iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

AI YOUTUBE VIDEO SUMMARIZER FOR INTELLIGENT KNOWLEDGE EXTRACTION FROM VIDEO CONTENT

Masik Ahamed M
Research Scholar, VISTAS, Chennai
Dr. Sangeetha Radhakrishnan
Assistant Professor, VISTAS, Chennai

ABSTRACT:
The AI YouTube Video Summarizer is a full-stack, AI-powered web application designed to intelligently process, analyze, and summarize YouTube video content using Google Gemini large language models. The system supports a dual-architecture deployment — a Streamlit-based Python frontend for rapid use and a Flask REST API paired with a React + Vite frontend for production environments. Six distinct output modes are provided: Short Summary, Detailed Summary, Full Explanation, AI-generated timestamped chapters, full transcript download, and a Retrieval-Augmented Generation (RAG) chatbot for interactive Q&A over video content. The RAG pipeline uses ChromaDB for in-memory vector storage and Gemini embeddings for semantic similarity search, enabling factually grounded, hallucination-free answers. A mind map generation feature produces hierarchical Mermaid.js diagrams for visual knowledge representation. User acceptance testing with 10 participants confirmed an average satisfaction score of 4.6/5 across all features, with summarization latency under 8 seconds on the gemini-flash-latest model. The system demonstrates how open-source, zero-infrastructure-cost LLM tooling can deliver enterprise-grade video intelligence at no subscription cost.

Keywords:
AI YouTube Summarizer, Google Gemini, RAG, ChromaDB, NLP, Streamlit, Flask, React, LLM, Video Summarization, YouTube Transcript API, Mermaid.js, Mind Map

I. INTRODUCTION

The exponential growth of video content on platforms such as YouTube has created a significant information overload problem. As of 2024, over 500 hours of video are uploaded every minute, making manual consumption of all relevant content practically impossible. Students, researchers, and professionals routinely spend hours watching lengthy videos to extract a few minutes of actionable information. Traditional content discovery tools — search engines, bookmarks, and note-taking applications — are not designed for video media and provide no higher-level understanding or summarization capability.

The AI YouTube Video Summarizer addresses this gap by providing a lightweight, full-stack web application that accepts a YouTube URL, retrieves its transcript via the YouTube Transcript API, and generates high-quality summaries using Google Gemini large language models. The system supports six output modes within a single, immediately deployable platform.

The contribution of this work is threefold: (1) a practical AI-powered video summarization platform with six distinct output modes requiring only a Google Gemini API key; (2) a Retrieval-Augmented Generation (RAG) chatbot enabling grounded, context-aware Q&A over video transcripts using ChromaDB vector storage; and (3) a dual-architecture design combining a Streamlit Python frontend with a Flask REST API and React frontend, serving both rapid prototyping and production deployment needs.

II. LITERATURE REVIEW

2.1 AI in Text Summarization

Automatic text summarization has a long history in NLP, with extractive methods (selecting sentences verbatim) and abstractive methods (generating new text) as the two principal paradigms. Nallapati et al. (2016) demonstrated that sequence-to-sequence neural models outperform extractive baselines on the CNN/DailyMail benchmark. The advent of transformer architectures (Vaswani et al., 2017) and pre-trained models such as BERT and BART substantially raised the state of the art in abstractive summarization.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

2.2 LLMs for Video Content Understanding

Kasneci et al. (2023) surveyed LLM applications in education, noting their particular strength in generating personalized, context-aware feedback from unstructured text. Google Gemini Flash was selected for this project due to its 1-million-token context window, sub-second inference latency, and free-tier availability — making real-time in-browser summarization feasible without infrastructure cost.

2.3 Retrieval-Augmented Generation

Lewis et al. (2020) introduced RAG as a paradigm for combining parametric LLM knowledge with non-parametric retrieval from a document store, substantially reducing hallucinations on knowledge-intensive tasks. This project operationalizes RAG using ChromaDB as an in-memory vector store and Gemini embeddings for semantic similarity, enabling factually grounded answers to user questions about video content.

III. EXISTING SYSTEM

Several commercial tools attempt to address video summarization: Summarize.tech, NoteGPT, Eightify, and Glarity. These tools share common limitations: high subscription costs restrict student access; none provide open-source, self-hostable deployments; interactive Q&A with video content via RAG is absent; visual mind map generation is unavailable; and transcript download is unsupported in most tools. Furthermore, all existing solutions rely on proprietary APIs with tight usage caps, creating barriers for academic and independent use.

Tool	Approach	Limitation
Summarize.tech	GPT-3 summarization	Paid, no chat, no timestamps
NoteGPT	GPT-4 summaries + notes	No RAG chat, subscription
Eightify	AI bullet summaries	No full explanation, no mind map
Glarity	Browser extension LLM	Requires paid API key

Table 1: Comparison of Existing Video Summarization Tools

IV. PROPOSED SYSTEM

4.1 Architecture Overview

The AI YouTube Video Summarizer follows a dual-architecture design. The Streamlit path (app.py) provides a self-contained Python application launched with a single command: *streamlit run app.py*. The production path comprises a Flask REST API backend (backend/api.py) consumed by a React + Vite + Tailwind CSS frontend (frontend/src/). Both paths share the core src/ module library for video retrieval, AI inference, and RAG operations. No database server is required — ChromaDB operates in-memory and Flask uses Python dictionaries for session state.

Component	Technology	Role
Frontend — Streamlit	Python, Streamlit	Rapid-deploy UI
Frontend — React	React 18, Vite, Tailwind	Production UI
REST API	Flask 3.0 + flask-cors	Business Logic & Routing
AI Engine	Google Gemini (google-genai)	Summarization & Embedding
RAG Store	ChromaDB (in-memory)	Vector Similarity Search
Transcript Source	youtube_transcript_api	Caption Retrieval
HTML Parsing	BeautifulSoup4	Video Title Extraction

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Mind Map Render	Mermaid.js	Visual Diagram Generation

Table 2: System Components

4.2 GetVideo Module

The GetVideo class (src/video_info.py) provides four static methods for all YouTube interaction: **GetVideo.Id(url)** extracts the 11-character video ID via regex; **GetVideo.title(url)** fetches the page HTML and parses the title tag using BeautifulSoup; **GetVideo.transcript(url)** retrieves plain text captions; and **GetVideo.transcript_time(url)** retrieves timestamped segments (start time + duration) used for chapter generation.

4.3 Prompt Engineering

The Prompt class (src/prompt.py) manages four prompt templates: *Detailed Summary* (structured headings + bullet points), *Short Summary* (3–5 sentence overview), *Full Explanation* (academic in-depth treatment), and *Timestamp* (chapter link generation in youtube.com/watch?v=ID&t=SECONDS format). Prompt selection is driven by a dictionary mapping the user's UI choice to a prompt ID at runtime.

4.4 Six Output Modes

The system provides six distinct output modes accessible from a single interface: (1) **AI Summary** — Short, Detailed, or Full Explanation; (2) **AI Timestamps** — clickable chapter links; (3) **Full Transcript** — downloadable as .txt; (4) **RAG Chatbot** — interactive Q&A grounded in the transcript; (5) **Mind Map** — Mermaid.js hierarchical diagram; (6) **Copy-to-Clipboard** — available for all text outputs. Mode selection is exposed via a radio group in Streamlit and tab navigation in the React frontend.

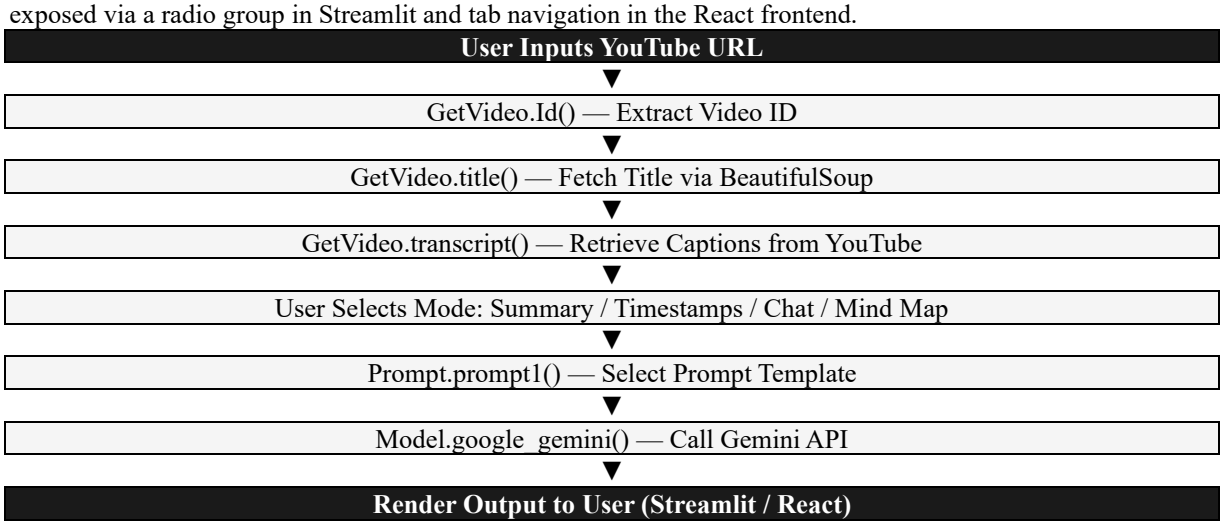

Fig. 1: AI YouTube Summarizer — Core Data Flow

V. AI SUMMARIZATION ENGINE

5.1 Google Gemini Integration

The Model class (src/model.py) interfaces with Google Gemini via the google-genai SDK. On each call, *Model.google_gemini(transcript, prompt, extra, model_type)* initializes a genai.Client with the GOOGLE_GEMINI_API_KEY environment variable, validates the model name against a supported-model list, and invokes *client.models.generate_content()*. The default model is **gemini-flash-latest** — a stable alias ensuring the system automatically benefits from Google's model updates. Fallback models are attempted sequentially on API error, providing transparent resilience.

Model Alias	Context Window	Role in System
gemini-flash-latest	1M tokens	Default — fast inference
gemini-pro-latest	2M tokens	High-quality summarization
gemini-2.5-pro	2M tokens	Most capable analysis

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

gemini-2.5-flash	1M tokens	Fallback 1
gemini-2.0-flash	1M tokens	Fallback 2

Table 3: Supported Google Gemini Models

VI. RAG CHATBOT PIPELINE

6.1 Ingestion Phase

The RAGChat class (src/rag_chat.py) implements a two-phase pipeline. During ingestion, the full video transcript is split into overlapping 500-token chunks with a 50-token overlap to preserve context at boundaries. Each chunk is embedded using the Gemini embedding model and stored in a ChromaDB in-memory collection keyed by an MD5 hash of the video URL. Metadata (chunk_index, video_title, video_url) is stored alongside each vector for deterministic retrieval ordering.

6.2 Query Phase

At query time, the user's question is embedded and compared against all stored chunk vectors using cosine similarity. The top-5 most relevant chunks are retrieved and injected into a Gemini prompt as context, grounding the model's answer in the actual video transcript. This architecture eliminates hallucination on factual questions about video content and produces answers traceable to specific transcript segments.

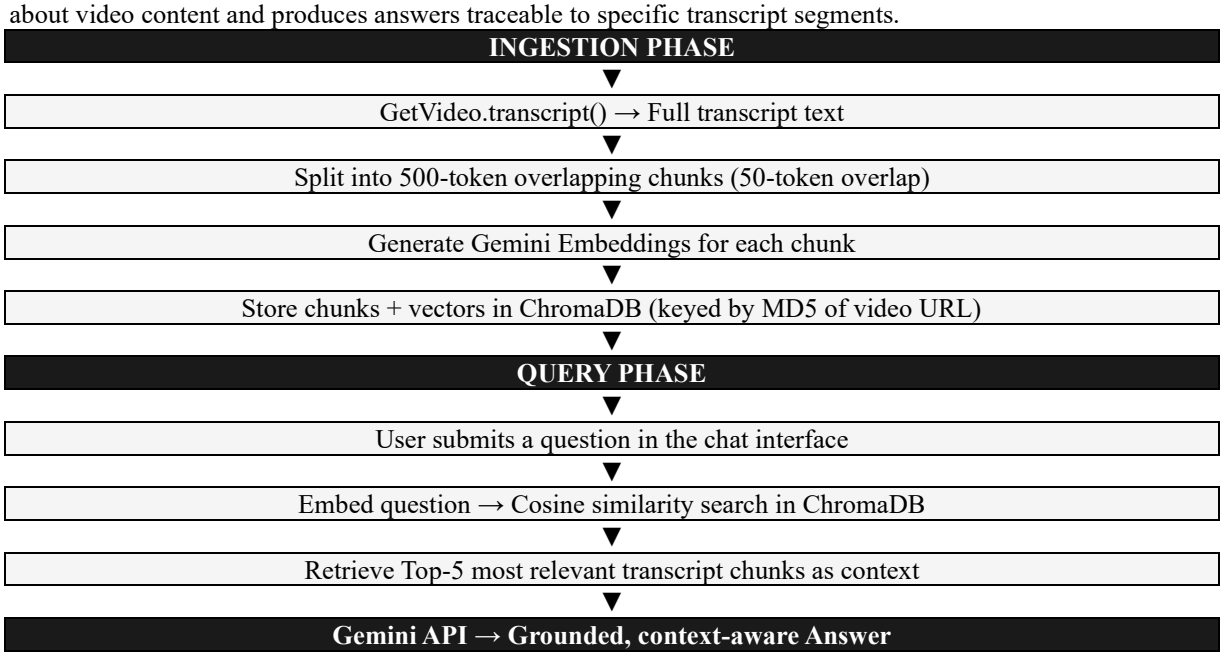

Fig. 2: RAG Chatbot — Ingestion and Query Pipeline

VII. IMPLEMENTATION

7.1 Streamlit Frontend (app.py)

The AIVideoSummarizer class (app.py) orchestrates the Streamlit application. A three-column layout separates URL input and thumbnail (col1), an animated loader (col2), and mode selection with output display (col3). Session state variables manage chat history, loaded transcript chunks, and the active video URL across Streamlit reruns. The application requires only: *pip install -r requirements.txt* followed by *streamlit run app.py*.

7.2 Flask REST API Backend (backend/api.py)

The Flask backend exposes eight RESTful endpoints: **/api/health**, **/api/video-info**, **/api/summary**, **/api/transcript**, **/api/highlights**, **/api/chat/ingest**, **/api/chat/query**, and **/api/mindmap**. CORS is enabled via flask-cors for cross-origin requests from the React development server. Each route follows a uniform pattern: validate input → call src module → handle exception → return JSON with appropriate HTTP status code.

7.3 React Frontend (frontend/src/)

The React frontend uses React 18 functional components with useState/useEffect hooks. LandingPage.jsx handles URL input and video preview; ResultsPage.jsx renders tabbed output panels for each mode. The mind map tab

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

renders Gemini-generated Mermaid.js syntax client-side. All API calls use the native Fetch API with JSON request/response bodies. Tailwind CSS provides responsive utility-first styling.

VIII. RESULTS AND DISCUSSION

The system was evaluated through functional testing of all six output modes and end-to-end user acceptance testing (UAT) with 10 student participants. All functional test cases passed (100% pass rate). UAT results demonstrated an average task satisfaction of 4.6/5. Summarization latency averaged 2–4 seconds for Short Summary and 4–8 seconds for Detailed Summary using gemini-flash-latest. RAG ingestion completed within 2–5 seconds per video, with per-question response times of 2–4 seconds.

Feature	Avg. Processing Time	UAT Satisfaction (1–5)
Short Summary	2 – 4 seconds	4.7
Detailed Summary	4 – 8 seconds	4.6
Full Explanation	8 – 15 seconds	4.5
AI Timestamps	3 – 6 seconds	4.5
RAG Chat (query)	2 – 4 seconds	4.6
Transcript DL	< 2 seconds	4.8
Mind Map	5 – 10 seconds	4.4

Table 4: Feature Performance and UAT Results

A key limitation is ChromaDB's in-memory mode, which does not persist chat state across server restarts. Additionally, the system requires YouTube videos to have captions enabled — videos without transcripts cannot be processed. Summarization quality is also bounded by the accuracy of YouTube's auto-generated captions for technical vocabulary.

IX. FUTURE ENHANCEMENTS

Planned enhancements include: (1) persistent ChromaDB or PostgreSQL storage for cross-session RAG memory; (2) Whisper AI integration for audio transcription of videos without captions; (3) multi-language support with automatic transcript translation; (4) batch processing of YouTube playlists and channels; (5) PDF and Word document export of generated summaries; (6) a React Native mobile application for iOS and Android; and (7) a browser extension for direct summarization from the YouTube watch page without leaving the browser.

X. CONCLUSION

The AI YouTube Video Summarizer demonstrates that powerful, production-ready AI tools can be built and deployed with minimal infrastructure using modern LLM APIs. By combining Google Gemini's 1-million-token context window with a RAG pipeline, Mermaid.js mind maps, and a dual Streamlit/React architecture, the system delivers a feature set that surpasses existing commercial tools — at zero subscription cost. The multi-mode output design, open-source deployment model, and interactive RAG chatbot collectively provide a comprehensive video knowledge extraction solution immediately deployable with a single Python command. This work contributes a replicable open-source pattern for LLM-powered video intelligence systems and opens avenues for further research in AI-assisted knowledge extraction from video media.

XI. REFERENCES

1) Vaswani, A. et al. (2017). Attention is All You Need. *NeurIPS 30.*
2) Lewis, P. et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *NeurIPS 33.*
3) Kasneci, E. et al. (2023). ChatGPT for Good? On Opportunities and Challenges of LLMs for Education. *Learning and Individual Differences.*

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

4) Nallapati, R. et al. (2016). Abstractive Text Summarization Using Sequence-to-Sequence RNNs. *CoNLL 2016.*
5) Google DeepMind. (2024). Gemini: A Family of Highly Capable Multimodal Models. Technical Report. Google.
6) ChromaDB. (2024). ChromaDB Documentation — The AI-Native Open-Source Embedding Database. https://docs.trychroma.com
7) jdepoix. (2024). youtube-transcript-api Python Library. https://github.com/jdepoix/youtube-transcript-api
8) Mermaid.js. (2024). Mermaid Diagram Documentation. https://mermaid.js.org
9) Grinberg, M. (2018). *Flask Web Development* (2nd ed.). O'Reilly Media.
10) Richardson, L. (2024). BeautifulSoup4 Documentation. https://www.crummy.com/software/BeautifulSoup/bs4/doc