CONVOLVE 4.0

A Pan-IIT AI/ML Hackathon

Track: Multi Agent System

Presenting Solution for:

Misinformation Detection in Indian Elections

A Graph-based AI Agent for detecting election misinformation, verifying EVM/VVPAT claims, and providing personalized assistance to election officials and citizens using Visual Forensics, detailed vector search and Persistent Memory.

Powered By:

Qdrant as a vector database and memory database.

LangGraph, LangChain as an framework to build MAS.

Presented By:

Keshav Bhatt

B.Tech CSE

Central University of Jammu

Problem Statement

What social are you addressing ?

Misinformation, fake claims, myths are very harmful for an organization and society, when it comes to Indian Elections it is very normal to spread myths and falsie claims such as *EVMs can be hacked, or Indelible ink being used by the Election Commission of India is produced from the fat of an animal and dissuaded people from voting during Lok Sabha Elections* and so many other disinformation is spread by twitter handles, YouTube videos and magazines.

Why Does it matter?

- Indian Election system has to face lot of problems due to these misinformations and conduct of election becomes difficult, it becomes a news highlight rather than a democratic process.
- People carrying and believing such info loose their vote and could not choose right leader.
- One major issue is this type of misinformation spreads very quickly through WhatsApp and other social media platforms.
- Normal person can not find the correct information in huge ECI manuals.
- Also officials or professionals of elections such as Presiding officer, Counting Agent has very long manuals and reading them to correct, sometimes feels impossible.

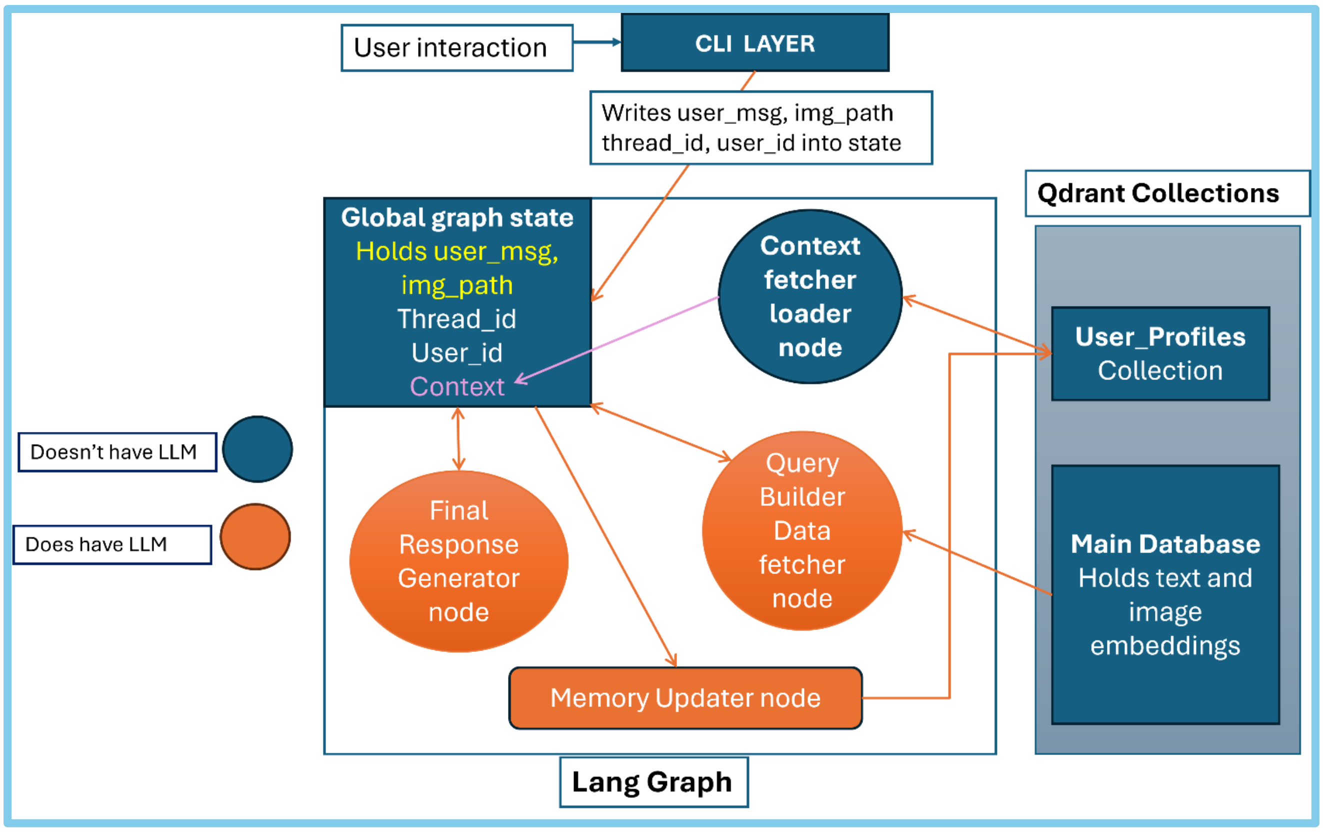
Election Commission of India keeps the record of disinformation and correct information and provide it via https://mythvsreality.eci.gov.in/ but it lacks in giving personalized answers and information from official pdfs, and also it is a static source which does not get updated regularly, even though government officers keep doing nullify such false claims.

- Here does our solution provide easy detection of misinformation because it has all the official data scraped from pdfs and press releases of government, twitter known fakes and myths which are busted by some officials, directly scraped from twitter itself.
- It supports visual recognition so just by uploading images and related claim or without claim it will give the correct information and also correct you if you are wrong.
- Reach payload schema provides you detailed source information and evidence urls, videos, images.
- Long term memory is able to give you your personalized answers and depth of knowledge at which you like.

System Design

Architecture Overview:

The stated Misinformation detection system should have high capability of generating accurate results and user personalized results which is not possible with single agent so here is the multi-agent workflow to solve the problem.



System Design for Misinformation Detection System

Workflow

1. User enters its user name which creates an UUID for each unique user, we will use it to get the payload which stores **Long Term Memory** of user in Vector Database's User_Profile Collection.
2. After authentication user get a Thread_id which for which Langgraph makes an unique global state from which any node can read any node can write to, it is used to handle **Session-based memory** of user because it stores all the chat and responses too, because at the end response will be written to it.
3. User message and attached image_path (if available) is stored into the **global state**, so finally our state stored all the necessary information about message, thread.

4. Now the first node which works is Long term Memory retrieval node/ Context Fetcher and loader node, it uses the user_id hash (UUID) to get the **Point of that user from Qdrant**, even though this **Point has dense_summary_vector and sparse_summary_vector** but we do not use any type of search to get the payload, **UUID makes it very fast**.
5. If the user is found so this node loads whole payload to the global state, which is used to store user's phycological profile and summary of conversation (not chat history). It looks like this:

```json
{
  "user_id": "kb",
  "name": "Radhe",
  "location": "Jammu",
  "persona": "Citizen",
  "interaction_style": "Informative",
  "content_preferences": {
    "show_twitter": true,
    "show_urls": true,
    "show_actions": true
  },
  "summary": "Radhe is a new user from Jammu who is interested in verifying visual content related to political processes, specifically election procedures. They shared an image they initially misinterpreted as depicting politicians fighting in a vote counting hall, but were open to receiving a detailed explanation with official sources to correct the misinformation. They appear to be a concerned citizen seeking accurate information."
}
```

Further workflow is divided into following sections.

Why Qdrant is essential:

- As a vector database it served really well**, sematic search, named vectors, sparse search, indexed payload fields** for fast filtering are some use cases that I personally liked most.
- In my architecture it work for data and memory too, so without this it wouldn't be possible because it has amazing tutorial series on YouTube, which I think other database does not provide.
- Since our system handles complex data types (text and potentially images), **Qdrant natively manages the vector embeddings generated by our encoder models**, allowing for unified storage of diverse data modalities.
- **The qdrant-client** library provides a robust, **Pythonic interface** for managing collections, points, and snapshots, It features first-class **support for modern LLM frameworks (LangChain, LlamaIndex), simplifying the implementation of RAG (Retrieval-Augmented Generation) pipelines.**

Multimodal Strategy:

What data types are used:

We used two datatypes images and text and embedded three type of information

1. Disinformation and Information pairs
2. Huge extracted text from official pdfs with proper chapter name, page number and section title
3. Images, image and its description both got embedded for including the capability of text search for images.

More info on this is just below.

How embeddings are created

Example Data/Payload of Image:

```
{
  "payload": {
   "record_type": "official_visual_truth",
   "media_type": "image",
   "category": "Polling Station essential",

   "title": "SET UP OF POLLING STATION FOR SINGLE ELECTION",

   "visual_concepts": [
    "Voting Compartment",
    "Enough Space for Voters",
    "Seperate entrance and exit for voters"
   ],

   "description": "Diagram of Model Polling Station Showing the layout when the polling party consist of 3 .........

   "source_name": "ECI Manual on Handbook for Polling Officer",
   "source_url": "https://www.eci.gov.in/eci- ",
   "page_number": 13,
   "image_url": "https://github.com/Keshav-CUJ/Qdrant-convole/raw/main/images/Layoutofpollingstation.png",
   "trust_score": 1.0,

   "actionable_intent": "educate_procedure",
   "agent_guidance": "If user claims that there is no proper layout of polling station, explain that this is the model layout of polling station as per the ECI Manual."
  }
}
```

In this payload embedded fields are **image and long description** which will definitely go through chucking process but **image get easily embedded into 512D vector**, it is embedded by **openAI's *clip-ViT-B-32 model*** which outperforms in case of images. Image's description is embedded similar as of other non-image payloads.

Example Data/Payload of PDF-informatio:

```
{
  "payload": {
    "record_type": "official_truth",
    "source_url": "https://mythvsreality.eci.gov.in/details/evm",
    "source_name": "ECI Myth vs Reality",
    "trust_score": 1.0,
    “chunk”: “There is no need to air condition the room/hall where EVMs are stored.”
    “chunk_index”: 1
    “Is_chuncked” : true
    "original_text_content": "There is no need to air condition the room/hall where EVMs are
stored. What is required is only to keep………………………………”
    "category": "Information on EVM & VVPAT",
    "topic_tags": [
      "EVM Electricity dependency",
      "EVM manufacturing"
    ],
    "evidence_media": [],
    "actionable_intent": "Spread_correct_info"
  }
 }
```

Semantic Chunking of LONG-PDF-TEXT

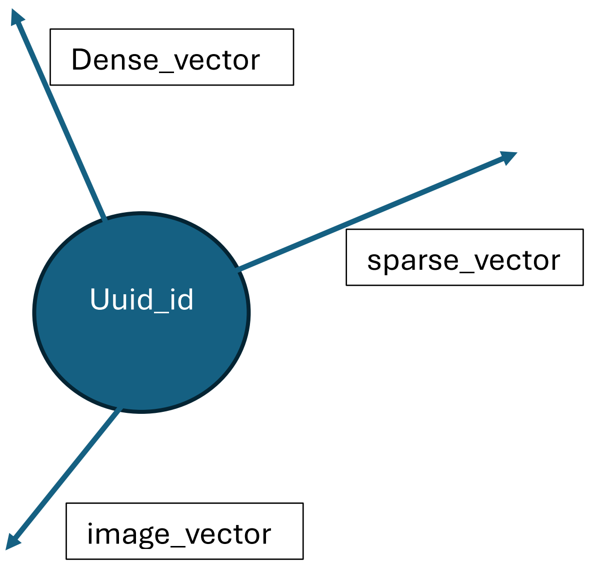
As you can see the above payload is result of chunking process, because the model that is used to embed the text is ***intfloat/multilingual-e5**-base* which expects 512 token as input, and out original_text_content is too large which not fits in **512 tokens.**

So we used best chunking method **Semantic Chunking** with the help of same embedding model and **llama_index's SemanticSplitterNodeParser** which run the embedding model to decide where to chunk semantically, but before this we checked exceeded token counts with the help of Tokenizer of same model. After chunking the chunks get embedded and goes under **dense_vector** named vector, similarly the image long description also.

Sparse vector creation

We created them for **keyword based search** and fast search and also for implementing hybrid search. They are created from chunks itself so we don't have to worry about **Avg_Doc_length**, using ***Qdrant/bm25*** model and we set the ***modifier=models.Modifier.IDF*** for automatic calculation of IDF scores.

So finally our single point look like this in **vector space of Qdrant:**



How embeddings are queried

It is the role of Query builder and data fetcher node, This node creates a list of tools to calls in following format:

```
[
  {{
    "tool": "search_image",
    "query": "path/to/image.jpg", //It will be used as path so please keep the input path of image.
    "filters": {{"category": "Vote Counting essential"}}, //in case of image you have only one filter i.e. category
    "purpose": "Identify the object"
  }},
  {{
    "tool": "search_hybrid",
    "query": "EVM hacking bluetooth 2024",
    "filters": {{"category": "Busted fake news", "topic_tags": ["Election scams"]}}, //in case of text you can use multiple filters
    "purpose": "Verify the text claim"
  }}]
```

And its helper function call the nodes tools get top 5 similar results.

In case of hybrid searching **dense_search() and sparse_search()** takes place and then result is combined with the help of **Recursive Rank Fusion technique**. This node will put the payloads of similar results in the global state as documents, payload.

There are three named vectors for each point

- **Dense_vector //768 dimensional text vector [intfloat/multilingual-e5-base]**
- **Sparse_vector //Varying length [Qdrant/bm25]**
- **Image vector //512 dimensional image vector [clip-ViT-B-32]**

When a payload which does not have any image to embed it gets **only two text vector but payload which has image to embed gets three vectors**

- **Image vector for similarity search via images**
- **Dense_vector for semantic similarity search (The description of image is embedded in this vector.)**
- **Sparse_vector for keyword based search (For making search faster.)**

In this way data retrieval node provide very detailed information with fields like evidence URLs, video URLs, tweet URLs, myth, correct information.

Search / Memory / Recommendation Logic

How retrieval works

Dense Search -> It does use **brute force search** takes time but gives more semantically correct information.

Sparse Search -> It does **keyword based search** gives faster results but average quality.

Hybrid Search-> Reranks the output of both searches and gives **optimal searches**.

The one more reason to implement each search is user preference if user want the fast search so our retrieval node will call sparse search only, if detailed so it will be do dense search,

This is the memory schema

```
{{  "name": "...",
   "location": "...",
   "persona": "...",
   "interaction_style": "...",   // this field store the type of search if specified
   "content_preferences": {{  //default is true for all.
      "show_twitter": true,
      "show_urls": true,
      "show_actions": true
   }},
   "summary": "Updated narrative summary..."
}}
```

Indexing the payload fields for faster filtering

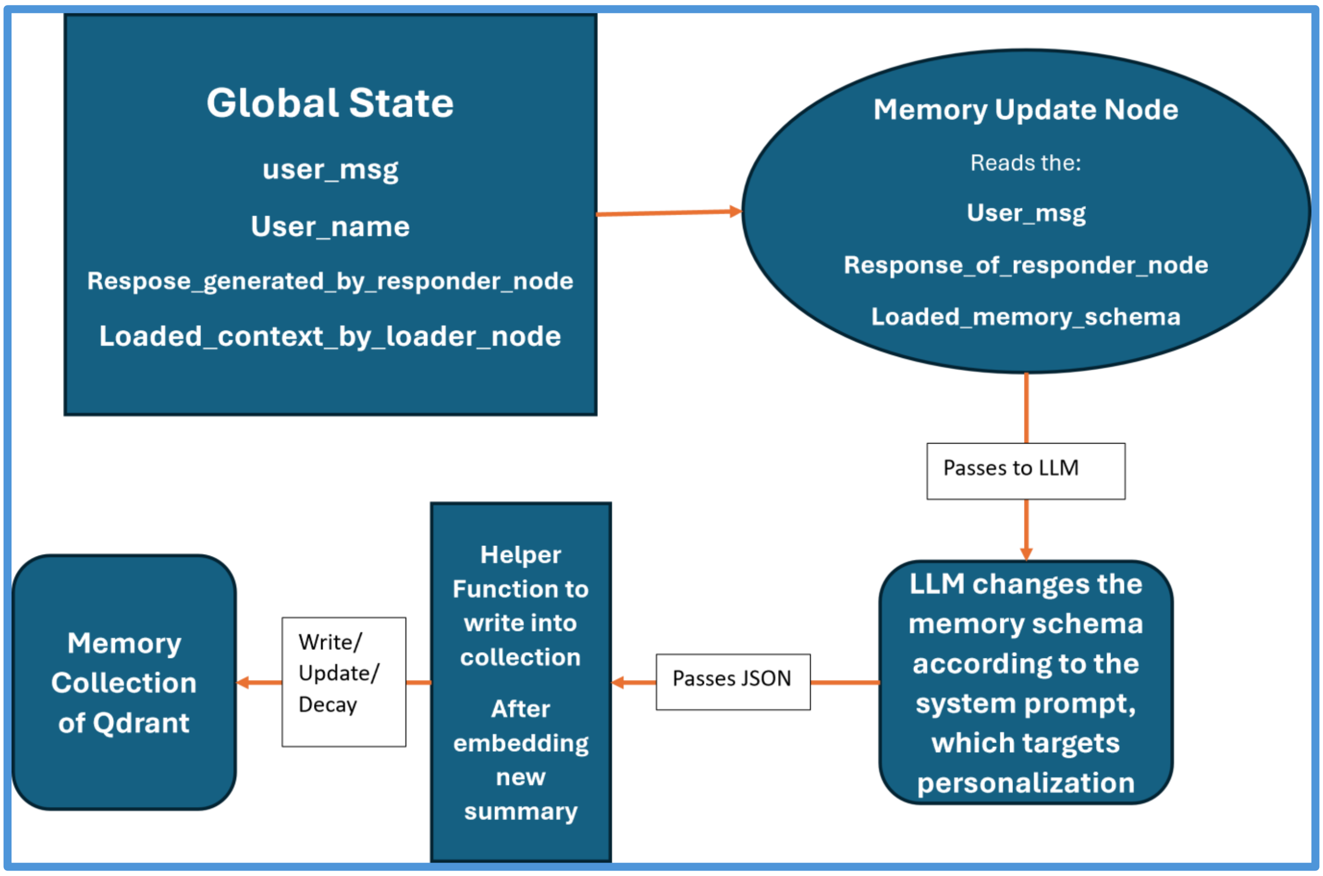
As we know Qdrant does filtering faster for indexed payload fields so we indexed following fileds

- **Category (Keyword)**
- **Trust_score (float)**
- **Topic_Tags (List)**

And filter builder helper function can do **AND operation in filters**.

How memory is stored, updated, and reused

LTM of user is stored in the user_profile collection of Qdrant in which each point has summary as text vector in both sparse and dense style, but **we don't use them we get a point with the help of uuid created from user name.**


Global State
user_msg
User_name
Respose_generated_by_responder_node
Loaded_context_by_loader_node
Memory Update Node
Reads the:
User_msg
Response_of_responder_node
Loaded_memory_schema
Passes to LLM
LLM changes the memory schema according to the system prompt, which targets personalization
Passes JSON
Helper Function to write into collection
After embedding new summary
Write/ Update/ Decay
Memory Collection of Qdrant

Memory Update/ Create/ Delete logic

Limitations & Ethics

Known Failure Modes

- **Bias:** The system relies on pre-trained embedding models, which may carry inherent demographic or cultural biases that skew retrieval results.
- **Privacy:** There is a risk of inadvertently retrieving and sending PII (Personally Identifiable Information) stored in long-term memory to third-party LLM providers.
- **Safety:** The database is vulnerable to "memory poisoning," where malicious prompts stored in history are retrieved later to bypass guardrails.

And with this, it is our Misinformation detection system that detects misinformation in election related data, have LTM support, session based memory, multimodal capability, recommendation capability and all required things.