Advanced CSV Processing & Semantic Search Pipeline

Presented by

Welcome to this technical overview of an intelligent pipeline designed to tackle challenges in CSV data processing and semantic search, aimed at delivering scalable, accurate, and insightful data retrieval solutions for complex datasets.

Problem Statement: Challenges with CSV Data

The Challenge with CSV Data

- Manual & Inefficient: Traditional CSV processing struggles with complexities such as commas within values, varying delimiters, and multi-line entries.
- Poor Searchability: Simple keyword searches do not capture user intent or semantic meaning, resulting in irrelevant results.
- One-Size-Fits-None: Basic chunking techniques generate poorly sized text fragments, limiting retrieval relevance.
- Lack of Insight: Valuable metadata like sentiment or entity recognition within rows is often neglected, restricting advanced filtering capabilities.

This slide highlights fundamental pain points that hinder efficient data utilization from CSV files.

Tech Stack: Open-Source Foundations

All tools are open-source and installable via pip, except monitoring tools which run as separate services.

This stack supports scalable, maintainable, and extensible pipeline development.

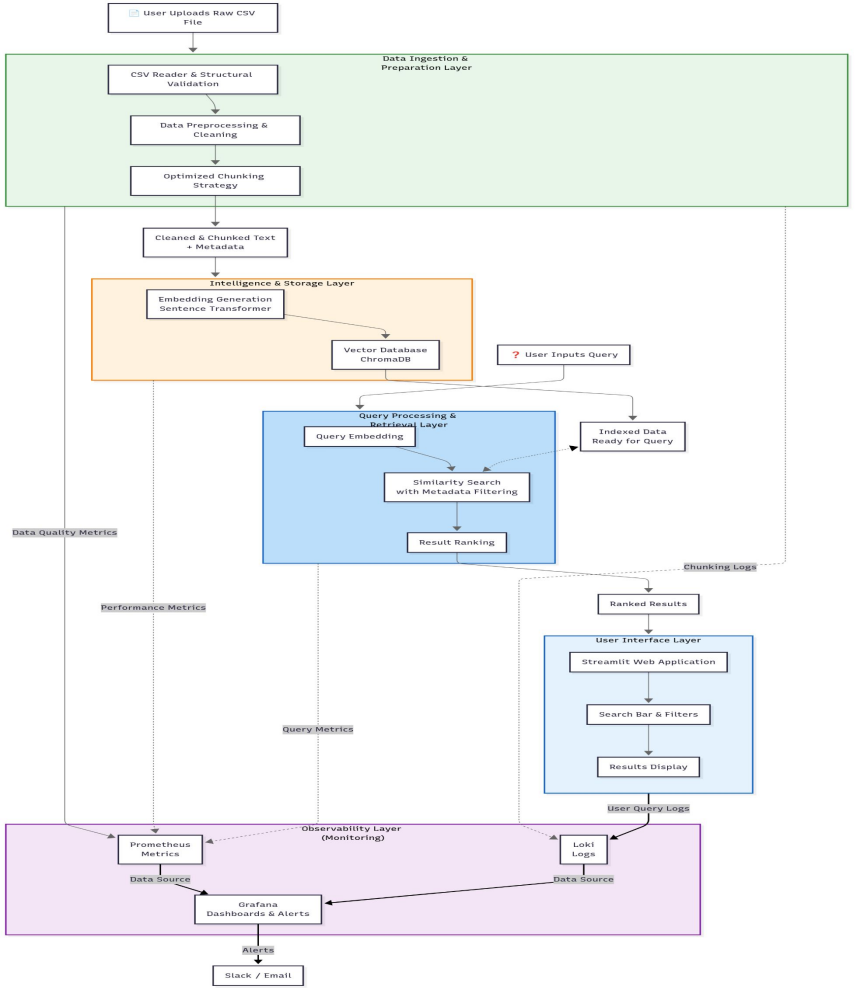
Component	Primary Tools
Core Framework	LangChain
Data Processing	Pandas, BeautifulSoup4
NLP & Enrichment	spaCy (NER), TextBlob (Sentiment)
Chunking	LangChain Text Splitters
Embedding Model	Sentence Transformers
Vector Database	ChromaDB
UI/API Framework	Streamlit
Monitoring	Prometheus, Loki, Grafana

Solution Overview: A Layered Intelligent Architecture

Automated Pipeline from Raw CSV to Semantic Search

- End-to-End Automation: Ingest, clean, chunk, and transform CSV data seamlessly.
- Semantic AI Integration: Converts raw text into a knowledge base that understands meaning for precise query results.
- Fast & Accurate Answers: Enables rapid retrieval of relevant information through intelligent processing layers.

This architectural overview sets the stage for detailed exploration of each processing layer.

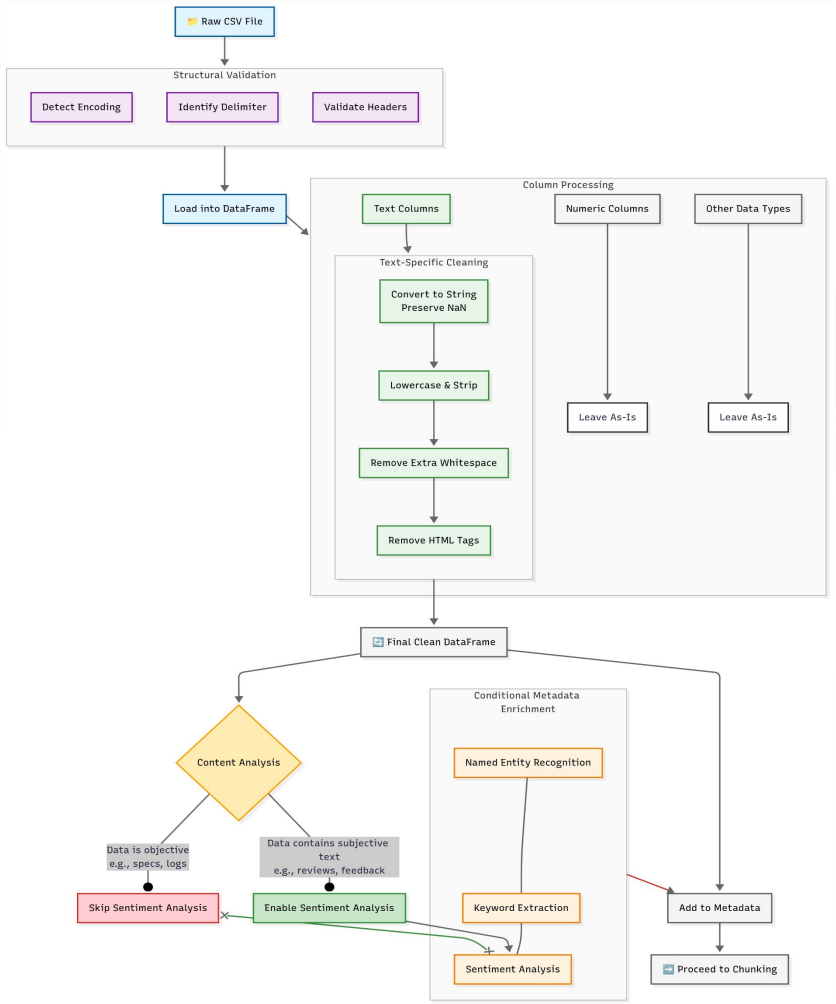


Architecture Layer 1: CSV Reader & Preprocessing

Purpose: Reliable Parsing and Cleaning of Raw CSV Data

- Robust Parsing: Handles complex CSV structures, including embedded commas and multi-line cells.
- Structural Validation: Auto-detects encoding, delimiters, and verifies header integrity.
- Smart Cleaning: Normalizes text (e.g., lowercasing, HTML stripping) while preserving numeric fields as metadata for downstream filtering.

This foundational layer ensures data quality and integrity for subsequent stages.



Architecture Layer 2: Optimized Chunking Strategies

Purpose: Breaking Text into Optimal Chunks for AI Retrieval

- Adaptive Strategies: Applies four chunking methods based on data characteristics:
  - Row-based
  - Cell-content
  - Multi-row
  - Parent-Child relationships
- Hybrid Approach: Simultaneously runs multiple chunking strategies to create an interconnected "web" of retrievable data.
- Metadata Preservation: Tags all chunks with rich metadata (source, attributes) to enable traceability and precise filtering.

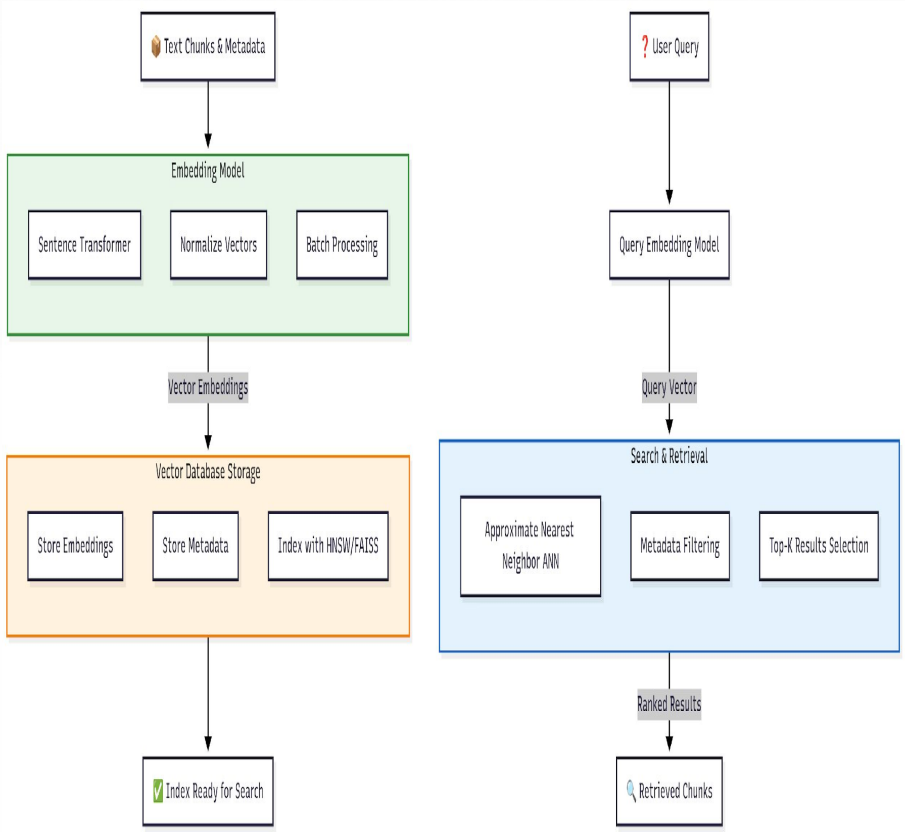
This layer balances chunk size and context to maximize search relevance.

Architecture Layer 3: Intelligence & Storage

Purpose: Semantic Understanding and Efficient Storage for Search

- Semantic Embedding: Uses Sentence Transformer models to encode text into vector representations capturing semantic meaning.
- Vector Database: Stores embeddings and metadata in ChromaDB/FAISS with HNSW indexing for Approximate Nearest Neighbor (ANN) search.
- Hybrid Search: Combines semantic vector search with metadata filters (e.g., "positive reviews for products under $100") for fine-grained querying.

This layer enables lightning-fast, context-aware retrieval to meet complex search demands.

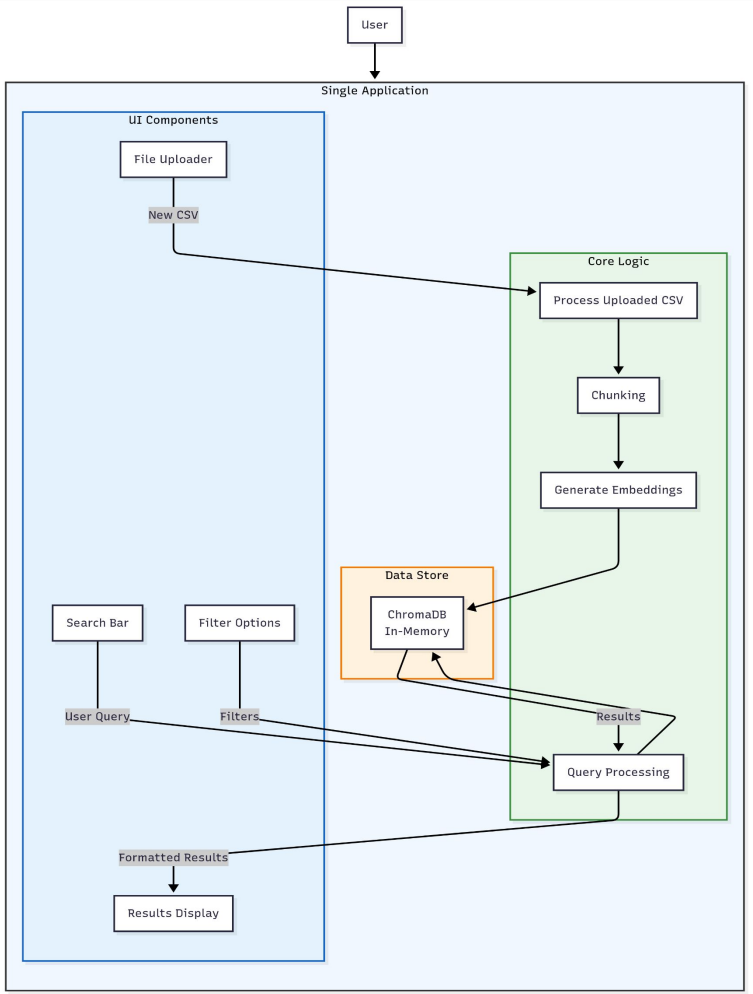


Architecture Layer 4: User Interface

Purpose: Simple, Powerful User Interaction

- Streamlit Web App: Clean UI for CSV upload and question entry.
- Real-Time Processing: Executes the full pipeline live within the app, from ingestion to embedding.
- Filtered Results: Presents ranked, semantically relevant answers with source context and filter options for enhanced user control.

This interface layer bridges technical complexity and user accessibility.

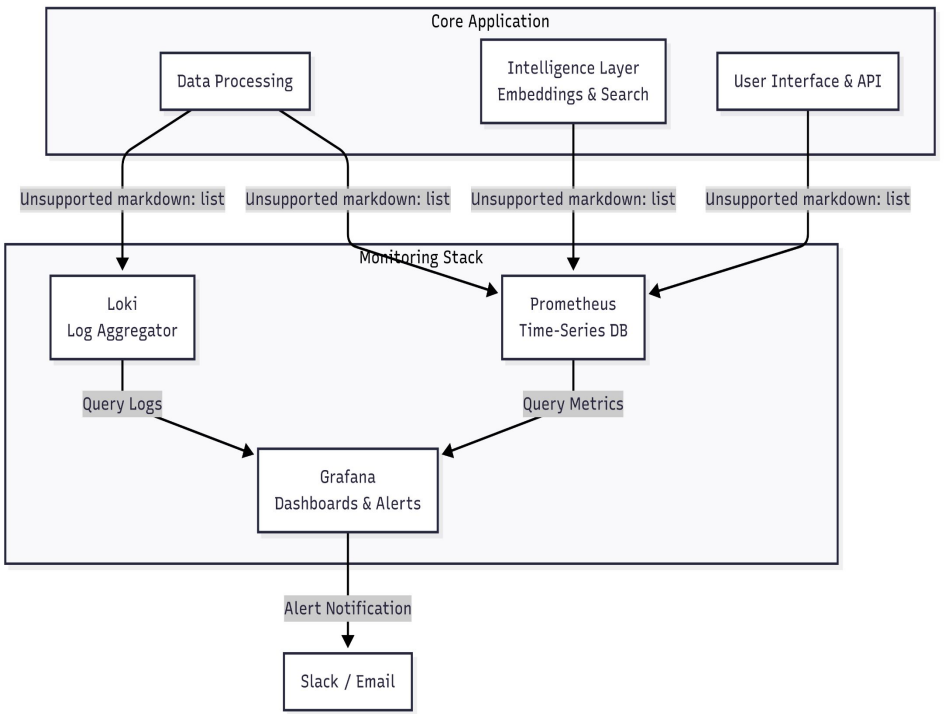


Architecture Layer 5: Monitoring & Observability

Purpose: Maintain System Health, Performance, and Quality

- Proactive Tracking: Uses Prometheus to monitor data quality, chunk sizes, query latency, and overall system health.
- Centralized Logging: Loki aggregates logs for debugging and auditing all pipeline stages.
- Actionable Alerts: Grafana dashboards visualize metrics and trigger alerts for anomalies such as slow queries or oversized chunks.

This ensures reliability and maintainability in production environments.



Sprint Goal – Week 1 – Foundation & Core Processing

Day	Task	Description	Deliverable
Day 1	Project Setup & Core Dependencies	Initialize project repository (e.g., Git). Create requirements.txt with libraries (LangChain, Pandas, Sentence-Transformers, ChromaDB, Streamlit). Set up virtual environment and install packages.	Working Python environment with dependencies installed. GitHub/GitLab repository.
Day 2	Robust CSV Reader & Validation	Implement CSV reading using Pandas. Add validation: auto-detect encoding, delimiter, validate headers. Handle errors gracefully.	Python function that returns cleaned DataFrame or useful error message.
Day 3	Text Preprocessing Pipeline	Build text cleaning functions: lowercase, strip whitespace, remove HTML tags. Separate text columns (to be cleaned) from numeric/categorical (stored as metadata).	Module with preprocessing functions returning cleaned DataFrame.
Day 4	Implement Row-Based Chunking	Develop chunking strategy: convert each row into formatted text (e.g., "ColumnName: Value, OtherColumn: Value").	Function that outputs list of text chunks from DataFrame.
Day 5	Implement Cell-Based Chunking	Use LangChain's RecursiveCharacterTextSplitter for long text columns (e.g., descriptions). Preserve row metadata for each new chunk.	Function to generate chunks from specific long-text columns.

Week 2 – Intelligence, Search & Interface

Day	Task	Description	Deliverable
Day 6	Embedding Generation & Storage	Initialize Sentence Transformer model. Create function to take list of text chunks, generate embeddings in batches, and store them in ChromaDB collection with metadata.	Script that processes chunks and populates ChromaDB vector store.
Day 7	Search & Retrieval Logic	Build core search function: (1) take user query, (2) generate embedding, (3) query ChromaDB for similar vectors, (4) return top-K results with text + metadata.	`Function search_query(query: str, filters: dict) returning ranked results.`
Day 8	Streamlit UI – Data Ingestion	Build UI components: file uploader + processing button. Trigger backend pipeline (Days 2–5) and store resulting vectors in ChromaDB (Day 6).	Web app where user uploads CSV and sees “Processing Successful” message.
Day 9	Streamlit UI – Search Interface	Build search bar + results display in Streamlit. Connect UI to search function (Day 7). Display results neatly, showing source text and metadata.	Web app where user asks a question and gets semantic search results.
Day 10	Debugging, Polish & Stretch Goals	Test pipeline with multiple CSVs. Fix bugs. Add loading indicators. Stretch goals: filter sidebar, better error messages, basic README for project.	Robust functional MVP + plan for future improvements.

Expected Outcome & Benefits

Delivering High-Impact Results

- High-Quality Search: Dramatically improved accuracy and relevance through semantic understanding.
- Adaptive Processing: Automatic selection of optimal strategies for diverse CSV data.
- Powerful Hybrid Queries: Complex user queries with precise metadata-based filtering.
- Production-Ready: Robust, observable, and maintainable system suitable for real-world deployment.

Thank you for your attention. Questions?