credit courtroom - policy embedding and retrieval pipeline

Youcef Chalbi

January 2026

"'latex [11pt]article [a4paper,margin=1in]geometry hyperref listings xcolor amsmath enumitem

codegrayrgb0.95,0.95,0.95

backgroundcolor=codegray, basicstyle=, `breaklines=true, frame=single`

Credit Courtroom
Policy Embedding & Retrieval Pipeline Youcef Chalbi January 29, 2026

1 Overview

This document describes the work completed for the **Credit Courtroom** project regarding policy ingestion, chunking, embedding, storage, and semantic retrieval. The goal of this pipeline is to allow the system (judge agent) to retrieve relevant policy sections based on semantic similarity rather than keyword matching.

2 Problem Statement

Judicial decisions must be evaluated against internal credit and commercial loan policies. These policies are long and unstructured documents that make traditional keyword search insufficient. The objective is to enable **semantic search** on policy documents so that relevant sections can be retrieved automatically during decision review.

3 High-Level Architecture

[noitemsep]

- Policy documents are split into semantic chunks
- Each chunk is embedded into a vector representation
- Vectors are stored in Supabase using `pgvector`

- At runtime, queries are embedded and matched via cosine similarity
- Top-K policy sections are returned to the judge agent

4 Chunking Strategy

4.1 Purpose

Chunking ensures that embeddings represent semantically coherent sections rather than entire documents, improving retrieval precision.

4.2 Implementation

Policies were split by logical structure (sections / paragraphs). Each chunk is stored as a row in the `policy_chunks` table (used recursive chunking)

Stored fields:

[noitemsep]

- `id`
- `policy_name`
- `section`
- `content`
- `created_at`
- `embedding` (vector)

5 Embedding Model

5.1 Model Used

[noitemsep]

- **sentence-transformers/all-MiniLM-L6-v2**
- Embedding dimension: 384
- Cosine similarity compatible

5.2 Justification

This model was chosen due to:

[noitemsep]

- High-quality sentence-level semantic representations
- Low latency and small footprint
- Proven performance for retrieval-augmented generation (RAG)

6 Embedding Pipeline

6.1 Script: `embed_policies.py`

This script is responsible for embedding all policy chunks and storing them in Supabase.

6.2 Steps Performed

1. Load environment variables using `python-dotenv`
2. Initialize Supabase client
3. Load sentence-transformer model
4. Fetch all policy chunks
5. Generate embeddings for each chunk's content
6. Store embeddings in the `embedding` column

6.3 Key Logic (Simplified)

embedding = model.encode(content).tolist()
supabase.table(”policy$_{c}hunks”).update(”embedding” : embedding).eq(”id”, chunk_{i}d).execute()$

7 Vector Storage

7.1 Database Extension

The `pgvector` extension is enabled in Supabase to support vector operations.

7.2 Column Type

embedding vector(384)

8 Similarity Search

8.1 Search Strategy

Cosine similarity is used to retrieve the most relevant policy sections.

8.2 SQL Query

SELECT policy$_{n}ame, section, content, 1-(embedding <=> query_{e}mbedding) AS similarity FROM policy_{c}hun$
$query_{e}mbedding LIMIT 5;$

9 Validation

[noitemsep]

- Duplicate chunks were removed
- Empty content rows were skipped
- Retrieval results were manually validated for semantic relevance

10 Current Status

Completed:

[noitemsep]

- Policy chunking
- Embedding pipeline
- Vector storage
- Semantic similarity search

Remaining Work:

[noitemsep]

- Wrap retrieval logic in a service function
- Expose retrieval endpoint via FastAPI
- Connect judge agent decision output to retrieval endpoint

11 Conclusion

This pipeline establishes a robust semantic foundation for policy-grounded judicial reasoning. All computationally expensive steps (embedding) are performed offline, enabling fast and scalable real-time retrieval.