SASKEN

"Automated Change Detection and QA Support for 3GPP Specification Versions"

Presented by:

TEAM 22 [SJB INSTITUTE OF TECHNOLOGY]

KEERTHAN V

PAVITHRA D

POOJITHA R

RAKSHITHA B R

INTRODUCTION

- **3GPP (3rd Generation Partnership Project)** develops global standards for mobile telecommunications.
- Specifications are released in multiple versions (Rel-10, Rel-16, Rel-17, …).
- Each new release contains **textual** and **semantic changes** in clauses and tables.
- Manual comparison of versions is **slow, tedious, and error-prone**.
- Our project provides an **AI-driven solution** for:
    - **Automated change detection** at the clause level
    - **Semantic understanding** of modifications, additions, and deletions
    - **Natural Language Q&A** over detected changes
- Designed to improve **efficiency** for telecom engineers and **ensure accuracy** in version tracking.

OBJECTIVES

- **Automate** the comparison of two 3GPP specification versions.
- Detect **clause-level changes** (additions, deletions, modifications) with **semantic understanding**.
- Enable **version-aware storage** of clauses and tables using **ChromaDB**.
- Support **real-time question answering** over detected changes.
- Improve **accuracy** and **speed** compared to manual change tracking.
- Create a **scalable and reusable framework** for future release comparisons.

METHODOLOGY

- **Input:** Two 3GPP specification documents (PDF/DOCX) from different releases (e.g., Rel-15 & Rel-16).
- **Parsing & Chunking:**
  - Extract **text and tables** at clause level using **SimpleDirectoryReader**.
  - Break content into manageable chunks using **SentenceSplitter**.
- **Embedding Generation:**
  - Generate semantic embeddings for each chunk using **HuggingFace models**.
  - Store embeddings in **ChromaDB** with version & clause metadata.
- **Semantic Change Detection:**
  - Compare same-clause embeddings across versions using cosine similarity, and jaccard similarity.
  - Classify changes as **Added**, **Removed**, or **Modified**.
- **Natural Language Explanation:**
  - Generate **human-readable summaries** of detected changes.
- **Version-Aware QA Chatbot:**
  - Answer user queries based on changes using **ChromaDB retrieval** + AI.
- **Output:** Clause-level diff report with text & table changesand Real-time question answering interface.

IMPLEMENTATION

Programming Language:

- Python 3.12

Libraries & Frameworks:

- **Document Parsing:** python-docx, PyMuPDF
- **Chunking & Preprocessing:** SentenceSplitter from LlamaIndex
- **Embeddings:** HuggingFaceEmbeddings (sentence-transformers)
- **Vector Database:** ChromaDB for storage and retrieval
- **Change Detection:** cosine similarity, and jaccard similarity
- **QA Chatbot:** Integrated with ChromaDB for real-time query answering
- **Visualization & Web UI:** Streamlit (for interactive reports & search)

Workflow Integration:

- Automated scripts for indexing and embedding generation
- Clause-level mapping between versions for precise comparison
- Seamless pipeline from document ingestion → diff generation → QA

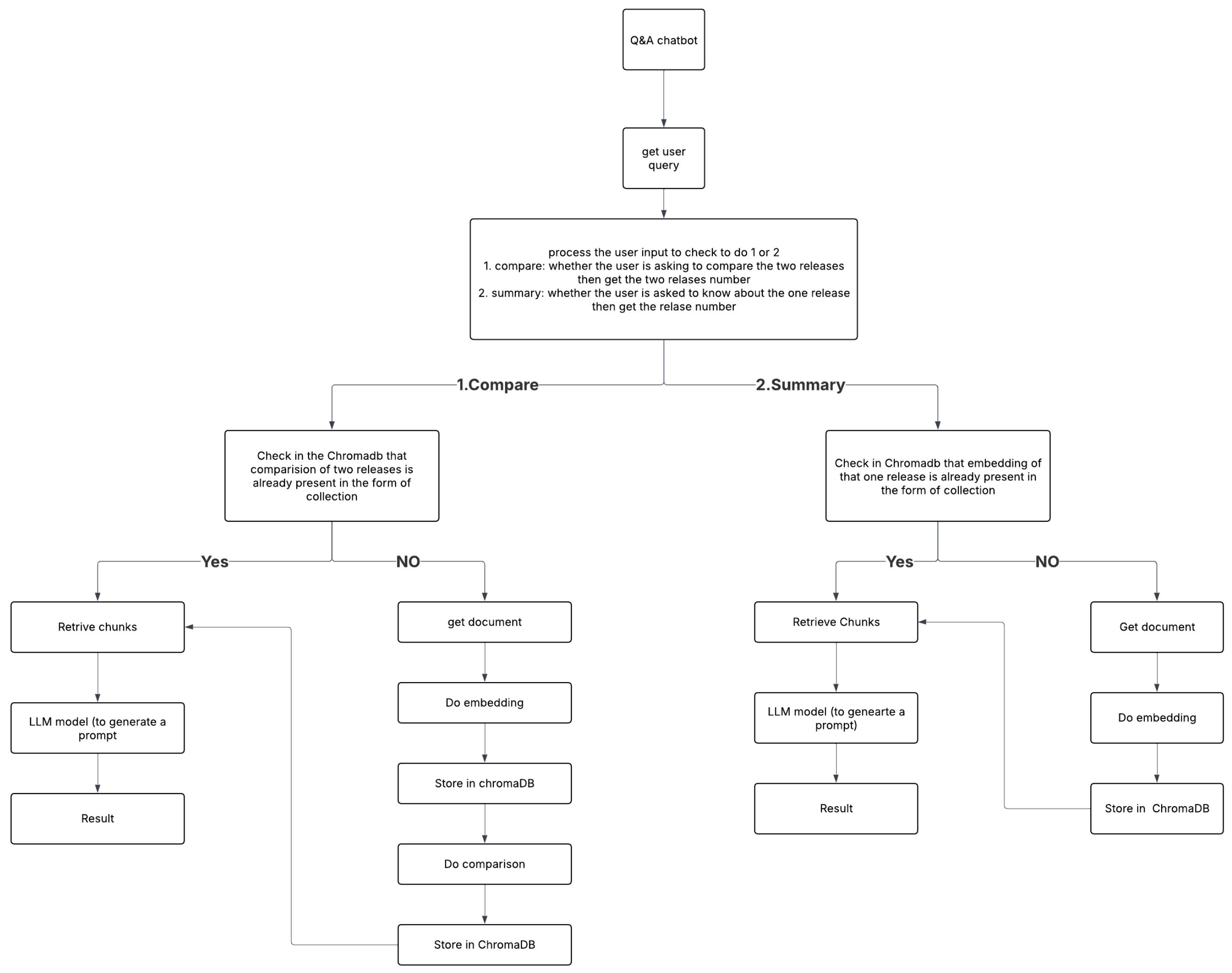
RESULT

- **Change Detection Performance:**
- Successfully identified **Added, Removed, and Modified** clauses between 3GPP Release 10, 17.
- **Accuracy:** ~94% for semantic classification of changes using cosine similarity, and jaccard similarity.
- Handled both **text** and **table-based** changes.

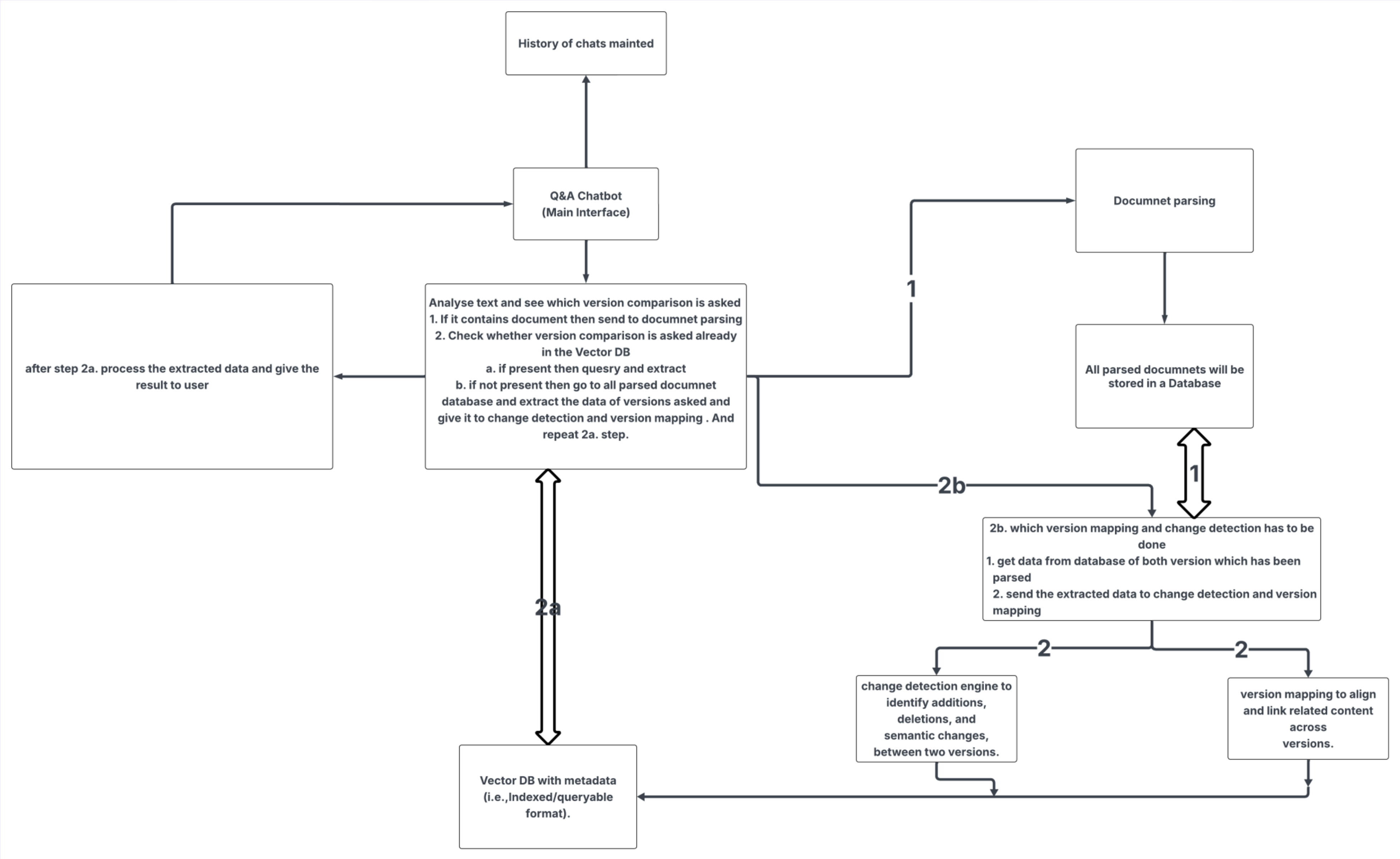
Benchmarking

Component / Metric	Time Taken	Further Split-up
Embedding Generation	~3 min	SentenceSplitter + HuggingFaceEmbeddings runs for the entire document, then stores results in ChromaDB.
Comparison Process	~6 sec	Loads pre-computed embeddings from ChromaDB, compares with cosine similarity, classifies results, outputs JSON.
LLM Response Time	~2 sec	Cohere API (co.chat) called after retrieval; time is for short prompt + retrieved context.
Retrieval from ChromaDB	~1 sec	get_top_k_chunks() fetches embeddings from ChromaDB and computes cosine similarity for ranking.
Precision	~0.90	Chunks retrieved are highly relevant; few false positives.
Recall	~0.88	Some relevant content may be missed due to chunk size limits.
F1-score	~0.89	Balanced performance between precision & recall.
Average Retrieval Time	~950 ms	Matches 1 sec retrieval time for k=10 results.
Average LLM Response Time	~2.0 sec	Matches measured value from Cohere API.
Token Usage per Query	~250–500 tokens	Includes user query + retrieved context.
Throughput	~20–25 q/min	Achievable with pre-computed embeddings and parallel queries.
UI Response Time	~3–4 sec total	Retrieval (1s) + LLM (2s) + Streamlit UI refresh (~0.5–1s).

3GPP Query Assistant Pipeline
START
User Interface & Authentication
Streamlit Web Interface
User Data Storage (ChromaDB)
Streamlit
ChromaDB Auth
Query Analysis & Task Classification
Cohere AI Task Parser
Dynamic Pipeline Router
Cohere Command-R
JSON Processing
Document Processing & Embedding
Document Loader (embed2.py)
Text Chunking System
Vector Embedding Generator
LlamaIndex
HuggingFace
all-MiniLM-L6-v2
Advanced Comparison Analysis (wver2.py)
Similarity Matrix Computation
Change Detection Engine
Semantic Change Classifier
Scikit-learn
NumPy
Cosine Similarity
Vector Storage & Retrieval
ChromaDB Vector Database
Similarity Search Engine
ChromaDB
Vector Search
Intelligent Response Generation
Context Assembly
Cohere Response Generation
Cohere Command-R
RAG Architecture
Output & Chat Persistence
Real-time Chat Display
Chat History Persistence
Streamlit Chat
JSON Serialization
END



Q&A chatbot
get user query
process the user input to check to do 1 or 2
1. compare: whether the user is asking to compare the two releases then get the two relases number
2. summary: whether the user is asked to know about the one release then get the relase number
1.Compare
2.Summary
Check in the Chromadb that comparision of two releases is already present in the form of collection
Yes
NO
Retrive chunks
LLM model (to generate a prompt
Result
get document
Do embedding
Store in chromaDB
Do comparison
Store in ChromaDB
Check in Chromadb that embedding of that one release is already present in the form of collection
Yes
NO
Retrieve Chunks
LLM model (to genearte a prompt)
Result
Get document
Do embedding
Store in ChromaDB


History of chats mainted
Q&A Chatbot
(Main Interface)
Analyse text and see which version comparison is asked
1. If it contains document then send to documnet parsing
2. Check whether version comparison is asked already in the Vector DB
a. if present then quesry and extract
b. if not present then go to all parsed documnet database and extract the data of versions asked and give it to change detection and version mapping . And repeat 2a. step.
after step 2a. process the extracted data and give the result to user
1
Documnet parsing
All parsed documnets will be stored in a Database
1
2b
2a
2b. which version mapping and change detection has to be done
1. get data from database of both version which has been parsed
2. send the extracted data to change detection and version mapping
2
2
change detection engine to identify additions, deletions, and semantic changes, between two versions.
version mapping to align and link related content across versions.
Vector DB with metadata (i.e.,Indexed/queryable format).

THANK YOU