Distributed Semantic Search [Task Outline]

Repo: https://github.com/akhildhiman7/Distributed-Semantic-Search
Project Idea: istributed Semantic Search Engine [Project Idea]

	A	B	C	D	E
1	Member	Member Name	Task	Status	Comments
2	Member 1	Akhil	E.g. Task 1 - short description	Blocked	E.g, I'm blocked on Akhil task 2
3			E.g. Task 2	To Do	
4	Member 2			To Do	
5				To Do	
6	Member 3			To Do	
7				To Do	
8	Member 4			To Do	
9				To Do	
10	Member 5			To Do	
11				To Do	
12				To Do	

Stack (best-in-class, fast to implement):

- **Storage/Index:** Milvus 2.4+ (HNSW / IVF_FLAT), **MinIO** (object store), **etcd** (metadata)
- **Orchestration:** Docker Compose (2–3 Milvus query nodes + proxy) — faster than K8s, still realistic
- **Embeddings:** Sentence-Transformers `all-MiniLM-L6-v2` (384-d; CPU is fine), batched inference
- **API:** FastAPI + `pymilvus`
- **Dataset:** Kaggle arXiv metadata (title + abstract), target **≥1 GB** text
- **Observability:** Prometheus + Grafana (containerized)
- **Benchmarks:** Locust (QPS/latency), Python harness for cold/hot latency + recovery tests
- **Repo structure:**

```
/infra         # compose files, MinIO/etcd, Milvus, Prom/Grafana
/data          # scripts to fetch/clean arXiv, schema
/embed         # embedding pipeline & exporters
/api           # FastAPI service (search/insert/health)
/bench         # load tests, reports
/docs          # VLDB paper, slides, architecture diagram
```

System Architecture (high level)

```
[User/Client]
   -> FastAPI (/search, /insert, /health)
      -> Milvus Proxy (LB)
```

```
          -> Milvus Query Nodes (HNSW/IVF index shards; replicas)
             -> MinIO (vectors/segments) + etcd (cluster metadata)
[Prometheus] -> scrapes FastAPI & Milvus metrics  -> [Grafana Dashboards]
```

Data model (Milvus collection):

- `paper_id` INT64 (PK, auto_id=false)
- `vector` FLOAT_VECTOR(384)
- `title` VARCHAR(512)
- `abstract` VARCHAR(4096)
- `categories` VARCHAR(256)
- **Index:** HNSW `{M: 16, efConstruction: 200}` (fast queries); alt: IVF_FLAT `{nlist: 4096}` for large batches
- **Metric:** cosine (IP) or L2 (pick 1 and keep consistent)

API Contract (FastAPI):

- `POST /search` `{ "query": string, "top_k": int=5, "filters": {"categories": ["cs.LG"]} }`
- `POST /insert` `{ "paper_id": int, "title": str, "abstract": str, "categories": str }` (server embeds & upserts)
- `GET /health` → `{ "api":"ok", "milvus":"ok", "index_loaded":true }`
- `GET /metrics` (Prometheus exposition)

Member-wise Detailed Plan

👤 Member 1 — Data Engineer (arXiv Fetch & Clean)

Objective: Deliver clean, deduplicated, ready-to-embed dataset ≥1 GB with robust provenance.
Concrete tasks:

1. **Ingestion**
    - Pull `arxiv-metadata-oai-snapshot.json` locally.
    - Stream-parse JSONL; extract `id`, `title`, `abstract`, `categories`. Drop empty/short abstracts.
1. **Cleaning**
    - Normalize whitespace; strip HTML/LaTeX; collapse multiple spaces.
    - Concatenate `title + ". " + abstract` → `text`.
    - Deduplicate by normalized title hash + first 200 chars of abstract.
1. **Partitioning**
    - Write **Parquet** partitions (`/data/out/clean/part-*.parquet`) ~100–250 MB each.
    - Produce a **10k row sample** for early integration (`sample.parquet`).

1. **Data dictionary & stats**
   - CSV of category counts; descriptive stats (avg length, #records).
   - Document exact filters so the dataset is reproducible.

Deliverables / DoD

- `data/clean_arxiv_parquet/` (≥1 GB total), `data/sample.parquet` (10k)
- `data/README.md` (commands, schema, filters)
- `data/profile.json` (counts, lengths, categories)

Can start Day 1; no dependencies.

👤 Member 2 — ML Engineer (Embeddings Pipeline)

Objective: Produce high-quality embeddings with batched CPU inference; export aligned with metadata.
Concrete tasks:

1. **Model & batching**
   - Use `sentence-transformers` `all-MiniLM-L6-v2`, `batch_size=64` (tune by RAM).
   - Persist embeddings in **NumPy memmap** to avoid RAM blowups.

1. **Processing pipeline**
   - Read partitions sequentially; keep `(paper_id, vector, title, abstract, categories)`.
   - Save per-partition **Feather/Parquet** + `.npy` or `.npy.memmap`.

1. **Integrity & speed**
   - Hash checks to ensure row order alignment.
   - Log `embeddings/speed_report.md` (docs/sec, ETA).

1. **Optional optimizations**
   - Try `normalize_embeddings=True` (cosine).
   - Evaluate int8 quantization (optional note in report).

Deliverables / DoD

- `embed/embeddings/part-*.npy` (+ matching metadata parquet)
- `embed/embedding_pipeline.py` (idempotent)
- `embed/README.md` (hardware used, speed, parameters)

Depends on M1's sample early (Day 2), full set by Day 3–4.

👤 Member 3 — Systems Engineer (Milvus + Storage + Indexing)

Objective: Stand up a **multi-node Milvus** over Docker Compose with MinIO & etcd; load data; build index; prove HA.
Concrete tasks:

1. **Infra bring-up**
    - Compose stack: etcd, MinIO, Milvus standalone → **then** switch to **cluster with 1 proxy + 2 query nodes**.
    - Persist volumes; .env for ports/credentials.
1. **Collection & index**
    - Create `papers` collection w/ schema above.
    - Insert metadata + vectors in batches from M2 outputs.
    - Build **HNSW** index; `collection.load()`.
1. **HA/replication**
    - Add second query node; verify proxy LB.
    - Kill one query node mid-search → demonstrate continued service.
1. **Exportable scripts**
    - `infra/scripts/create_collection.py`, `load_data.py`, `build_index.py`, `smoke_search.py`.

Deliverables / DoD

- `infra/docker-compose.yaml` (cluster mode), `infra/.env.example`
- `infra/scripts/*` (create, load, index)
- `infra/ops_guide.md` (start/stop, failure demo)
- Evidence: screenshot/logs of node loss + continued query success

Can start Day 1 using synthetic vectors; integrate real embeddings by Day 4–5.

👤 Member 4 — Backend Engineer (FastAPI + Client)

Objective: Ship a clean, documented API for `/search` & `/insert` with unit tests and a minimal UI.
Concrete tasks:

1. **App skeleton**
    - FastAPI app; config via env (`MILVUS_HOST`, `COLLECTION_NAME`, metric).
    - Dependency for SentenceTransformer (for query embedding only).
1. **Endpoints**
    - `POST /search` → embed `query` → Milvus search (`top_k`, optional `categories` filter). Return list of `{paper_id,title,abstract,categories,score}`.
    - `POST /insert` → embed & upsert one paper (calls Milvus insert).
    - `GET /health` + `/metrics` (Prometheus client).
1. **Quality**
    - Unit tests with `pytest` + Milvus mocked interface (adapter class).
    - Rate limit middleware (basic).
    - CORS for simple web client.

1. **Demo UI (nice to have)**
    - Streamlit page in `/api/ui/` with a search box and result list.

Deliverables / DoD

- `api/main.py`, `api/requirements.txt`, `api/Dockerfile`
- `api/tests/` (search & insert tests)
- `api/README.md` (run locally; curl examples)
- Successful integration against M3's Milvus proxy

Can start Day 1 with mock; switch to live Milvus Day 4.

👤 Member 5 — DevOps & Evaluator (Monitoring + Benchmarks + Paper)

Objective: Provide hard numbers, dashboards, and final paper/slides; prove scale and resilience.
Concrete tasks:

1. **Observability**
    - Prometheus scrape for FastAPI (`/metrics`) & Milvus (use exporter or scrape proxy stats).
    - Grafana dashboards: latency (P50/P90/P95), QPS, CPU/mem, node up/down.
1. **Benchmarks**
    - **Locust** workload: configurable RPS, query sets of 100 canned queries.
    - Python harness to measure cold vs. warm latency, `top_k` sensitivity, index type (HNSW vs IVF) comparison.
    - **Failure drill:** kill one query node under load, chart error rate & recovery time.
1. **Documentation**
    - `bench/report.md` with charts (export PNGs from Grafana).
    - Assemble **VLDB 4-page** report (Intro, System, Implementation, Evaluation, Discussion, Refs).
    - Build crisp 6–8 slide deck; include architecture & dashboards.

Deliverables / DoD

- `infra/prometheus.yml`, `infra/grafana/` provisioning
- `bench/locustfile.py`, `bench/harness.py`
- `docs/COMP6231_Final_Report.(tex/pdf)`, `docs/slides.pdf`
- Dashboard JSON exports + screenshots

Can start Day 1 (wire to mock endpoints), go live once M4 connects to M3.

Integration & Quality Gates

Gate A (Day 4):

- M1 sample + M2 sample embeddings ready
- M3 cluster alive w/ test data; index created
- M4 `/search` returns real results on sample
- M5 has first Grafana metrics showing API latency

Gate B (Day 9):

- ≥1 GB text processed & indexed
- `/search` stable under Locust at target QPS
- Dashboard shows P95 latency & node health

Gate C (Day 12):

- Failure drill (kill 1 query node): <10s recovery, no API crash
- Side-by-side HNSW vs IVF comparison chart

Final (Day 14):

- Demo script rehearsed; report & slides finalized

Acceptance Criteria (Definition of Done)

- **Functionality:** `/search` returns top-k semantically relevant arXiv papers in **<120ms P95** on sample queries; **<300ms P95** on full set (targets adaptable by hardware).
- **Scale:** At least **1 GB** text ingested; **index built** and **loaded**.
- **Resilience:** Under steady RPS, a node failure **does not** crash API; recovery demonstrated & measured.
- **Observability:** Grafana dashboards show latency, QPS, node status.
- **Reproducibility:** Fresh clone + `docker compose up` + one command to index sample should yield a working demo.
- **Docs:** VLDB-style 4-pager + slides with metrics & architecture.

Two-Week Gantt-Style Schedule (Who/What/When)

	Day	M1 – Data	M2 – Embeddings	M3 – Milvus/Infra	M4 – API/UI	M5 – DevOps/Bench
1	1	Fetch JSONL; schema plan	Set up ST model; test batch	Compose (etcd/MinIO/Milvus-standalone); smoke	FastAPI skeleton; mock Milvus	Prom+Grafana stack; seed dashboards
2	2	Clean, dedup, partition; **sample.parquet**	Dry-run on sample; memmap setup	Switch to **cluster** (proxy + 2 query nodes)	Wire `/search` to mock; unit tests	Connect Prom → API; synthetic load
3	3	Deliver ≥1GB parquet set	Batch encode sample; export vectors+meta	Create collection; load **sample vectors**	Connect to live Milvus; first real results	First latency charts; basic Locust run
4	4	Data README + profile	Start **full** embedding run	Build HNSW index; `collection.load()`	`/insert` endpoint; CORS	Dashboard polish; capture baseline
5	5	(buffer)	Full run continues	Bulk insert automation	Response shaping; error handling	Draft benchmark plan (QPS tiers)
6	6	(buffer)	Finish full encode; export	Load full vectors; rebuild index	End-to-end validation on full set	Baseline run on full set (charts)
7	7	(buffer)	Speed report + params	Tune `ef`, `M` (HNSW); alt IVF build	Small UI (Streamlit)	Report skeleton; import figures
8	8	(support)	(support)	Add 3rd query node; verify LB	Logging, `/health`, `/metrics`	Load test (RPS ramp, P95/P99)
9	9	(support)	(support)	Optimize insert/search params	API hardening; timeouts	Compare HNSW vs IVF; table/plots
10	10	(support)	(support)	**Failure drill**: kill node under load	Graceful errors; retries	Recovery chart; error budget notes
11	11	(support)	(support)	Stabilize configs; ops guide	Final API README; curl scripts	Draft findings; finalize dashboards
12	12	(support)	(support)	Freeze infra; tag release	Freeze API; tag release	Write VLDB report (Methods/Results)
13	13	(support)	(support)	Demo rehearsal checklist	Demo script; sample queries list	Slides (arch, metrics, drill)
14	14	(all)	(all)	Final demo	Final demo	Final paper + slides

Notes on “best quality” choices

- **HNSW** chosen for **low latency** at moderate memory; IVF available for ablations.
- **Memmap embeddings** to avoid memory spikes during export/load.
- **Proxy + multiple query nodes** gives real LB & resilience semantics without K8s overhead.
- **Prometheus/Grafana** over ad-hoc prints → professional-grade observability.
- **Contract-first API** + mocks → M4 unblocked from Day 1.

- **Gate-based integration** avoids late surprises; failure drill is non-negotiable.