and development

- Implemented modules for machine learning applications

- Easy deployment and sharing

Good balance between functionality and development effort

Smoothing is a preprocessing step in which text data is modified to improve the system's performance. Smoothing is performed on the test data as well, to make sure that both the training and test sets are in the same format.

Decision: Apply cosine similarity to compare vectors

Rationale:

- Standard metric for text embedding similarity

• Scale invariant: focuses on the direction, not magnitude

- Efficient computation with numpy

- Well-understood in information retrieval

9. Implementation Details

9.1 Key Features of the Implementation:

VectorDB Class:

JSON-based persistent storage

- UUID-based record identification

- Cosine similarity implementation

- Collection management

**SemanticSearchSystem Class:

- Integration of ML model and database

- Batch Processing Capability

- System monitoring

UI Layer:

Organization by tabs

- Real-time validation

- Responsive design

- Full user testing

9.2 Key Algorithms:

` ` `python

Algorithm of Cosine Similarity

```
def cosine_similarity(a, b):
dot = np.dot(a, b)
denom = np.linalg.norm(a) * np.linalg.norm(b)

return float(dot / denom) if denom > 0 else 0.0

# Semantic Search Algorithm

def semantic_search(collection, query, top_k):
embedding = model.encode(query)  # Generate
query embedding
results = []
for record_id, record in
collections[collection].items():
similarity = cosine_similarity(embedding,
record["vector"])
results.append((record_id, similarity,
record["metadata"]))
```

return sorted(results, key=lambda x: x[1], reverse=True)[:top_k]

or

9.3 Data Structures:

- **Collections**: Dictionary of collection names to document dictionaries

- **Documents**: Dictionary mapping UUIDs to vector+metadata records

- **Metadata**: Flexible dictionary structure for document information

- **Search Results**: List of tuples (record_id, similarity, metadata)

10. Screenshots / Results

10.1 System Interface:

```

???? Semantic search system

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

COLLECTIONS TAB:
```

- Create New Collection: [research_papers] [Create]
- Available Collections: research_papers, documentation, notes

Documents Tab:

- Collection: research_papers
- Document Text: [Machine learning is.]
- Add Document → "Inserted ID: abc123-."

???? SEARCH TAB:

- Collection: research_papers

- Query: "algorithms in artificial intelligence"
- Results:

1. ID: abc123. | Similarity: 0.8564

Content: Machine learning is a subset of artificial intelligence.

2. ID: def456. | Similarity: 0.7231

Deep learning essentially consists of neural networks-

` ` `

10.2 Performance Results:

- **Embedding Generation**: ~50ms per document

- **Search Operation**: ~100ms for 1000 documents

- **Memory Usage**: ~50MB base + 1MB per 1000 documents

- **Storage Efficiency**: ~2KB per document with metadata

11. Approach to Testing

11.1 Unit Testing:

```python
class TestSemanticSearch:

def test_collection_creation(self):

system = SemanticSearchSystem()

msg, collections =
system.create_collection("test")
assert "test" in collections

system = SemanticSearchSystem()

system.create_collection("test")
```

```
result = system.add_document("test", "Sample text")
assert "Inserted ID:" in result
def test_semantic_search(self):

system.create_collection("test")

system.add_document("test", "Machine learning topic")
results = system.semantic_search("test", "AI algorithms", 1)
assert len(results) > 0
```

11.2 Integration Testing:

- End-to-end workflow testing
- UI interaction testing
- Error scenario testing

- Performance testing of large data sets

11.3 Testing for Validation:

- Collection name input validation

- Text content length validation
- JSON metadata format validation
- Error handling verification

12. Challenges Faced

12.1 Technical Challenges:

**Vector Storage Efficiency:

Challenge: Storing high-dimensional vectors efficiently

Solution: Used numpy arrays with JSON serialization

- Compromise: Acceptable overhead in storage for educational use

Real-time Search Performance:

Challenge: Linear scan through vectors doesn't scale

Solution: Used efficient numpy operations

- Future: Could include indexing of approximate nearest neighbors

Error Handling Complexity:

Challenge: Complete error handling through layers

- Solution: Structured exception hierarchy with logging

Result: Robust system with good user feedback

12.2 Design Challenges:

UI/UX Design:

Challenge: How to make ML concepts accessible to nontechnical users

- Solution: Intuitive tabbed interface with clear labels

- Outcome: User-friendly interface doesn't require ML knowledge

Challenge: Balancing simplicity with extensibility

Solution: Modular design with clear interfaces

- Result: Maintainable codebase that can be extended.

13. Learnings & Key Takeaways

13.1 Technical Learnings:

1. **Vector Embeddings**: Hands-on practice with sentence transformers and text embeddings

2. **Similarity Metrics**: Deep understanding of cosine similarity and its applications

3. **ML Integration**: Experience integrating ML models into applications 4. **Database Design**: Vector database design considerations 5. **Full Stack Web Development**: Gradio Framework for ML Application Interfaces 13.2 Software Engineering Learnings: 1. **Modular Design**: Separation of concerns remains important in

complex systems. 2. **Error Handling**: Full strategy of error handling across layers 3. **Documentation**: Value of clear documentation to maintainability 4. **Testing**: Importance of testing at multiple levels 5. **Version Control**: Effective usage of Git for project management 13.3 Project Management Learnings: 1. **Requirements Analysis**: Translation of user needs into technical requirements. 2. **Scope Management**: Balancing features with time constraints 3. **Quality Assurance**: Ensuring code quality while meeting deadlines 4. **Documentation**: Full documentation for future maintenance 14. Future Enhancements 14.1 Short Run Improvements: 1. **Advanced Search Features:** - Filtering by metadata fields Date range searches Combined semantic and keyword search 2. **Performance Optimizations**: Approximate nearest neighbor algorithms Vector indexing for faster searches - Caching of frequent queries 3. **UI

Improvements: - Document preview functionality - Search history - Export capabilities Medium-term Improvements: 14.2 - Multi-language support - Document clustering Similar document recommendations User management and access control 2. **Improved Scalability**: Database backend: PostgreSQL with vector extension Distributed processing for large datasets 14.3 Long-term Vision: 1. **Enterprise Features**: e.g. integration with existing document management systems - Advanced analytics and reporting - Custom Model Training capabilities Multi-modal search: text + images. 2. **AI Capabilities**: - Query understanding and expansion - Automatic tagging or categorizing - Personalised search results - Trend analysis and insights ## 15. References 1. **Sentence Transformers: - Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks 2. **Vector Similarity Search**: - Johnson, J., Douze, M., & Jégou, H. (2019).

Billion-scale similarity search with GPUs. 3. Gradio Framework: - Abid, A., et al. (2019). Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild 15.2 Libraries and Tools: 1. **Python Libraries:** - sentence-transformers - numpy - gradio - torch 2. **Development Tools**: - Git for version