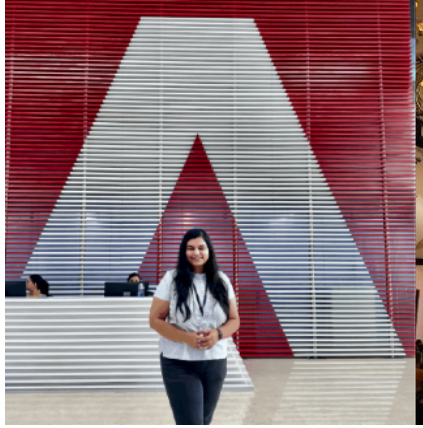
Beyond the Vector DB: Building a RAG Stack That Actually Stays Fresh

Drishti Jain

@drishtijjain





About Me

- Software Developer / ML
- Published Technical Book
- Social Entrepreneur
- International Tech Speaker

- Career Coach – SkillUp with Drishti

@drishtijjain

Retrieval Augmented Generation (RAG)

@drishtijjain

Retrieval Augmented Generation (RAG)

- RAG = Retrieval-Augmented Generation
- Combines two components:
  - **Retriever**: Pulls relevant documents from external knowledge source (e.g., vector DB)
  - **Generator**: Uses LLM to generate answers based on retrieved context
- Benefits: factual grounding, domain-specific responses, better interpretability

@drishtijjain


Why RAG is Powerful
@drishtijjain

Why RAG is Powerful

Overcomes knowledge cutoff of LLMs

Customizes generation using your data

Safer and more up-to-date than pure prompting

@drishtijjain

Why Freshness Matters

@drishtijjain

Why Freshness Matters

- Hallucinations from outdated data
- Missed updates lead to user mistrust
- Examples from real RAG failures (e.g., "company policy" answer is 6 months outdated)

Traditional RAG Setup

@drishtijjain

Traditional RAG Setup

- Components: Vector DB + Chunking + LLM
- Assumes static knowledge base
- Works well *only initially*

@drishtijjain


RAG in the Real World
@drishtijjain

RAG in the Real World

Internal documentation changes weekly

Wikis, Confluence, GitHub READMEs, product specs

Need continuous updates & quality assurance

@drishtijjain

Our Goal

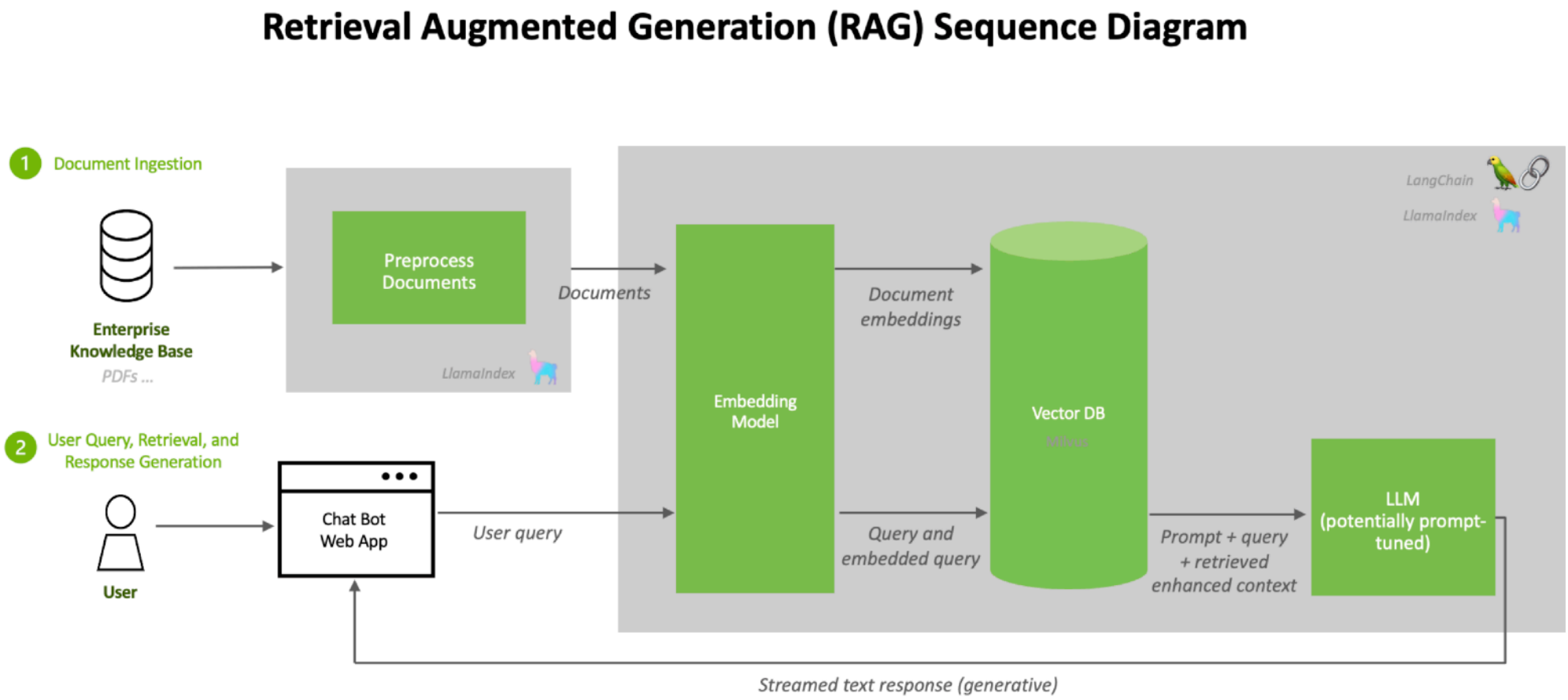
- RAG stack that updates *without full re-index*
- Tracks changes, freshness, and versioning
- Optimizes cost + quality tradeoffs

@drishtijjain

Overview of RAG Architecture

@drishtijjain

Overview of RAG Architecture



Source: https://developer.nvidia.com/blog/rag-101-demystifying-retrieval-augmented-generation-pipelines/?utm_source=chatgpt.com

@drishtijjain


Key Challenges

Key Challenges

Avoiding redundant re-indexing

Detecting meaningful content changes

Preventing silent embedding drift

Retraining at the right time

Prioritizing fresh answers

@drishtijjain

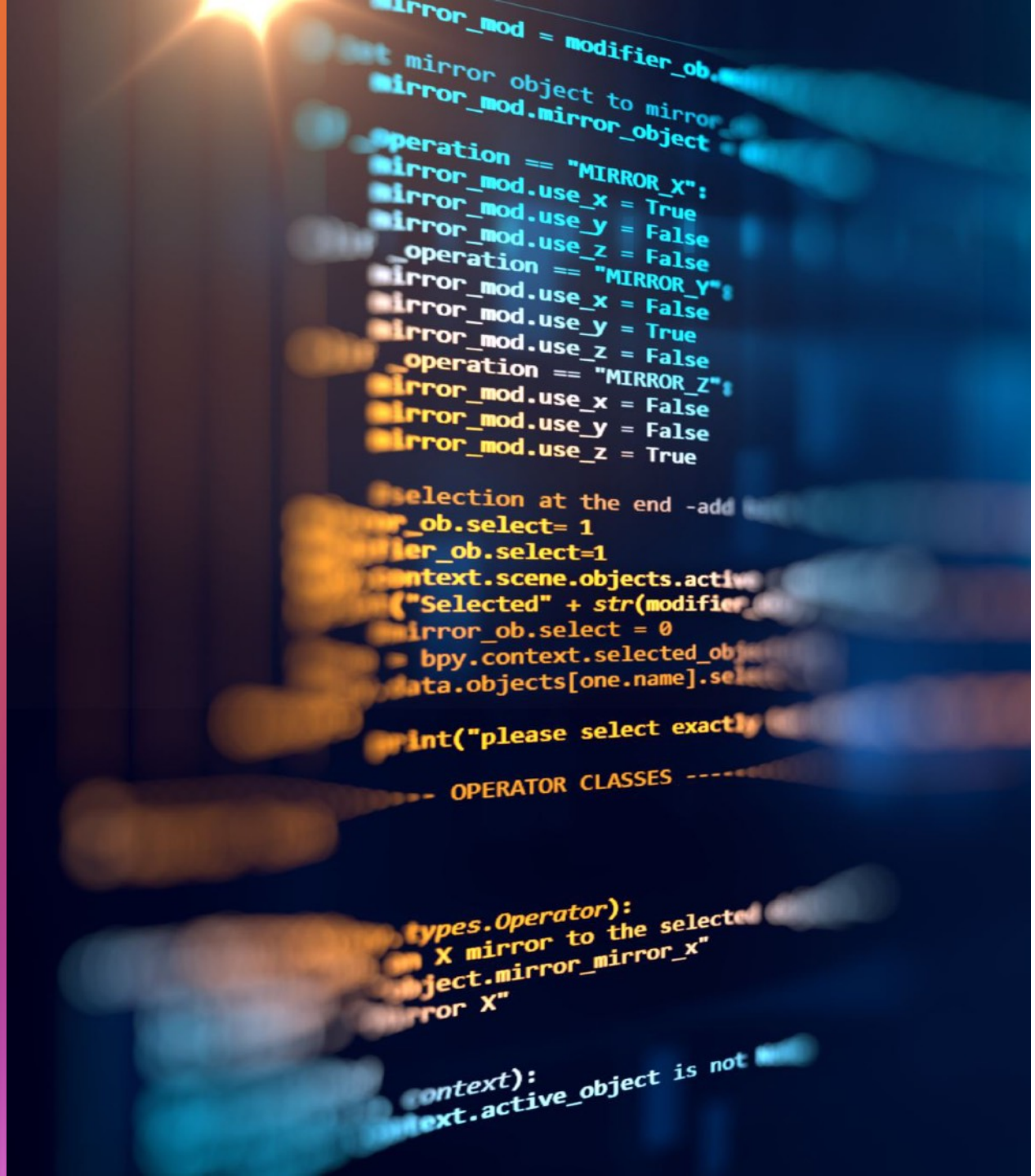

Step 1 — Change Detection
@drishtijjain

Change Detection

DETECTING IF A DOCUMENT HAS CHANGED SINCE LAST INDEXING

AVOID COSTLY RE-EMBEDDING

@drishtijjain



Technique — Checksum-Based Detection

- Compute hash per text chunk
- Compare with previous version
- Efficient and stateless

@drishtijjain

Technique — Checksum-Based Detection

- Download raw docs
- Chunk them
- Compute checksums
- Compare to previous version
- Only embed+index changed chunks

```python
import hashlib
import os
import json

def chunk_text(text, chunk_size=1000):
    return [text[i:i+chunk_size] for i in range(0, len(text), chunk_size)]

def checksum(s: str) -> str:
    return hashlib.sha256(s.encode('utf-8')).hexdigest()

def detect_changes(docs_dir, state_file='state.json'):
    if os.path.exists(state_file):
        prev = json.load(open(state_file))
    else:
        prev = {}
    curr = {}
    changed = []
    for fname in os.listdir(docs_dir):
        text = open(os.path.join(docs_dir, fname)).read()
        for chunk in chunk_text(text):
            cs = checksum(chunk)
            curr[cs] = {'file': fname, 'chunk': chunk}
            if cs not in prev:
                changed.append((cs, fname, chunk))
    json.dump(curr, open(state_file, 'w'))
    return changed

# Usage
changed = detect_changes("docs/")
print(f"{len(changed)} new/changed chunks detected")
```

@drishtijjain

Step 2 — Embedding Metadata and Versioning

@drishtijjain

Step 2 — Embedding Metadata and Versioning

Track: Embedding model, timestamp, source

Avoid accidental mixing of incompatible embeddings

@drishtijjain

Why Versioning Matters

@drishtijjain

Why Versioning Matters

EMBEDDING MODEL UPGRADES SILENTLY ALTER VECTOR SPACE

RETRIEVAL ACCURACY DEGRADES

@drishtijjain

Embedding Versioning

@drishtijjain

```python
EMBED_MODEL = "openai-embedding-2.3.0"

def embed_batch(chunks):
    # ... pseudocode embedding call
    return [get_embedding(text) for text in chunks]

def index_new(changed):
    for cs, fname, chunk in changed:
        emb = embed_batch([chunk])[0]
        vector_db.upsert(
            id=cs,
            vector=emb,
            metadata={'file': fname, 'model_ver': EMBED_MODEL, 'ts': time.time()}
        )
```

Embedding Versioning & Drift Detection

@drishtijjain


Step 3 — Semantic Change Detection
@drishtijjain

Semantic Change Detection

- Sometimes content changes *subtly*
- Text looks different but *meaning* is same — or vice versa

Technique — Semantic Diffing

- Use sentence transformers to compare old vs new chunk meaning

@drishtijjain

Semantic Diffing & Metadata Tagging

```python
from sentence_transformers import SentenceTransformer, util

st = SentenceTransformer('paraphrase-MiniLM-L6-v2')

def semantically_diff(old, new, threshold=0.7):
    emb_old = st.encode(old, convert_to_tensor=True)
    emb_new = st.encode(new, convert_to_tensor=True)
    return util.pytorch_cos_sim(emb_old, emb_new).item() < threshold
```

@drishtijjain

Apply semantic diffing on changed chunks, tag them:

```
for cs, fname, chunk in changed:
    old_chunk = load_old_chunk(cs)
    if old_chunk and semantically_diff(old_chunk, chunk):
        tags = ['major_update']
    else:
        tags = ['minor_update']
    vector_db.upsert(..., metadata={'tags': tags})
```

@drishtijjain


Step 5
Indexing with
Intelligence
@drishtijjain

Indexing with Intelligence

Only index meaningful changes

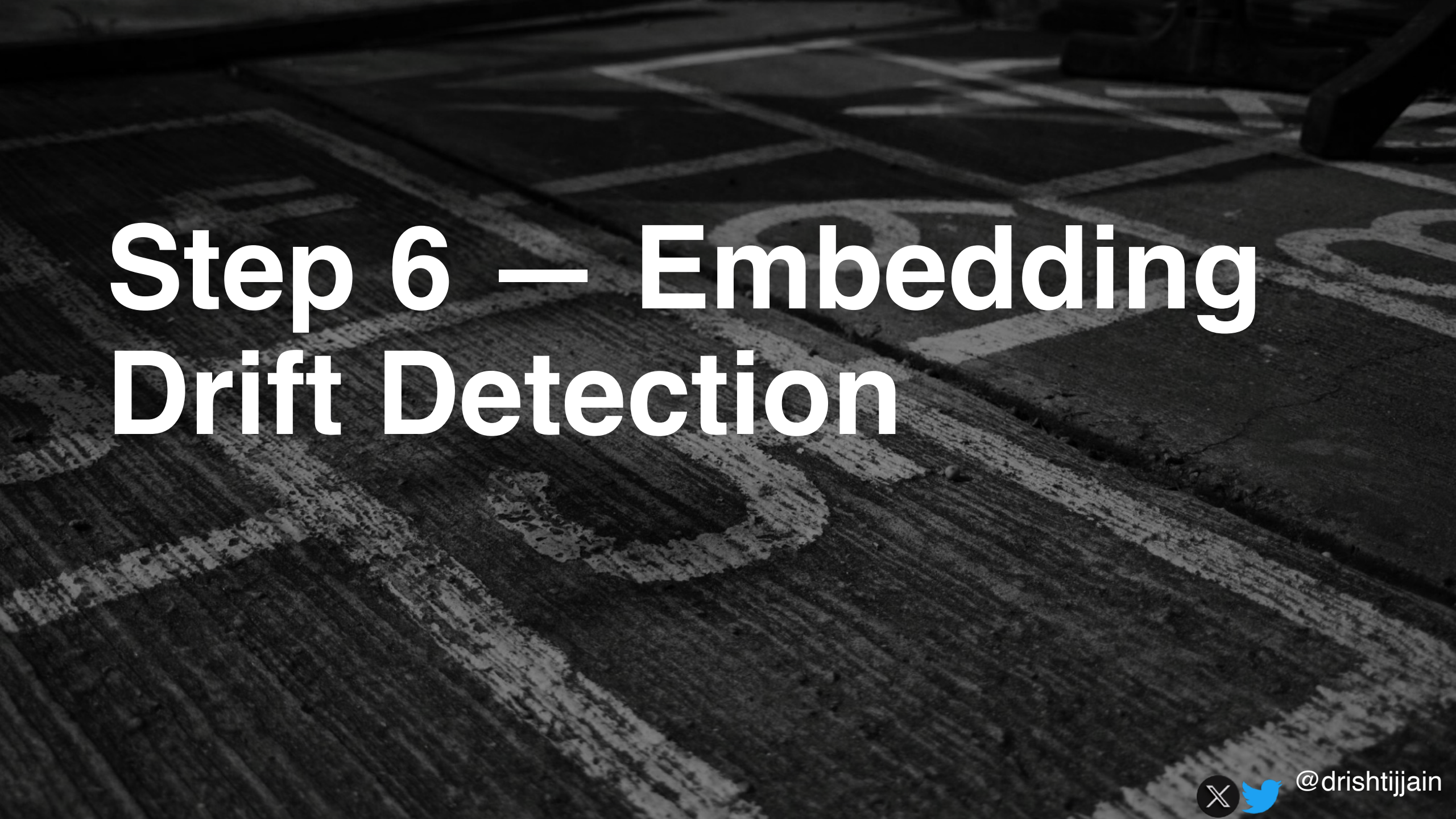
Reduce cost on OpenAI, Cohere, HuggingFace APIs

@drishtijjain

Architecture — Index Pipeline

Text > Chunker > Change Detector > Embedder > Vector DB

@drishtijjain


Step 6 — Embedding Drift Detection
@drishtijjain

Embedding Drift Detection

- Track cosine drift between old vs new embeddings
- Use alert thresholds

- **Why Drift Matters**
  - If drift > 0.3, retrieval quality drops
  - Re-ranking or retraining needed

@drishtijjain

```
# compare older embeddings distribution to new ones
old_embs = vector_db.query(... filter by metadata.model_ver=="2.2.1")
dist = average_cosine_distance(old_embs, new_embs)
if dist > 0.3:
    alert("embedding drift detected")
```

Drift Check

@drishtijjain

Step 7 — Retraining Workflow

@drishtijjain

Retraining Workflow

- Train new embedding model on KB
- Train reranker on Q&A pairs
- Maintain separate pipelines for:
  - Embedding model retrain on updated KB
  - Ranker training on user feedback + similarity data

@drishtijjain

Pseudo-framework using MLflow

```
# embedding retrain flow
steps:
  - extract: docs/ → chunks
  - embed_old = load_embeddings()
  - train: fine-tune new embedder
  - evaluate: compute embedding drift
  - publish model if performance ↑

# ranking flow
- generate training examples (query, relevant chunk)
- train a lightweight cross-encoder reranker
- version model & record metrics
```

@drishtijjain

Step 8 Freshness-Aware Retrieval

@drishtijjain

Freshness-Aware Retrieval

Combine relevance with recency

Weighted scoring: LLM score + time decay

@drishtijjain

Freshness-Aware Retrieval

```python
def fresh_retrieve(query, top_k=10, recency_lambda=0.5):
    q_emb = get_embedding(query)
    hits = vector_db.search(q_emb, top_k=top_k*2)

    scored = []
    now = time.time()
    for hit in hits:
        age = now - hit['metadata']['ts']
        recency_score = math.exp(-recency_lambda * age)
        score = hit['score'] * (0.7 + 0.3 * recency_score)
        scored.append((score, hit))
    scored.sort(reverse=True, key=lambda x: x[0])
    return [hit for _, hit in scored][:top_k]
```

@drishtijjain

Monitoring & Metrics

@drishtijjain

Monitoring & Metrics

Alert embedding drift

Metrics dashboard: counts of updated chunks, retrain logs, freshness score

User feedback loop: implicit signals (clicks, selections) → train “reranker”

@drishtijjain


Best Practices
@drishtijjain

Best Practices

- Hash per chunk
- Semantic diff
- Embed with version
- Score by recency
- Retrain as needed

@drishtijjain

Mistakes to Avoid

@drishtijjain

Mistakes to Avoid

- Blindly re-indexing entire corpus
- Ignoring embedding version
- No metadata tagging

@drishtijjain

THANK YOU!

@drishtijjain

linkedin.com/in/jaindrishti/

medium.com/@drishtijjain

@geekyearthian