Distributed Semantic Search Engine [Project Idea]

- What the project is
- What tech stack you're using
- What each person will work on
- What deliverables and demos are expected
- Why it's valuable and aligned with course objectives

📄 **DOCUMENT TITLE**

Project Proposal: Distributed Semantic Search Engine using Milvus and Kubernetes

---

1. Introduction & Motivation

Traditional search systems rely on **keyword matching**, which struggles to capture *semantic intent* — e.g., a search for "AI model" may not retrieve "neural network" or "deep learning system."
To overcome this limitation, modern large-scale systems (e.g., Google Search, OpenAI RAG, LinkedIn Recommendations) use **vector embeddings** — numerical representations of meaning — to find semantically related items.
This project proposes building a **Distributed Semantic Search Engine** that can index and query millions of embeddings across distributed nodes.
It directly aligns with the course objectives:

- Apply **distributed concepts** (replication, sharding, fault tolerance).
- Demonstrate **horizontal scalability** and **big data processing**.
- Showcase **real-world applications** of distributed systems (AI-driven search).

---

2. Project Objectives

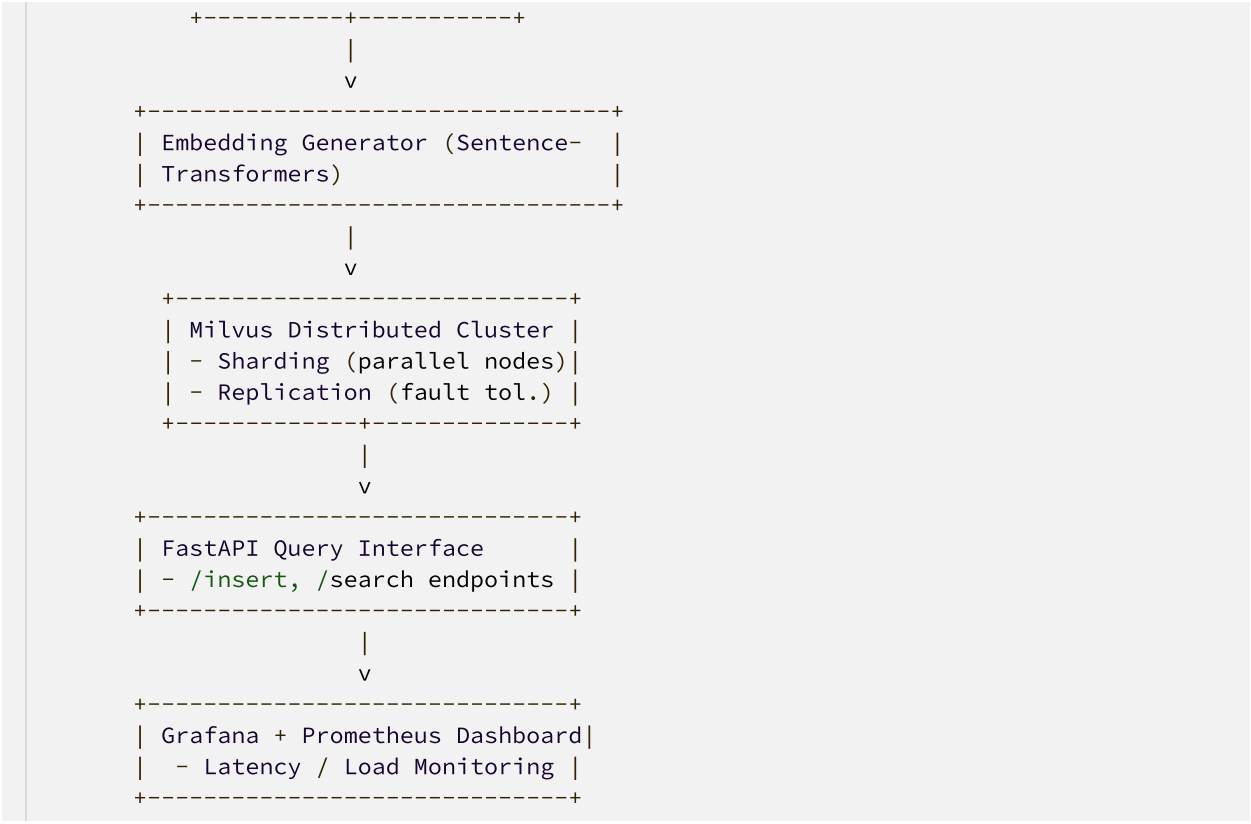
1. Design a **distributed, fault-tolerant vector search system** using Milvus (open-source vector database).
2. Ingest and index a **real dataset ≥ 1 GB** (e.g., Wikipedia abstracts).
3. Build an **API layer** for querying semantically similar texts.
4. Demonstrate **horizontal scalability** (performance improves as nodes increase).
5. Measure **latency**, **throughput**, and **replication recovery** using Grafana dashboards.
6. Deliver a **live demo, 4-page VLDB-style report**, and **presentation**.

---

3. System Overview

3.1. CONCEPTUAL FLOW

```
          +----------------------+
          |  Text Dataset (1GB+) |
```



---

4. Technology Stack

	Layer	Tool	Description
1	**Vector DB**	**Milvus 2.4+**	Distributed vector database supporting sharding and replication
2	**Orchestration**	**Docker Compose / Kubernetes (Minikube)**	Deploys multiple Milvus instances for distributed setup
3	**Embedding Model**	**Sentence-BERT (all-MiniLM-L6-v2)**	Converts text into 384-dimensional vectors
4	**API Layer**	**FastAPI (Python)**	REST endpoints for search and insertion
5	**Dataset**	**Wikipedia abstracts (1.5 GB)**	Text corpus for vector indexing
6	**Monitoring**	**Prometheus + Grafana**	Collects metrics (QPS, latency, CPU)
7	**Storage**	**MinIO (object storage)**	Milvus uses MinIO for distributed persistence
8	**Metadata**	**etcd**	Milvus cluster metadata management

---

5. Dataset Details

- **Source:** Wikipedia Abstracts Dump
- **Size:** ~1.5 GB (compressed) ≈ 20 GB uncompressed text

- **Processing:** Extract `<abstract>` tags, clean markup, split into sentences
- **Embeddings:** Generated locally using Sentence-Transformers (~3–4 GB vectors)
- **Storage:** Stored across multiple Milvus shards

---

6. Distributed System Features Demonstrated

	Concept	Implementation in Project
1	**Sharding**	Milvus partitions vectors across nodes automatically
2	**Replication**	Data replicated for redundancy and high availability
3	**Consistency**	Uses eventual consistency; validates read-after-write behavior
4	**Scalability**	Add more nodes → faster queries
5	**Fault Tolerance**	Demonstrate recovery after simulated node failure
6	**Load Balancing**	Proxy distributes queries across all active nodes

---

7. System Components & Team Roles

	Component	Responsibility	Owner
1	**Dataset Preparation**	Parse and clean Wikipedia dump, generate embeddings	*Member A*
2	**Milvus Cluster Setup**	Configure multi-node Milvus (Docker/K8s)	*Member B*
3	**API Layer**	Implement FastAPI endpoints for /insert and /search	*Member C*
4	**Monitoring Dashboard**	Set up Prometheus + Grafana metrics	*Member D*
5	**Testing & Benchmarking**	Measure latency, throughput, failover behavior	*Member E*
6	**Presentation & Report**	Prepare final demo and 4-page VLDB report	*All*

---

8. Execution Plan & Milestones

	Phase	Deliverable	Timeline
1	**Phase 1**	Architecture design & dataset selection	Week 1
2	**Phase 2**	Milvus cluster deployment (multi-node setup)	Week 2
3	**Phase 3**	Embedding generation (1.5 GB dataset)	Week 3
4	**Phase 4**	API integration & query interface	Week 4
5	**Phase 5**	Benchmarking & Grafana metrics visualization	Week 5
6	**Phase 6**	Final presentation + VLDB-formatted report	Week 6

---

9. Expected Deliverables

1. **Live Demo** – Distributed semantic search working over ≥ 1 GB dataset

2. **Presentation (7%)** – Slides showing architecture, metrics, and demo results
3. **Report (3%)** – 4-page paper in VLDB format: Introduction, System Design, Implementation, Results, Conclusion
4. **Source Code Repository** – Dockerized and reproducible setup
5. **Grafana Dashboard Snapshot** – System monitoring output

---

10. Evaluation Metrics

	Metric	Description	Goal
1	Query latency (95th %)	Average response time	< 120 ms
2	Throughput	Queries per second	> 500 QPS (simulated)
3	Index build time	Time to ingest dataset	< 30 min
4	Fault tolerance	Recovery time after node failure	< 10 s
5	Scalability	Latency change after adding node	> 25% improvement

---

11. Challenges & Mitigation

	Challenge	Mitigation
1	High memory usage during embedding generation	Use batched inference (64 samples / batch)
2	Node communication failures	Retry policies + logging
3	Limited hardware resources	Downsample text; run 2–3 node cluster locally
4	Parsing large dataset	Use streaming XML parser (SAX)

---

12. Future Extensions

- Real-time streaming ingestion with **Kafka + Flink**
- GPU-accelerated indexing via FAISS backend
- Hybrid search (semantic + keyword)
- Integration with **LLM Retrieval-Augmented Generation (RAG)**
- Cloud deployment on **GCP Kubernetes Engine**

---

13. Conclusion

This project provides a complete, realistic application of distributed systems concepts within the context of modern AI-driven search infrastructure.
By the end of the project, the team will demonstrate:

- Sharding and replication in practice
- Scalability and fault tolerance under load
- Real dataset processing and vector indexing

The **Distributed Semantic Search Engine** will not only satisfy course outcomes but also strengthen each team member's understanding of real-world distributed data systems used across FAANG and enterprise-scale AI platforms.