**Overview**

This system enables:

1. **Collection of Telegram chat data** (messages, media, polls) using Telethon and MongoDB as the backend.
2. **Storage & enrichment** of messages (including images, audio, and polls) in MongoDB and GridFS.
3. **Analysis pipeline** that:
   * Summarizes conversations.
   * Classifies chat topics using a fine-tuned HuggingFace model.
   * Extracts captions from images (vision model).
   * Transcribes audio/video files (Whisper).

It consists of two main scripts:

* **telegram\_mongo\_poller.py** → Collects and stores messages/media into MongoDB.
* **telegram\_llm\_analyzer.py** → Analyzes stored data using ML/NLP pipelines.

**⚙️ telegram\_mongo\_poller.py**

**Purpose**

A **polling/streaming collector** that:

* Connects to Telegram via Telethon.
* Tracks chosen chats interactively.
* Fetches old messages or listens to new ones in real-time.
* Persists data to MongoDB (messages collection + GridFS for media).

**Key Components**

**MongoDB Setup**

* **Database**: telegram\_data
* **Collection**: messages
* **GridFS**: Used for binary media (photos, documents, audio/video).

**Chat Selection**

* User manually selects chats from a numbered list (choose\_chats()).
* Supports fetching historical messages by:
  + All messages.
  + Last N messages.
  + Messages from last X days.

**Message Saving**

save\_message() function:

* Normalizes a Telegram message into a MongoDB document.
* Handles:
  + Text messages.
  + **Polls** → parsed into poll\_data with question, answers, votes.
  + **Photos** → downloaded as bytes, stored in GridFS, reference stored in file\_id.
  + **Documents (audio/video/files)** → downloaded to GridFS, MIME type saved.

Example document structure in messages:

{

"message\_id": 12345,

"chat\_id": -987654321,

"chat\_name": "Test Group",

"sender": "john\_doe",

"date": "2025-08-29T12:34:56",

"text": "Check this out!",

"media\_type": "photo",

"file\_id": "photo\_12345.jpg",

"poll\_data": null

}

**Poll Updates**

* Handles UpdateMessagePoll and UpdateMessagePollVote events.
* Refreshes stored poll data in MongoDB.

**Event Handlers**

* events.NewMessage: Saves new messages (if in selected chats).
* /menu command → reloads chat selection and re-fetches history.
* events.Raw: Updates polls in real-time.

**🤖 telegram\_llm\_analyzer.py**

**Purpose**

A **data enrichment and intelligence pipeline** that:

* Loads previously collected Telegram data from MongoDB.
* Extracts insights from chats (summarization, classification, captions, transcripts).
* Produces flagged summaries for each chat.

**Key Components**

**MongoDB Setup**

* Reuses telegram\_data.messages collection.
* Reuses GridFS for media lookup.

**HuggingFace & Whisper Models**

* **Summarizer**: t5-small for short abstractive summaries.
* **Classifier**: Fine-tuned model from ./model\_output.
  + Custom tokenizer padding fix ([PAD] added if missing).
  + Runs on GPU with FP16 if available.
* **Image Captioning**: nlpconnect/vit-gpt2-image-captioning.
* **Speech-to-Text**: [OpenAI Whisper](https://github.com/openai/whisper?utm_source=chatgpt.com) (base model).

**Media Handlers**

* **process\_photo(file\_id)**:
  + Reads photo from GridFS.
  + Resizes to 224×224 for faster inference.
  + Generates a caption via the image captioning pipeline.
* **process\_audio\_video(file\_id)**:
  + Reads file from GridFS.
  + Writes to temporary file.
  + Runs Whisper to transcribe.
  + Deletes temp file after use.

**Chat Analyzer**

* Groups messages by chat\_name.
* Enhances text with:
  + [Image Caption] …
  + [Transcript] …
* For each chat:
  + **Summarization**: Abstracted overview of conversation.
  + **Classification**: Topic detection (custom fine-tuned classifier).
  + **Output**: Summary + classification + confidence score.

Example output:

💬 Chat: CryptoGroup (250 msgs)

📝 Summary: Group discussing new token launch, price volatility, and NFT hype.

🚩 Classified as: finance (0.943)

------------------------------------------------------------

**🔄 Data Flow**

1. **Polling Script (telegram\_mongo\_poller.py)**
   * User selects chats.
   * Old + new messages fetched → stored in MongoDB/FS.
2. **Analysis Script (telegram\_llm\_analyzer.py)**
   * Pulls recent messages.
   * Enhances with captions/transcripts.
   * Summarizes & classifies chat conversations.
   * Outputs structured intelligence.

**🛡️ Design Choices & Rationale**

* **MongoDB + GridFS**: Handles both structured (text, metadata) and unstructured (binary media) data efficiently.
* **Separate Scripts**:
  + Collector → lightweight, only storage.
  + Analyzer → heavier (ML pipelines).
  + This separation prevents blocking and simplifies scaling.
* **Explicit Tokenizer Handling**: Avoids HuggingFace issues where pad\_token == eos\_token.
* **GPU Acceleration**: Optional FP16 for performance but gracefully falls back to CPU.
* **Summarization before classification**: Prevents classifiers from handling extremely long texts.

**🚀 How to Run**

**1. Start Poller**

python telegram\_mongo\_poller.py

* Select chats.
* Choose history fetch mode.
* Script runs indefinitely to store new messages.

**2. Run Analyzer**

python telegram\_llm\_analyzer.py

* Pulls last 500 messages (default).
* Outputs summaries and classifications per chat.