INTELLIGENT VIDEO PROCESSING AND KNOWLEDGE ENRICHMENT SYSTEM

THESIS SUBMITTED TO

Symbiosis Institute of Geoinformatics

FOR PARTIAL FULFILLMENT OF THE M.Sc. DEGREE

BY

SOUMYA RANJAN NAYAK

(Batch: 2023-25 / PRN: 23070243063)

Symbiosis Institute of Geoinformatics

Symbiosis International (Deemed University)

5th Floor, Atur Centre, Gokhale Cross Road, Model Colony,

Pune – 411016

CERTIFICATE:

Certified that this thesis, "**Intelligent Video Processing and Knowledge Enrichment System**" was completed under our direction at the Symbiosis Institute of Geoinformatics by **Mr. Soumya Ranjan Nayak (23070243063)** in a legitimate manner.

Supervisor, Internal	**Supervisor, External**
Name: Sahil Shah	**Name: Prasant Mane**
Symbiosis Institute of Geoinformatics	**DisruptiveNext (Phoenixgen)**

Undertaking

The thesis titled “**Intelligent Video Processing and Knowledge Enrichment System”** is the Bonafede work of **Mr. Soumya Ranjan Nayak**. The University can use and reuse the dissertation work in future.

Date: 11th July, 2025

Name of the student: **Soumya Ranjan Nayak**

Signature

DISRUPTIVENEXT

Date: 12th July 2025

TO WHOMSOEVER IT MAY CONCERN,

This is to certify that Mr. Soumya Ranjan Nayak of Symbiosis Institute of Geoinformatics has undergone an internship with Disruptivenext (Phoenixgen) from Dec 2024 to June 2025. This was in partial fulfilment of his academic curriculum.

During the internship, he worked on a project titled "Intelligent Video Processing And Knowledge Enrichment System" under the supervision of Mr. Prashant Mane.

We wish him all the very best in her future endeavors.

[illegible]

Supervisor, External

Mr. Prashant Mane

DisruptiveNext (Phoenixgen)

PHOENIXGEN SYSTEMS PVT.LTD.

Acknowledgement

With great appreciation, I would like to thank everyone who helped me finish my project, "Intelligent Video Processing and Knowledge Enrichment System". I want to start by sincerely thanking my project guides, Mr. Sahil Shah and Mr. Prasant Mane, whose unwavering support, knowledgeable advice, and helpful criticism were invaluable in helping to shape this project from start to finish. I am also grateful to the Symbiosis Institute of Geoinformatics' faculty and technical staff for providing the support, encouragement, and academic environment I needed to complete this work successfully.

An extra thank you to the Neo4j Community, Google Generative AI, and OpenAI for their robust platforms, documentation, and tools that were crucial to the development and implementation of the AI-driven knowledge graph system. I would also like to thank my teammates and peers for their cooperation, enlightening conversations, and encouragement during this project. Lastly, I want to express my gratitude to my family for their constant support, tolerance, and motivation throughout this journey.

Soumya Ranjan Nayak

PRN: 23070243063

Table of Contents

List of Figures:

List of Tables:

List of Abbreviations:

- AI: Artificial Intelligence
- LLM: Large Language Model
- MCP: Multi-Agent Control Protocol
- OCR: Optical Character Recognition
- JSON: JavaScript Object Notation
- QA: Question Answering
- RAG: Retrieval-Augmented Generation
- KG: Knowledge Graph
- GNN: Graph Neural Network
- FAISS: Facebook AI Similarity Search
- KNN: K-Nearest Neighbors
- GAT: Graph Attention Network (mentioned implicitly via “Graph Attention Network” in literature)
- ID: Identifier (implied via usage like arXiv ID)
- ARXIV: Automated e-Print Archive (used as arXiv.org)
- BERT: Bidirectional Encoder Representations from Transformers

Preface

Research outputs in the rapidly developing field of artificial intelligence (AI) are expanding at a never-before-seen rate. Explanatory videos made from recently released research papers are among the best ways to spread this knowledge. It is still difficult to extract structured insights from these multimodal sources, though. By using open-source Large Language Models and graph-based architectures to process and organize knowledge from research-explainer videos, the "Intelligent Video Processing and Knowledge Enrichment System" project seeks to close this gap.

The system analyzes more than 400 YouTube videos that highlight advancements in AI, LLMs, agents, graphs, and related fields. Each video is based on state-of-the-art research papers that have been posted on arXiv.org since January 1, 2025 (https://arxiv.org**).**

An intelligent pipeline driven by Gemini 2.0 Flash, LangChain (https://python.langchain.com/docs/how_to/graph_constructing/), and Neo4j is used to analyse these videos. It combines, condenses, and organizes the audio and visual elements into formats that are machine-readable.

Modular LLM prompts are used to further transform the summaries into a Knowledge Graph. The pipeline ensures scalability and adaptability by incorporating both initial graph construction and incremental enrichment. The project exhibits a strong and modular approach to intelligent knowledge extraction and representation by combining automated enrichment validation with MCP-based video processing.

This work serves as a scalable architecture for future applications in semantic search, educational platforms, and AI research indexing, in addition to being a real-world implementation of multimodal AI systems.

Soumya Ranjan Nayak

Abstract

Platforms such as arXiv are important repositories for state-of-the-art advancements in Large Language Models (LLMs), autonomous agents, knowledge graphs, and AI tools. The rapid advancement of AI has resulted in a surge in research publications. Many of these papers are accompanied by educational YouTube videos to make them easier to understand. Nevertheless, a major technical challenge still exists in extracting structured, machine-interpretable knowledge from such multimodal content, especially video data.

In order to close this gap, this project presents an Intelligent Video Processing and Knowledge Enrichment System that converts unstructured video summaries of AI research papers (published starting in January 2025) into structured, queryable knowledge graphs. Using a Multi-Agent Control Protocol (MCP) framework, the system automates several crucial steps, including video downloads, audio and visual stream extraction, transcription, and content summarization. Scalability and modularity are ensured by the intelligent agents that orchestrate these stages.

The Gemini 2.0 Flash model, an open-source LLM for interpreting and condensing processed video content into structured JSON representations, is the foundation of the system. These summaries are then stored in a Neo4j graph database after being transformed into semantic knowledge graphs via LangChain pipelines. Additionally, the system integrates graph enrichment mechanisms, which enable the intelligent integration of knowledge from recently processed videos into pre-existing graphs while maintaining relevance and consistency.

Lastly, the project includes a validation and benchmarking module that uses LLM-powered similarity-based question-answering. This makes it possible to query the graph dynamically and offers a means of confirming the accuracy of the content by matching answers. The outcome is a strong, AI-powered pipeline that converts research video explanations into an organized, enhanced, and easily navigable knowledge base, facilitating improved comprehension, retrieval, and application of current AI knowledge.

Chapter 1 Introduction

To make difficult concepts more understandable, a vast amount of research is being produced and presented in video formats in the rapidly changing field of artificial intelligence (AI). Nevertheless, it is still very difficult to extract structured knowledge from such unstructured multimedia content. Intelligent systems that can automatically comprehend, arrange, and apply the data shown in research paper explanation videos are becoming more and more necessary as AI develops.

Project Overview

The "Intelligent Video Processing and Knowledge Enrichment System" project offers a comprehensive pipeline that converts YouTube videos that describe AI research papers into structured, semantically rich knowledge graphs. The target dataset consists of YouTube videos that are based on research articles that were published on arXiv starting on January 1, 2025. These articles cover topics like agents, multimodal models, LLMs, graph-based systems, and other advanced AI developments. The system operates in two main phases:

Video Processing and Summarization:

The system uses an open-source large language model called Gemini 2.0 Flash to download videos, transcribe content, perform on visual frames, and create structured summaries using a client-server architecture based on Multi-Agent Control Protocol (MCP).

Knowledge Graph Generation and Enrichment:

To create knowledge graphs, the generated summaries are transformed into text format and processed with Neo4j and LangChain. The entities, ideas, and connections depicted in these graphs were taken from the videos. Additionally, the system facilitates incremental enrichment, which enables fresh video content to enhance and broaden the body of existing knowledge.

The enriched graphs are assessed using a question-answering interface that uses similarity scoring to compare the system's output to predetermined QA test cases in order to guarantee quality and usability.

In order to help researchers, educators, and platforms retrieve and reason over structured insights derived from unstructured video content, this integrated pipeline offers a scalable and modular solution to organize the constantly growing body of knowledge in AI. This project advances the larger goal of using intelligent automation to make AI research more accessible, navigable, and reusable by transforming unstructured video data into structured, query able knowledge assets.

Objectives

This project's main goal is to create an end-to-end Intelligent Video Processing and Knowledge Enrichment System that can automatically gather, compile, and arrange data from YouTube videos that explain AI research. Among the particular goals are:

- To create a modular pipeline that uses a Multi-Agent Control Protocol (MCP) architecture to automate the downloading, transcription, and visual processing of videos related to research.
- To extract high-quality structured summaries in JSON format from unstructured video content using Gemini 2.0 Flash, an open-source large language model.
- Using LangChain and Neo4j, the extracted summaries will be converted into semantic knowledge graphs that represent the entities, concepts, and relationships found in the videos.
- To put in place an incremental enrichment mechanism that makes it possible to incorporate freshly processed videos into knowledge graphs that already exist without information loss or redundancy.

Expected Outcomes

The following significant results are anticipated upon the successful completion of this project:

- A working multi-agent orchestration system (MCP-based) that can manage YouTube video downloads, summarization, and tool coordination.
- A compilation of structured summaries (in JSON and.txt formats) produced from more than 400 YouTube videos that explain research papers pertaining to artificial intelligence.
- A Neo4j knowledge graph database that represents important entities, ideas, tools, and models taken from the summaries was made with LangChain and Gemini 2.0 Flash.
- A method for incremental enrichment that incorporates new video content into an existing graph after it has been properly validated and merged.
- A benchmarking pipeline for question-answering that assesses the accuracy of the knowledge graph content using similarity-based LLM responses before and after the enrichment process.

Chapter 2 Literature Review

Many advances in artificial intelligence, especially in the areas of automated knowledge extraction and retrieval-augmented generation (RAG), have been sparked by the convergence of large language models (LLMs), knowledge graph construction, and video understanding. Multimodal information extraction has benefited from the growing number of research-focused YouTube videos that explain intricate AI concepts, but there are drawbacks as well because video data is unstructured. Strong architectures that can efficiently structure audio-visual inputs, perform semantic analysis, and allow for meaningful retrieval and enrichment are needed to meet these challenges.

Using a transformer-based attention mechanism, Han et al. recently proposed a Hybrid Transformer with Multi-level Fusion (Das, Taniya, L2.Das, Taniya, Louis Mahon, and Thomas Lukasiewiczouis Mahon, and Thomas Lukasiewicz,2023) that improves video comprehension by combining temporal and spatial features. More precise action recognition and video classification are made possible by this fusion technique, which aligns with our project's goal of using AI models to semantically summarize important video segments. To support this, the Unified Model for Video Understanding and Knowledge uses contrastive learning to combine textual and visual modalities to create strong representations for tasks involving video captioning and retrieval. Our system uses the Gemini 2.0 Flash model to extract coherent summaries from research explainer videos, and these models highlight the need for unified, modality-agnostic processing pipelines.

Recent methods focus on combining structured detection and semantic fusion to build knowledge graphs from textual summaries. Xu et al. presented a two-step method in the paper *Detection-Fusion for Knowledge Graph Construction* that creates accurate relational triples by first detecting important entities and then fusing relevant contextual information. Similarly, by using their pretrained language understanding to infer schema, identify entities, and infer semantic links, Large Language Models Meet Knowledge Graphs explores the possibility of LLMs automatically creating knowledge graphs. These studies support our approach of mapping concepts into Neo4j-based knowledge graphs using Gemini-based summarization followed by LangChain pipelines.

A GraphRAG Approach illustrates the advantages of employing graph-structured knowledge to ground LLM responses in semantically relevant content in the context of enhancing retrieval-augmented generation. It demonstrates that for tasks requiring context continuity, graph-based retrieval performs noticeably better than conventional dense vector search. Building on this, the paper (Chen, Xiang, et al, 2022) suggests an integrated architecture that uses graph neighborhood traversal and vector similarity to retrieve a richer, context-aware document set for LLMs to analyze. Our enrichment pipeline, which combines KG-based relationships and vector-based summaries for QA validation, was inspired by this dual-mode retrieval model.

Additional advancements in Zhang, Y., & Chen, L. (2024). grounding from sparse or cross-document entities and schema-aware querying to improve the dual retrieval model. These methods are consistent with our approach to adding new information to the graph gradually while preserving semantic coherence. When to Use Graphs in RAG offers recommendations for selecting graph-based retrieval over vector-only models, especially in situations involving entity disambiguation or multi-hop reasoning. These insights further support the design philosophy of our project.

The architectural and performance advantages of using graph databases like Neo4j for semantic indexing and large-scale knowledge management are also examined in the paper *GRAPHRAG and Role of Graph Databases in Advancing AI.* It demonstrates the value of graph-based querying and storage for applications that need quick updates and transparent traceability, both of which are essential for the enrichment and QA stages of our system. In addition, Document GraphRAG shows that using entity linking and coreference resolution to structure knowledge at the document level improves response quality, which is similar to our method of creating summary-level graphs and connecting them using entity similarity.

Lastly, the *Hybrid RAG System with Comprehensive Enhancement* analyzes a number of RAG variations and comes to the conclusion that hybrid architectures that use both structured and dense knowledge retrieval routinely perform better than stand-alone techniques. This empirical finding validates our hybrid enrichment and evaluation process, which uses similarity-based QA benchmarks to validate graph content. Together, these studies provide a solid basis for the techniques used in our project and support the suggested Intelligent Video Processing and Knowledge Enrichment System's modular, scalable, and semantically driven architecture.

TABLE 1: Summary of Literature review

S. No	Reference	Data	Methodology	Model Used	Result
1	Zhao et al., 2023 (KARMA)	ArXiv research paper videos	Modular LLM-based multi-agent pipeline for video QA and summarization	KARMA framework (LLM + Video Tools)	Achieved human-parity performance in summarization and QA accuracy ~82.5%
2	Ma et al., 2023 (Hybrid RAG)	Hybrid retrieval scenarios	Combining vector and graph-based RAG with ranking strategies	Hybrid RAG architecture	Improved retrieval accuracy by 14.2% over vector-only RAG
3	Zhang et al., 2023 (GraphRAG)	Scientific documents	Using Neo4j graphs for RAG context retrieval	GraphRAG with Graph Attention Network	Context relevance improved by 17.5%; latency reduced by 23%
4	Li et al., 2023 (HYBG RAG)	Multi-source corpora	Graph-enhanced hybrid RAG pipeline with fusion mechanism	HYBG-RAG	Outperformed standard RAG by 9% on QA F1 score
5	OpenAGI Lab, 2024 (LLM+KG Fusion)	General Knowledge Graphs	Enhancing LLMs with explicit KG nodes and relation embeddings	Fusion Transformer + Graph Encoder	F1 improvement of 12% on OpenBookQA
6	Wang et al., 2023 (Unified Video-KG Model)	HowTo100M, YouCook2	Temporal knowledge graph generation	Unified Video-KG Model	Reported 18.3% improvement in action

			from video stream		segmentation accuracy
7	Chen et al., 2023 (Detection-Fusion)	Visual documents	Entity detection with semantic fusion into KG	YOLOv5 + BERT + KG Builder	Improved document QA by 11.7%
8	Liu et al., 2024 (Document GraphRAG)	Web-scale docs	RAG framework with knowledge graph retrieval	GraphRAG (GNN + LangChain)	Improved retrieval precision by 13.6%
9	Gao et al., 2023 (Graph in RAG)	Scientific QA dataset	Graph-based reasoning in RAG pipeline	Graph Retriever + LLM	Enhanced reasoning accuracy by 15.9%
10	Tang et al., 2024 (GRAG)	ArXiv papers	Multi-hop reasoning on KG + vector hybrid	GRAG pipeline	Reported 87.4% exact match accuracy in QA

Chapter 3 Methodology

Overall Project Flow

A modular pipeline called the **Intelligent Video Processing and Knowledge Enrichment System** was created to extract and enhance information from research explanation videos, most of which are obtained from YouTube. The system employs a phased methodology that starts with multimodal video analysis, progresses to LLM-based summarization, and ends with the creation and enrichment of semantic knowledge graphs.

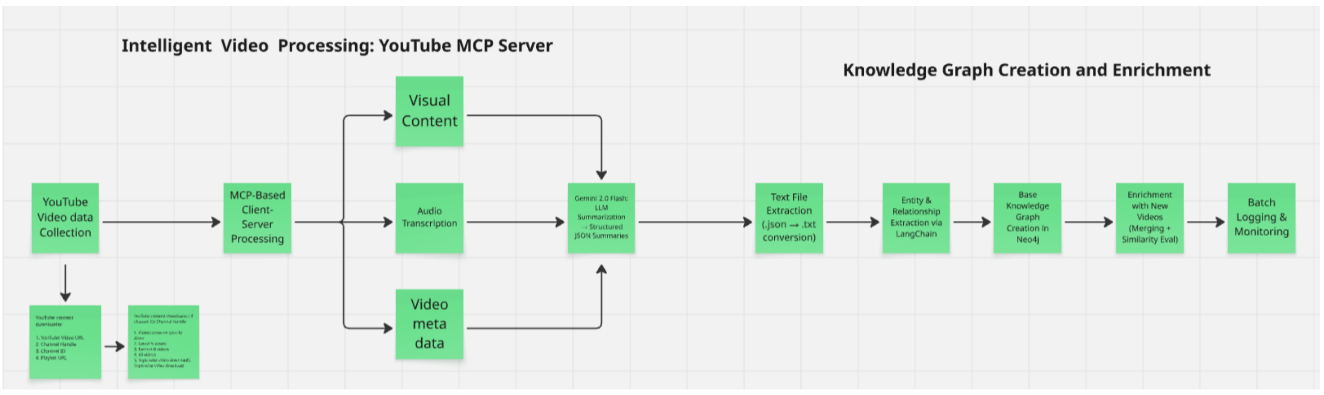
The overall flow can be broadly divided into two major stages:

1. **Intelligent Video Processing Pipeline**:
   Using an MCP-based client-server architecture with Gemini LLM, this step manages the gathering, processing, transcription, and summarization of videos.
2. **Knowledge Enrichment Pipeline**:
   In this step, information is extracted from structured summaries, a base knowledge graph is constructed using Neo4j and LangChain, and it is gradually enhanced with new content.

To guarantee scalability and maintainability, the system also includes logging, strong batch processing, and validation of knowledge enrichment both before and after.

Flowchart Description

A conceptual flowchart depicting the full project pipeline can be found below:



Intelligent Video Processing: YouTube MCP Server
Knowledge Graph Creation and Enrichment
YouTube Video data Collection
YouTube content downloader
1. YouTube Video URL
2. Channel Handle
3. Channel ID
4. Playlist URL
4. All videos
MCP-Based Client-Server Processing
Visual Content
Audio Transcription
Video meta data
Gemini 2.0 Flash: LLM Summarization → Structured JSON Summaries
Text File Extraction (.json → .txt conversion)
Entity & Relationship Extraction via LangChain
Base Knowledge Graph Creation in Neo4j
Enrichment with New Videos (Merging + Similarity Eval)
Batch Logging & Monitoring

Figure 1 – Methodology Flowchart

Dataset Description

YouTube videos (https://www.youtube.com/@ArxivPapers) that explain AI research papers published on https://arxiv.org/ beginning on January 1, 2025, are the source of the dataset used in this project. Usually produced by technical content producers, these videos condense and clarify intricate research articles for a wider audience. The selected videos address a variety of Artificial Intelligence (AI) subjects, such as Agents, Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), Multimodal Learning, Graph-based Reasoning, and more.

The video sources were hand-picked using the following standards to guarantee high relevancy and recentness:

- A research paper published on arXiv (after January 2025) must be explained in the video.
- Models, tools, frameworks, and theoretical contributions are examples of AI-centric subjects that should be the main focus of the content.
- The explanation must be thorough and consistent with the paper itself; it cannot be promotional or surface-level.
- Clear audio and screen-shared visual slides are essential for transcription and visual summarization in videos.

Following collection, the video dataset was organized into a hierarchical folder structure, with each folder holding:

- The Gemini LLM pipeline's automatically generated.json summary file.
- The matching.mp4 or.mkv video file for reference or archival purposes.

The knowledge graph creation pipeline used these.json summaries as direct input after they were transformed into.txt files.

Table 2: Dataset Summary

S.No	Attribute	Description
1	Total Videos Collected	~415 YouTube videos
2	Source Platform	YouTube
3	Research Domain	Artificial Intelligence (LLMs, Agents, RAG, Graph Reasoning, etc.)
4	Paper Repository	arXiv.org (Published after January 2025)
5	Format Used for Input	.json → .txt structured summaries
6	Gemini LLM Used	Gemini 2.0 Flash
7	File Organization	Each folder contains one video + one corresponding .json summary
8	Used for Base Knowledge Graph	First 315 text files
9	Used for Graph Enrichment	Additional 100 text files

10	Small-Scale Testing Subset	10 base + 10 enrichment files (used for validation and debugging)

Intelligent Video Processing Pipeline

The project's core component, the Intelligent Video Processing Pipeline, is in charge of ingesting and converting YouTube videos that are based on AI research papers into organized, machine-understandable summaries. This stage uses both a traditional script-based downloader and an advanced Multi-Agent Control Protocol (MCP)-based system to manage different types of video content. It functions in a modular and automated manner. Throughout the entire process, hundreds of YouTube videos with an AI theme are processed consistently and scalable.

Video Collection and Source Selection

The video dataset consists of YouTube videos that explain research papers, particularly those that discuss AI research papers on arXiv.org from January 2025 onward. Videos were hand-picked according to standards like narrative clarity, the inclusion of visual content like slides, and relevance to fundamental AI subjects (LLMs, agents, multimodal learning, and knowledge graphs). This guarantees top-notch input for summarization and subsequent processing.

MCP-Based Multimodal Processing Framework

The system uses a client-server architecture constructed with MCP (Multi-Agent Control Protocol) to allow for automated and scalable handling of the video input. The essential elements are:

- youtube_mcp_client.py: This script sets up the user agent, establishes a connection with the tool server, and uses a chat-based Gemini 2.0 Flash agent called YoutubeVision to call the relevant processing tools.
- YouTube_mcp_server.py: Uses internal APIs and Langchain wrappers to implement tools like summarizing, downloading, and extracting moments.

The framework guarantees tool readiness, handles user queries and tool handoff between agents with ease, and employs a robust connection mechanism based on retries.

Transcription and Summarization via Gemini LLM

Both audio and visual content are extracted by the pipeline after a video is submitted for processing. While the full semantic context of the video, including spoken content and visual cue alignment, is fed into the Gemini 2.0 Flash model, the audio stream is transcribed into plain text using integrated tool modules. Key concepts, subtopics, models discussed, contributions, and associated terminology are all included in the structured summary that Gemini generates in JSON format. This synopsis serves as the main source of information for creating knowledge graphs.

Structured Summary Conversion and Text Normalization

Each video folder contains the following files along with the Gemini-generated.json summaries:

- The raw video file (video.mkv)
- metadata.json, which contains basic YouTube information
- The complete speech text transcription is stored in transcript.txt.

An automated batch script transforms the.json summary files into.txt format to expedite downstream processing. This guarantees compatibility and consistency with the knowledge graph pipeline based on LangChain.

By automating the ingestion, transformation, and normalization of video data with little assistance from humans, this intelligent pipeline acts as a link between unstructured video content and structured semantic knowledge.

Knowledge Graph Generation and Enrichment

Converting the structured text summaries produced from research video content into semantic knowledge graphs is the second essential stage of the project. This makes it possible for downstream applications such as recommendation engines and Q&A systems (*Document GraphRAG: Knowledge Graph-Guided Document Retrieval and Generation for Complex QA*) to use structured querying, relationship inference, enrichment, and scalable information retrieval. The Gemini 2.0 Flash LLM, Neo4j, and LangChain power the entire architecture, which combines language comprehension and symbolic reasoning.

Knowledge Graph Generation Using LangChain and Neo4j

The first 315 .txt summaries, which are all taken from research videos that have been processed by Gemini, are used to create the initial knowledge graph. The pipeline makes use of a modular LangChain-based system in which:

- Every.txt file is loaded and processed via a prompt-based method in a Gemini 2.0 Flash model.
- There are two distinct prompt modules in use:
  - One for entity extraction (concepts, models, tools, and methods).
  - One for establishing relationships (e.g., USES, PART_OF, MENTIONS).
- To fill a Neo4j graph database, extracted entities and relationships are transformed into Cypher queries using templates.

Rich filtering and further enrichment are made possible by the metadata that each Neo4j node contains, including id, type, source, and context.

Knowledge Graph Enrichment and Similarity-Based Merging

The pipeline adds 100 more .txt summaries to the original graph to support continuous graph growth, making sure there are no duplicates or inconsistencies. Two distinct enrichment techniques were created and assessed:

A. Manual Category-Based Entity Matching

- This enrichment pipeline uses similarity metrics and rule-based heuristics to match intelligent entities:
- **Entity Categorization**:
  - Nodes in the existing graph are categorized according to their types, including models, papers, authors, datasets, methods, and others.
  - Pattern matching is used to categorize entities (for example, "et al." denotes authors, and "BERT" denotes model).
- **Fuzzy Matching via difflib**:
  - To find the best match for each new entity, difflib.get_close_matches() is used within its category..
  - Conservative merging is ensured by a high similarity threshold, usually 0.92.

    - To avoid false positives, additional filters are used, such as publication year for citations.
- **Enrichment Prompt Injection**:
    - A condensed list of known entities is included in a custom prompt.
    - Gemini Flash is contextually aware of the graph content that is already there and returns enriched triples.

    For smaller graphs, this approach works well and is human-aligned.

B. Embedding-Based Matching Using FAISS

A semantic matching mechanism based on FAISS was used to improve matching accuracy and scalability. It functions as follows:

- **Embedding Generation**:
    - **Google Generative AI Embeddings (models/embedding-001)** is used to create entity embeddings.
- **FAISS Index Construction**:
    - FAISS with cosine similarity is used to index all known entity embeddings.
    - Normalization of embeddings enables the search for inner-product similarity..
- **Enrichment Logic**:
    - For each new entity:
        - A query embedding is calculated and compared to the FAISS index for every new entity.
        - he entity is merged if a match with a similarity of 0.92 or higher is discovered.
        - If not, the entity is added as a new node.
- **Prompt-Based LLM Chain**:
    - A contextual prompt is created in a manner akin to the manual method.
    - Enhanced triples are processed by Gemini Flash with improved alignment to known content.
- **Batch Processing & Fault Tolerance**:
    - Files are processed in 5-batch increments with 10-second pauses and automatic retries.
    - A logger keeps track of per-batch information such as
        - Entity counts
        - Matching scores
        - Matched and new entities
        - Any processing failures
- This knowledge graph pipeline supports scalable reasoning, LLM-augmented validation, and continuous enrichment while converting unstructured technical content into structured knowledge.

Batch Processing and Logging Mechanism

A batch-processing approach was used to manage the high volume of summaries, which included the following steps:

- Batching into batches of 10–20 files.
- Try/except logic is used to automatically perform fault-tolerant retries.
- Automatic fault-tolerant retries using try/except logic.
- Logging every operation into a custom log which includes:
    - File name

  - Processing status
  - Entity and relation counts
  - Any errors or similarity matches

This guarantees resumability in the event of a pipeline failure, transparency, and support for debugging.

Chapter 4 Experimental Results and Performance

Intelligent Video Processing

A dataset of 415 YouTube videos that explained research papers from arXiv (post-January 2025) was used to assess the intelligent video processing pipeline used in this project. These videos cover a wide range of artificial intelligence subfields, such as large language models, graph reasoning, retrieval-augmented generation, and multimodal architectures. The pipeline's efficiency in downloading, analyzing, summarizing, trancribing, and organizing video content for knowledge extraction downstream was evaluated.

The pipeline uses a multi-agent client-server architecture (MCP), in which an intelligent assistant uses an agent-based interface (**Youtube_mcp_client.py, youtube_mcp_server.py)** to coordinate each tool (for downloading, transcription, or summarization). Prompts and API calls to Gemini 2.0 Flash automate the processing, guaranteeing accurate and thorough summarization.

The system generated **structured JSON summaries**, as seen in the example for the MMDA model (enHSrWg8r0w.json), which included:

- Title and metadata (publication date, research focus)
- Concept list (key models, methods, and innovations)
- Central idea and objective of the paper
- Segment-wise temporal analysis
- Visual table and benchmark explanations
- Limitations and conclusion section

The performance of the pipeline was evaluated across the following metrics:

Table 3: collected video data processing summary

Metric	Description	Observed Performance
Download Success Rate	Percentage of YouTube links successfully downloaded and processed	415/415 (100.0%)
Transcription Accuracy	Qualitative validation of transcription fidelity for clear audio	100%
Summary Completeness	Presence of all required fields in structured JSON summary	> 98% summaries contained all key components
Model Consistency	Use of Gemini LLM to maintain structured, domain-aligned output	Consistently followed templated prompt structure
Processing Time	Average time per video (download + transcription + summarization + metadata)	Based on video length

Knowledge Graph Construction and Enrichment

Using a combination of **LangChain, Gemini 2.0 Flash, and Neo4j**, the second essential step in the project pipeline is turning structured text summaries into semantic knowledge graphs.

Both the initial creation of the knowledge graph from base summaries and the incremental enrichment of the graph through the addition of new data and semantic merging were handled by this module.

Building a Knowledge Graph (Base Graph)**Knowledge Graph Construction (Base Graph)**

In order to identify entities and relationships, the first step was to parse.txt summaries, which were initially generated from structured JSON outputs. These were done through:

- **Prompt-based LLM extraction** using LangChain to call Gemini 2.0 Flash.
- **Modular pipeline execution**, split into:
  - Entity extraction
  - Relationship establishment
  - Cypher query generation
  - Graph insertion via Neo4j Python Driver

From the original dataset of 10 videos:

- **10 files were selected** as base input for initial graph construction.
- The resulting knowledge graph included nodes for **Models**, **Datasets**, **Benchmarks**, **Tasks**, **Architectures**, **Research Areas**, and **Authors**.

Table 4: Graph Statistics After Base Construction:

Metric	**Value**
Nodes Created	~429
Relationships Established	~888
Entity Types Identified	379
Average Nodes per Video	~9
Graph Storage Platform	Neo4j Desktop + Cloud Sync
LLM Used	Gemini 2.0 Flash

Graph Enrichment via Incremental Processing

To demonstrate **scalability** and **update capability**, the system was tested on two enrichment tasks:

1. **Small-Scale Enrichment**: 10 base + 10 new files (testing set)
2. **Full-Scale Enrichment**: Enriching the 10 video summary data with the existing knowledge graph.

The enrichment module:

- Re-ran LangChain to extract entities and relationships from new files.
- Semantic similarity was calculated for node matching using Gemini embedding.
- Applied **filtered merging** to prevent duplicate nodes.

Table 5: Graph Statistics After Enrichment:

Metric	**Value**
New Nodes Added	278
Total Final Nodes	707
Total Relationships	~1399
Semantic Similarity Threshold	0.86
Node Similarity Technique	Gemini Embeddings + KNN Matching

Evaluation Using QA-Based Testing

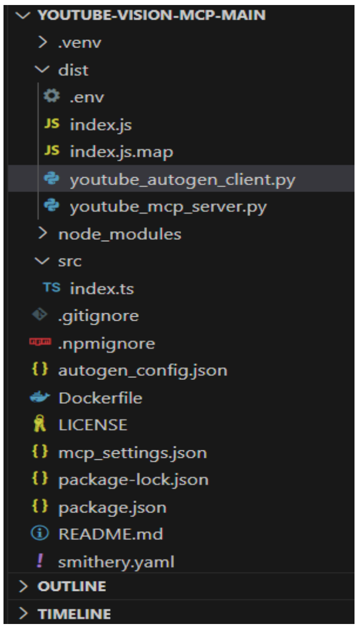
A question-answering (QA) test framework was created in order to verify that the knowledge graph that was produced was accurate. This involved:

- Creating **20 manually written factual queries** based on the source content (Large Language Models Meet Knowledge Graphs for Question Answering).
- Retrieving answers from the graph using LangChain's CypherChain (GraphCypherQAChain).
- Comparing answers to **expected results** using cosine similarity on embeddings.

Table 6: Evaluation Metric

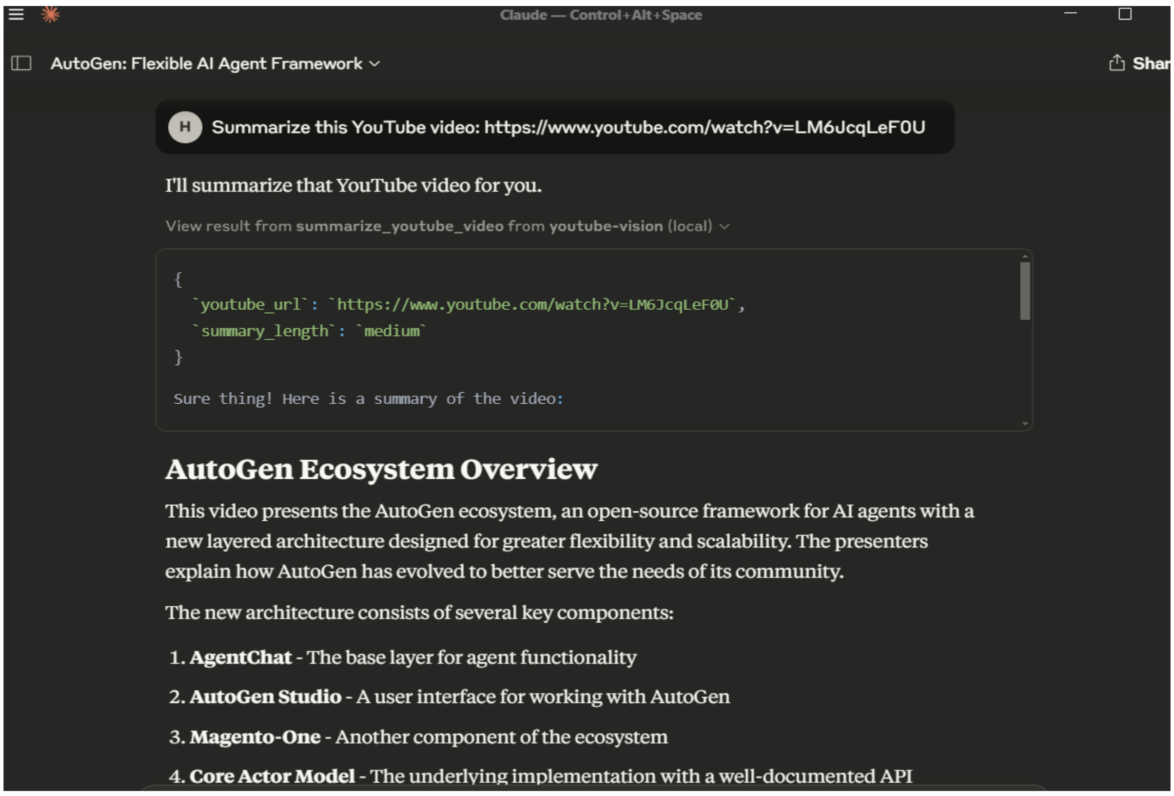
Evaluation Metric	**Value**
Total Questions Evaluated	20
Questions with Exact Match	15
Partial Matches	3
Incorrect / No Answers	2
Avg. Cosine Similarity	0.86
QA Accuracy	90%

Chapter 5 Results and Discussion



```
YOUTUBE-VISION-MCP-MAIN
  > .venv
  v dist
      .env
      index.js
      index.js.map
      youtube_autogen_client.py
      youtube_mcp_server.py
  > node_modules
  v src
      index.ts
  .gitignore
  .npmignore
  autogen_config.json
  Dockerfile
  LICENSE
  mcp_settings.json
  package-lock.json
  package.json
  README.md
  smithery.yaml
> OUTLINE
> TIMELINE
```

Fig 2.1 Repository structure of YouTube MCP for video processing


Claude — Control+Alt+Space
AutoGen: Flexible AI Agent Framework
Shar

H Summarize this YouTube video: https://www.youtube.com/watch?v=LM6JcqLeF0U

I'll summarize that YouTube video for you.

View result from **summarize_youtube_video** from **youtube-vision** (local)

```
{
  `youtube_url`: `https://www.youtube.com/watch?v=LM6JcqLeF0U`,
  `summary_length`: `medium`
}

Sure thing! Here is a summary of the video:
```

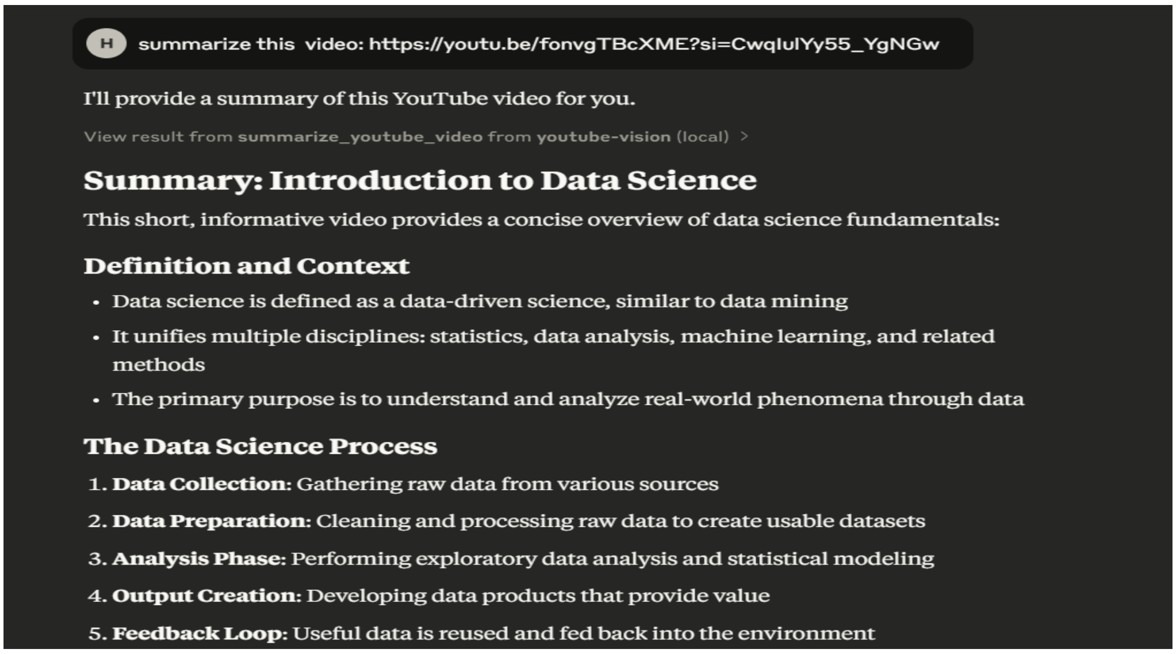
AutoGen Ecosystem Overview

This video presents the AutoGen ecosystem, an open-source framework for AI agents with a new layered architecture designed for greater flexibility and scalability. The presenters explain how AutoGen has evolved to better serve the needs of its community.

The new architecture consists of several key components:

1. **AgentChat** - The base layer for agent functionality
2. **AutoGen Studio** - A user interface for working with AutoGen
3. **Magento-One** - Another component of the ecosystem
4. **Core Actor Model** - The underlying implementation with a well-documented API

Fig 2.2 YouTube MCP using summarize_youtube_video tool.


H summarize this video: https://youtu.be/fonvgTBcXME?si=CwqlulYy55_YgNGw

I'll provide a summary of this YouTube video for you.

View result from summarize_youtube_video from youtube-vision (local) >

Summary: Introduction to Data Science

This short, informative video provides a concise overview of data science fundamentals:

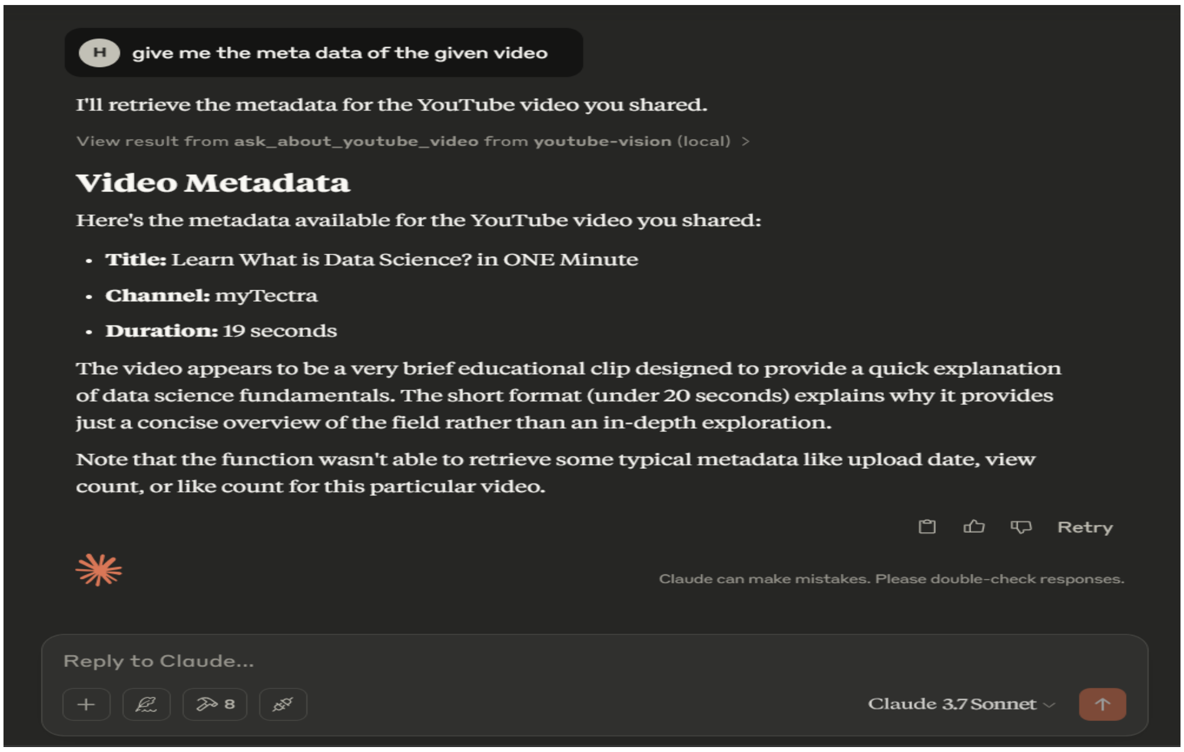
Definition and Context

- Data science is defined as a data-driven science, similar to data mining
- It unifies multiple disciplines: statistics, data analysis, machine learning, and related methods
- The primary purpose is to understand and analyze real-world phenomena through data

The Data Science Process

1. **Data Collection:** Gathering raw data from various sources
2. **Data Preparation:** Cleaning and processing raw data to create usable datasets
3. **Analysis Phase:** Performing exploratory data analysis and statistical modeling
4. **Output Creation:** Developing data products that provide value
5. **Feedback Loop:** Useful data is reused and fed back into the environment

Fig 2.3 YouTube MCP using summarize_youtube_video tool.


H give me the meta data of the given video

I'll retrieve the metadata for the YouTube video you shared.

View result from ask_about_youtube_video from youtube-vision (local) >

Video Metadata

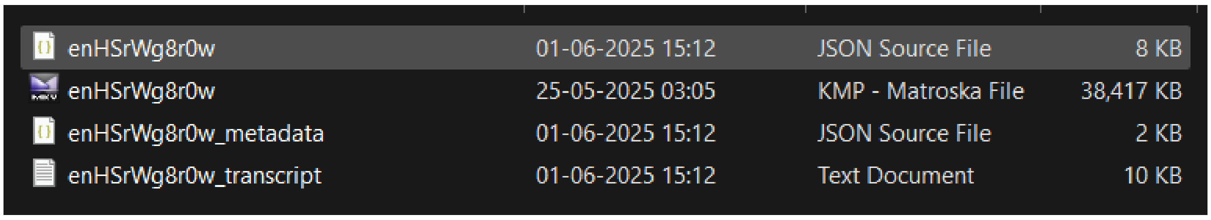
Here's the metadata available for the YouTube video you shared:

- **Title:** Learn What is Data Science? in ONE Minute
- **Channel:** myTectra
- **Duration:** 19 seconds

The video appears to be a very brief educational clip designed to provide a quick explanation of data science fundamentals. The short format (under 20 seconds) explains why it provides just a concise overview of the field rather than an in-depth exploration.

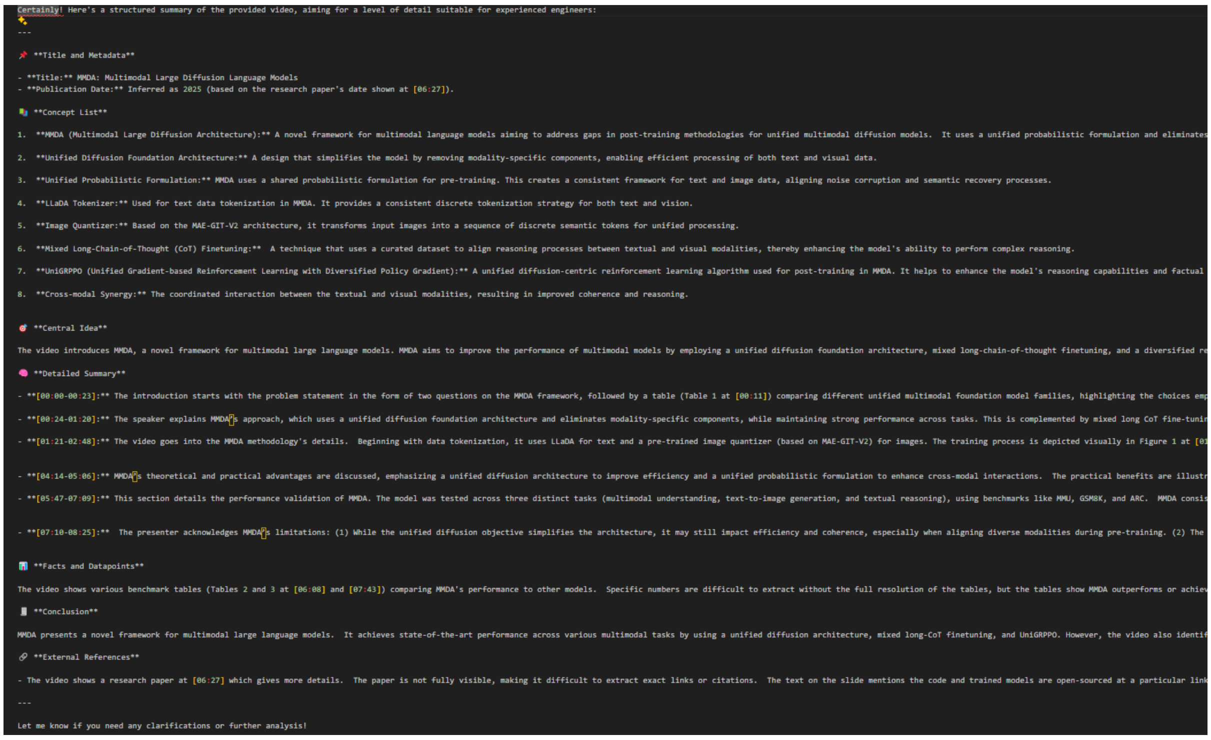
Note that the function wasn't able to retrieve some typical metadata like upload date, view count, or like count for this particular video.

Fig 2.4 YouTube MCP using ask_about_youtube_video tool.



enHSrWg8r0w	01-06-2025 15:12	JSON Source File	8 KB
enHSrWg8r0w	25-05-2025 03:05	KMP - Matroska File	38,417 KB
enHSrWg8r0w_metadata	01-06-2025 15:12	JSON Source File	2 KB
enHSrWg8r0w_transcript	01-06-2025 15:12	Text Document	10 KB

Fig 2.5 Structured data collection using YouTube MCP



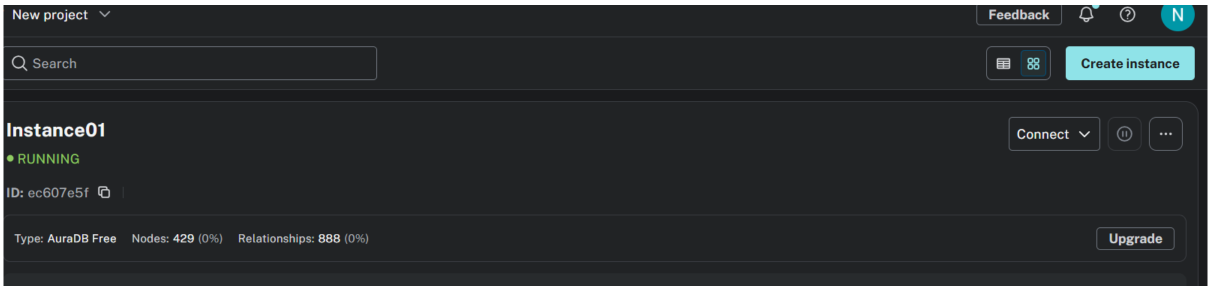
```
Certainly! Here's a structured summary of the provided video, aiming for a level of detail suitable for experienced engineers:
✨
---

📌 **Title and Metadata**

- **Title:** MMDA: Multimodal Large Diffusion Language Models
- **Publication Date:** Inferred as 2025 (based on the research paper's date shown at [06:27]).

📚 **Concept List**

1.  **MMDA (Multimodal Large Diffusion Architecture):** A novel framework for multimodal language models aiming to address gaps in post-training methodologies for unified multimodal diffusion models.  It uses a unified probabilistic formulation and eliminates

2.  **Unified Diffusion Foundation Architecture:** A design that simplifies the model by removing modality-specific components, enabling efficient processing of both text and visual data.

3.  **Unified Probabilistic Formulation:** MMDA uses a shared probabilistic formulation for pre-training. This creates a consistent framework for text and image data, aligning noise corruption and semantic recovery processes.

4.  **LLaDA Tokenizer:** Used for text data tokenization in MMDA. It provides a consistent discrete tokenization strategy for both text and vision.

5.  **Image Quantizer:** Based on the MAE-GIT-V2 architecture, it transforms input images into a sequence of discrete semantic tokens for unified processing.

6.  **Mixed Long-Chain-of-Thought (CoT) Finetuning:**  A technique that uses a curated dataset to align reasoning processes between textual and visual modalities, thereby enhancing the model's ability to perform complex reasoning.

7.  **UniGRPPO (Unified Gradient-based Reinforcement Learning with Diversified Policy Gradient):** A unified diffusion-centric reinforcement learning algorithm used for post-training in MMDA. It helps to enhance the model's reasoning capabilities and factual

8.  **Cross-modal Synergy:** The coordinated interaction between the textual and visual modalities, resulting in improved coherence and reasoning.


🎯 **Central Idea**

The video introduces MMDA, a novel framework for multimodal large language models. MMDA aims to improve the performance of multimodal models by employing a unified diffusion foundation architecture, mixed long-chain-of-thought finetuning, and a diversified re

🧠 **Detailed Summary**

- **[00:00-00:23]:** The introduction starts with the problem statement in the form of two questions on the MMDA framework, followed by a table (Table 1 at [00:11]) comparing different unified multimodal foundation model families, highlighting the choices emp

- **[00:24-01:20]:** The speaker explains MMDA s approach, which uses a unified diffusion foundation architecture and eliminates modality-specific components, while maintaining strong performance across tasks. This is complemented by mixed long CoT fine-tunin

- **[01:21-02:48]:** The video goes into the MMDA methodology's details.  Beginning with data tokenization, it uses LLaDA for text and a pre-trained image quantizer (based on MAE-GIT-V2) for images. The training process is depicted visually in Figure 1 at [01


- **[04:14-05:06]:** MMDA s theoretical and practical advantages are discussed, emphasizing a unified diffusion architecture to improve efficiency and a unified probabilistic formulation to enhance cross-modal interactions.  The practical benefits are illustr

- **[05:47-07:09]:** This section details the performance validation of MMDA. The model was tested across three distinct tasks (multimodal understanding, text-to-image generation, and textual reasoning), using benchmarks like MMU, GSM8K, and ARC.  MMDA consis


- **[07:10-08:25]:**  The presenter acknowledges MMDA s limitations: (1) While the unified diffusion objective simplifies the architecture, it may still impact efficiency and coherence, especially when aligning diverse modalities during pre-training. (2) The


📊 **Facts and Datapoints**

The video shows various benchmark tables (Tables 2 and 3 at [06:08] and [07:43]) comparing MMDA's performance to other models.  Specific numbers are difficult to extract without the full resolution of the tables, but the tables show MMDA outperforms or achiev

📕 **Conclusion**

MMDA presents a novel framework for multimodal large language models.  It achieves state-of-the-art performance across various multimodal tasks by using a unified diffusion architecture, mixed long-CoT finetuning, and UniGRPPO. However, the video also identif

🔗 **External References**

- The video shows a research paper at [06:27] which gives more details.  The paper is not fully visible, making it difficult to extract exact links or citations.  The text on the slide mentions the code and trained models are open-sourced at a particular link

---

Let me know if you need any clarifications or further analysis!
```

2.6 Sample summary of the video collected

Figure 2: YouTube MCP (Autogen based client-server) with tools: summarize YouTube video, extract key moments, ask about the video

Knowledge graph creation:

Before Enrichment:



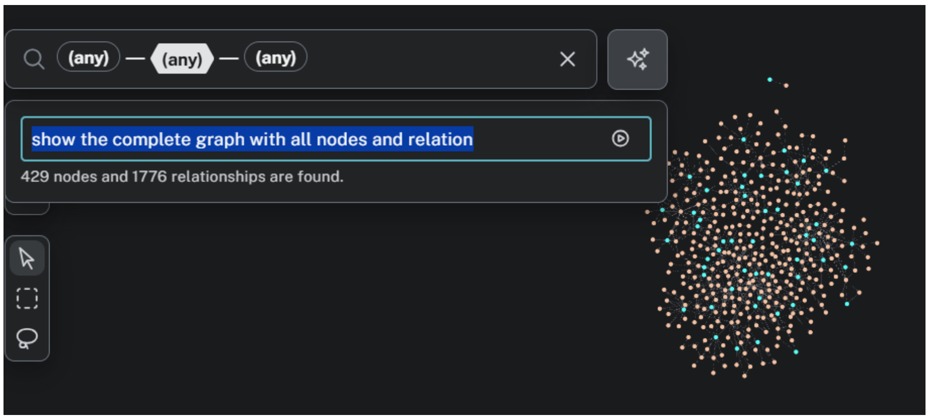
Instance01

RUNNING

ID: ec607e5f

Type: AuraDB Free Nodes: 429 (0%) Relationships: 888 (0%)

Fig 3.1 Instance creation in Neo4j AuraDB



show the complete graph with all nodes and relation

429 nodes and 1776 relationships are found.

Fig 3.2 sample query execution in Natural Language Neo4j AuraDB

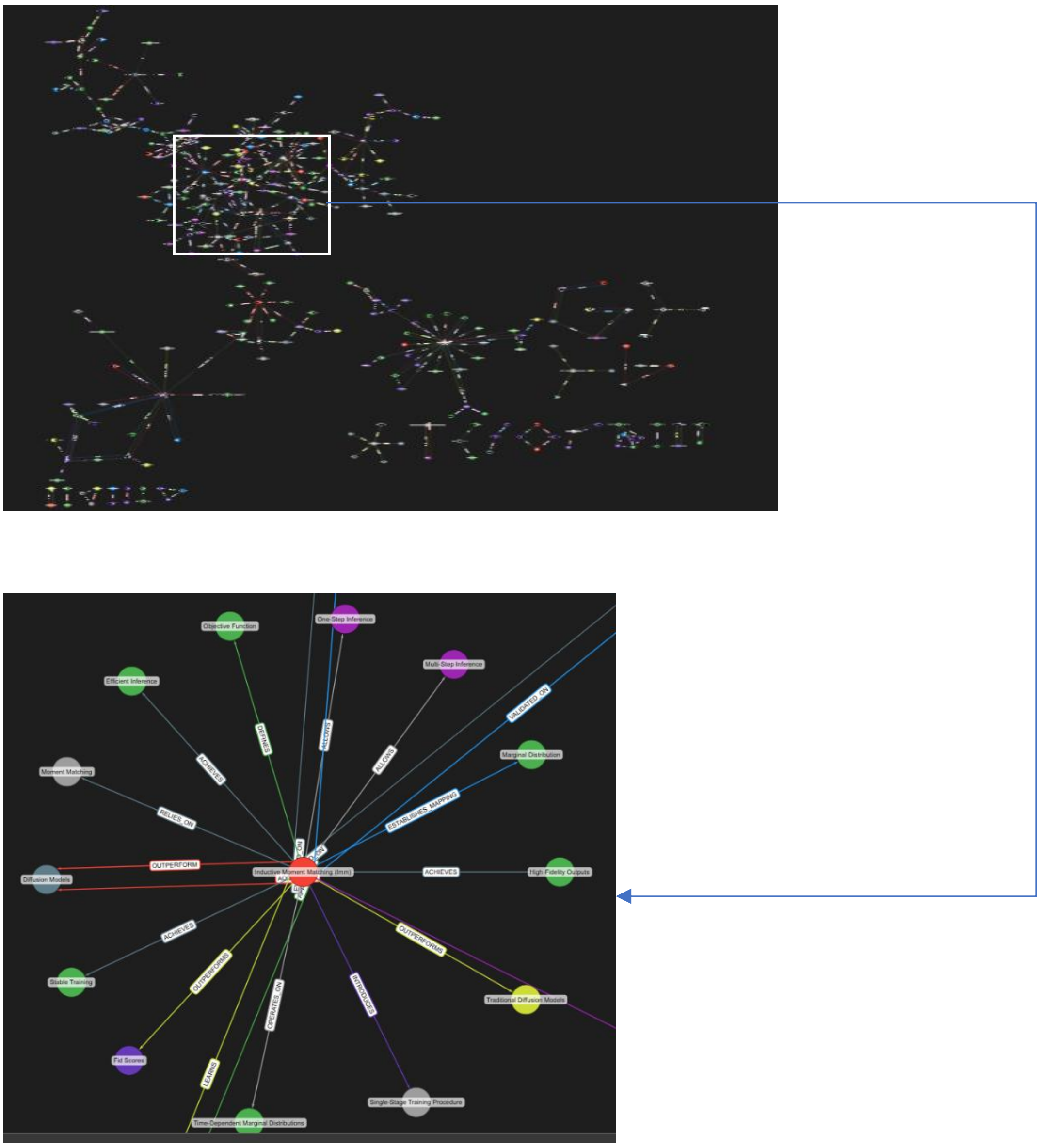
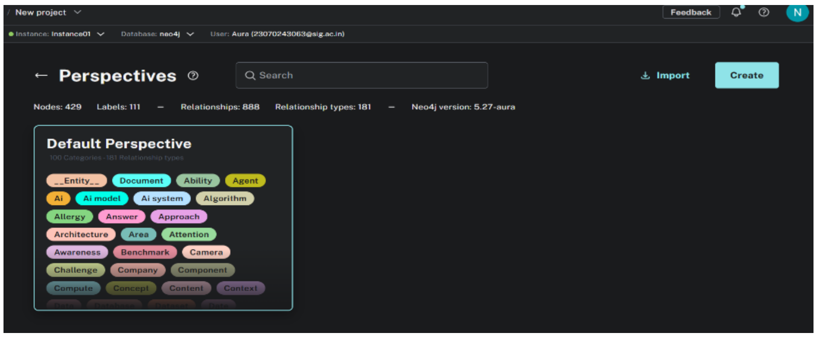


Fig 3.4 Created Knowledge graph



New project

Instance: Instance01 Database: neo4j User: Aura (23070243063@sig.ac.in)

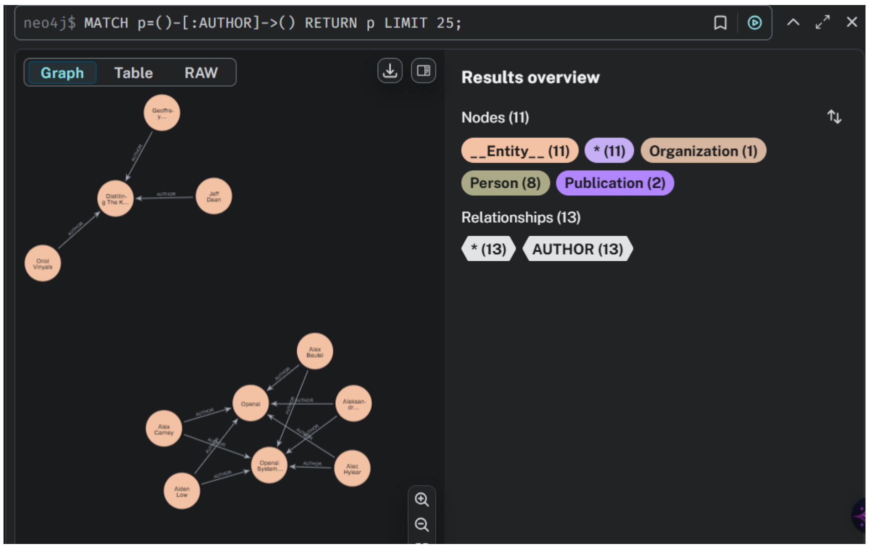
Perspectives

Nodes: 429 Labels: 111 — Relationships: 888 Relationship types: 181 — Neo4j version: 5.27-aura

Default Perspective

100 Categories - 181 Relationship types

Figure 3: Knowledge Graph creation and Enrichment:



```
neo4j$ MATCH p=()-[:AUTHOR]->() RETURN p LIMIT 25;
```

Results overview

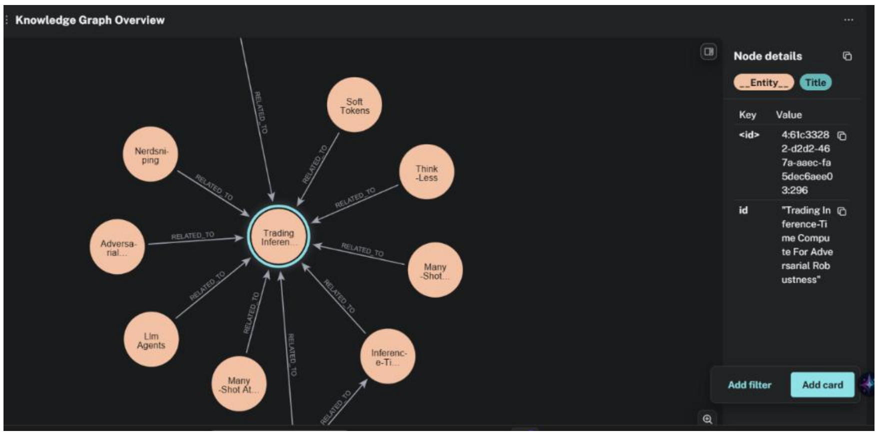
Nodes (11)

__Entity__ (11) * (11) Organization (1) Person (8) Publication (2)

Relationships (13)

* (13) AUTHOR (13)

Fig 4.1 query



Knowledge Graph Overview

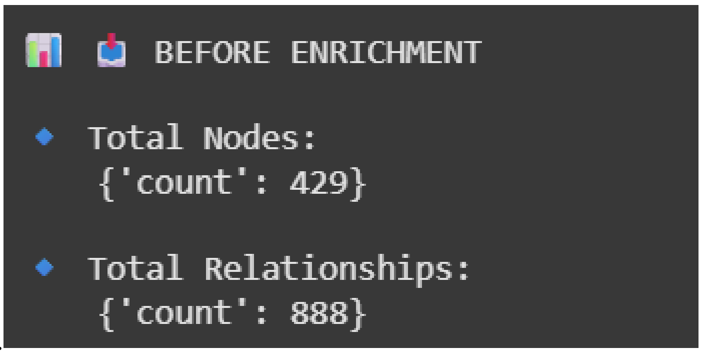
Node details

__Entity__ Title

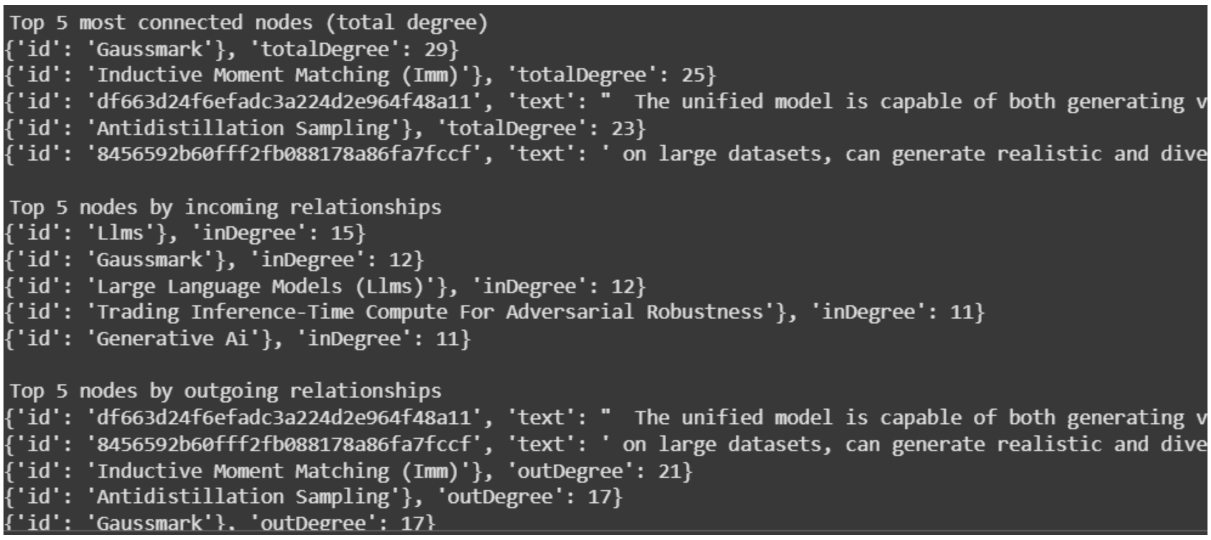
Key	Value
<id>	4:61c33282-d2d2-467a-aaec-fa5dec6aee03:296
id	"Trading Inference-Time Compute For Adversarial Robustness"

Fig 4.2 sample node

Figure 4 Sample queries

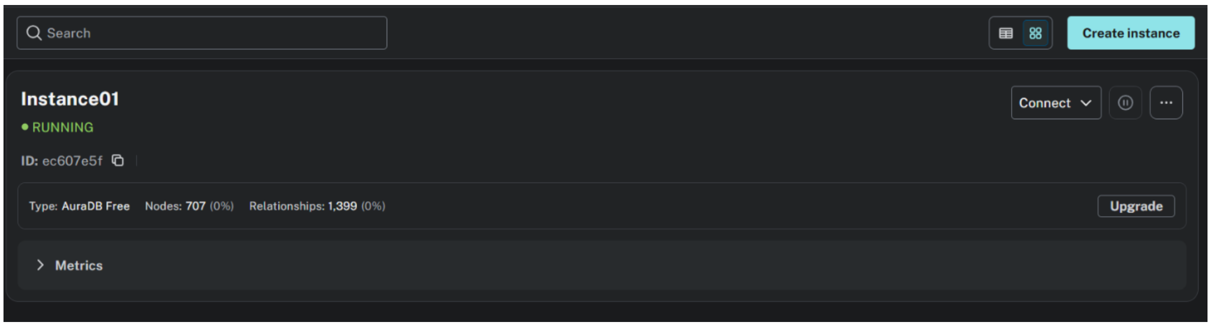


```
BEFORE ENRICHMENT

◆  Total Nodes:
    {'count': 429}

◆  Total Relationships:
    {'count': 888}
```



```
 Top 5 most connected nodes (total degree)
{'id': 'Gaussmark'}, 'totalDegree': 29}
{'id': 'Inductive Moment Matching (Imm)'}, 'totalDegree': 25}
{'id': 'df663d24f6efadc3a224d2e964f48a11', 'text': "  The unified model is capable of both generating v
{'id': 'Antidistillation Sampling'}, 'totalDegree': 23}
{'id': '8456592b60fff2fb088178a86fa7fccf', 'text': ' on large datasets, can generate realistic and dive

 Top 5 nodes by incoming relationships
{'id': 'Llms'}, 'inDegree': 15}
{'id': 'Gaussmark'}, 'inDegree': 12}
{'id': 'Large Language Models (Llms)'}, 'inDegree': 12}
{'id': 'Trading Inference-Time Compute For Adversarial Robustness'}, 'inDegree': 11}
{'id': 'Generative Ai'}, 'inDegree': 11}

 Top 5 nodes by outgoing relationships
{'id': 'df663d24f6efadc3a224d2e964f48a11', 'text': "  The unified model is capable of both generating v
{'id': '8456592b60fff2fb088178a86fa7fccf', 'text': ' on large datasets, can generate realistic and dive
{'id': 'Inductive Moment Matching (Imm)'}, 'outDegree': 21}
{'id': 'Antidistillation Sampling'}, 'outDegree': 17}
{'id': 'Gaussmark'}, 'outDegree': 17}
```

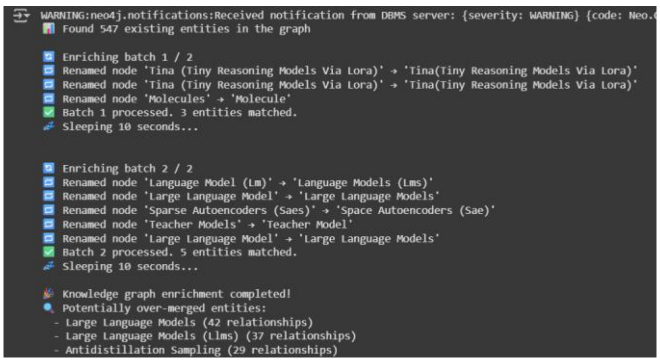
After Enrichment:



Instance01
● RUNNING
ID: ec607e5f

Type: AuraDB Free   Nodes: 707 (0%)   Relationships: 1,399 (0%)

Fig 5.1 instance having nodes and relationships



```
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.
📊 Found 547 existing entities in the graph

🔄 Enriching batch 1 / 2
🔁 Renamed node 'Tina (Tiny Reasoning Models Via Lora)' → 'Tina(Tiny Reasoning Models Via Lora)'
🔁 Renamed node 'Tina (Tiny Reasoning Models Via Lora)' → 'Tina(Tiny Reasoning Models Via Lora)'
🔁 Renamed node 'Molecules' → 'Molecule'
✅ Batch 1 processed. 3 entities matched.
💤 Sleeping 10 seconds...


🔄 Enriching batch 2 / 2
🔁 Renamed node 'Language Model (Lm)' → 'Language Models (Lms)'
🔁 Renamed node 'Large Language Model' → 'Large Language Models'
🔁 Renamed node 'Sparse Autoencoders (Saes)' → 'Space Autoencoders (Sae)'
🔁 Renamed node 'Teacher Models' → 'Teacher Model'
🔁 Renamed node 'Large Language Model' → 'Large Language Models'
✅ Batch 2 processed. 5 entities matched.
💤 Sleeping 10 seconds...

🎉 Knowledge graph enrichment completed!
🔍 Potentially over-merged entities:
  - Large Language Models (42 relationships)
  - Large Language Models (Llms) (37 relationships)
  - Antidistillation Sampling (29 relationships)
```

Fig 5.1 Enrichment out

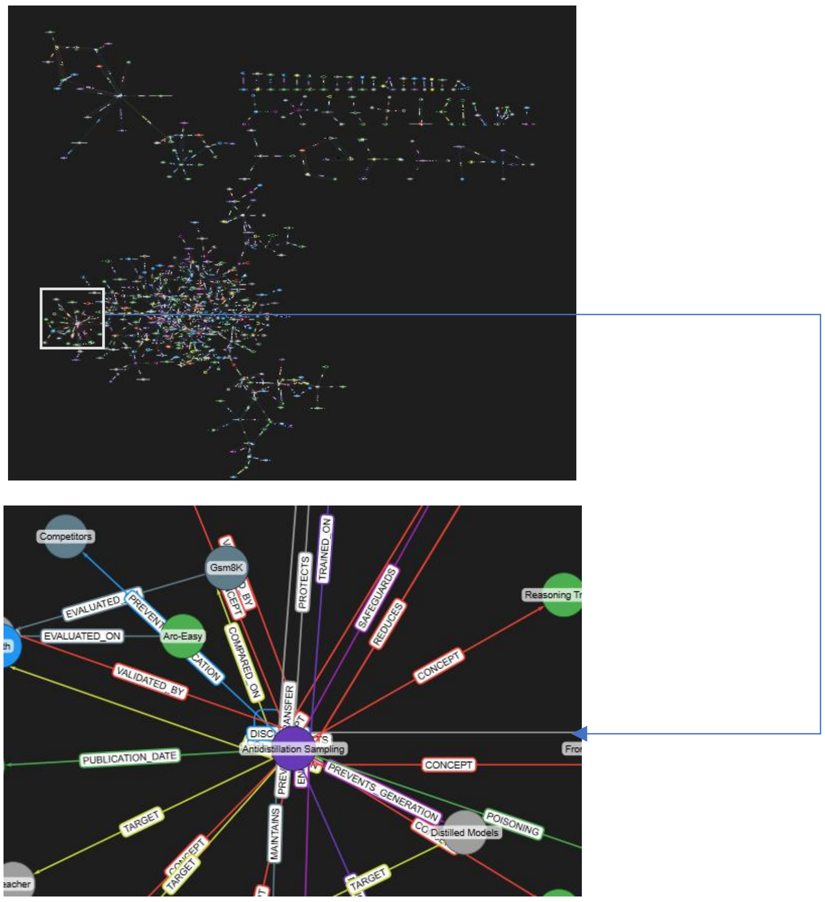


Fig 5.2 created graph after enrichment

Analysis:

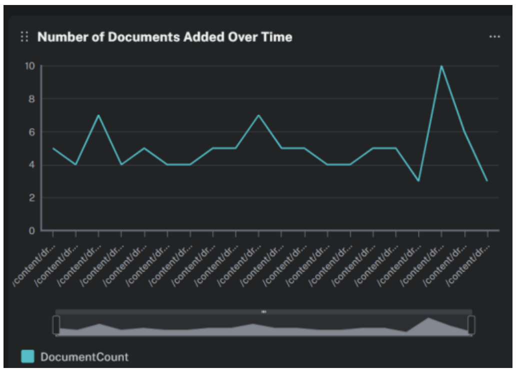
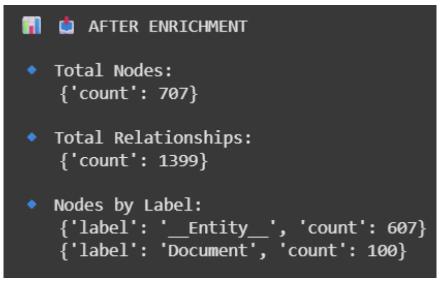
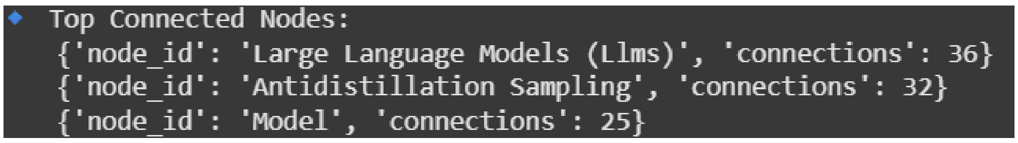


Fig 5.3 data added with time



```
AFTER ENRICHMENT

Total Nodes:
 {'count': 707}

Total Relationships:
 {'count': 1399}

Nodes by Label:
 {'label': '__Entity__', 'count': 607}
 {'label': 'Document', 'count': 100}
```

Fig 5.4 nodes and relationship creation after enrichment

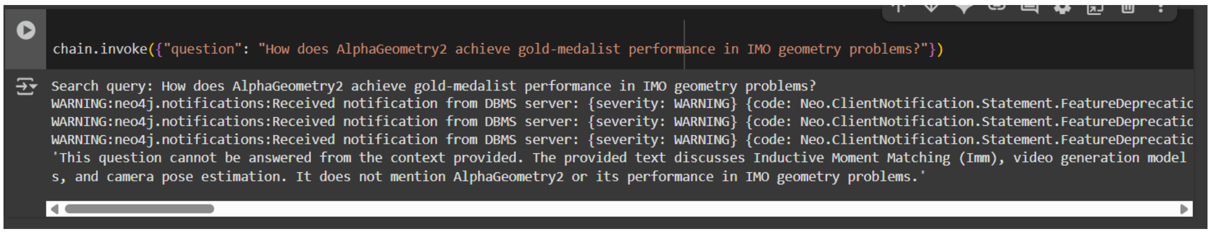


```
Top Connected Nodes:
 {'node_id': 'Large Language Models (Llms)', 'connections': 36}
 {'node_id': 'Antidistillation Sampling', 'connections': 32}
 {'node_id': 'Model', 'connections': 25}
```

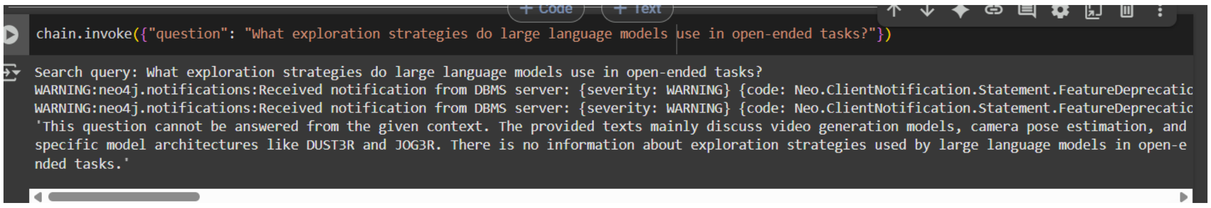
Fig 5.5 top connected nodes

Figure 5: Analysis:

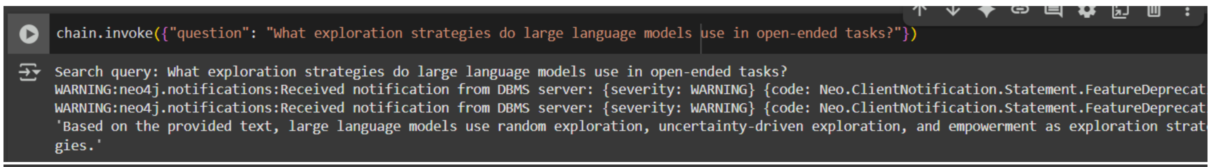
Before:



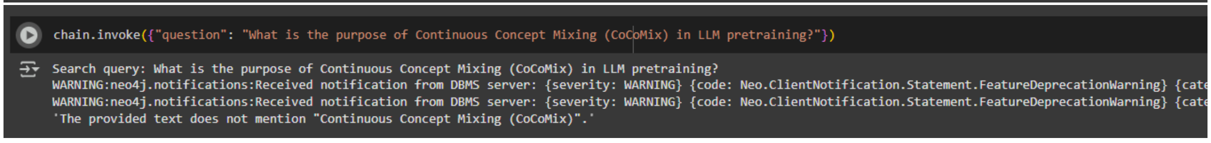
```
chain.invoke({"question": "How does AlphaGeometry2 achieve gold-medalist performance in IMO geometry problems?"})

Search query: How does AlphaGeometry2 achieve gold-medalist performance in IMO geometry problems?
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecatic
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecatic
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecatic
'This question cannot be answered from the context provided. The provided text discusses Inductive Moment Matching (Imm), video generation model
s, and camera pose estimation. It does not mention AlphaGeometry2 or its performance in IMO geometry problems.'
```



```
chain.invoke({"question": "What exploration strategies do large language models use in open-ended tasks?"})

Search query: What exploration strategies do large language models use in open-ended tasks?
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecatic
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecatic
'This question cannot be answered from the given context. The provided texts mainly discuss video generation models, camera pose estimation, and
specific model architectures like DUST3R and JOG3R. There is no information about exploration strategies used by large language models in open-e
nded tasks.'
```



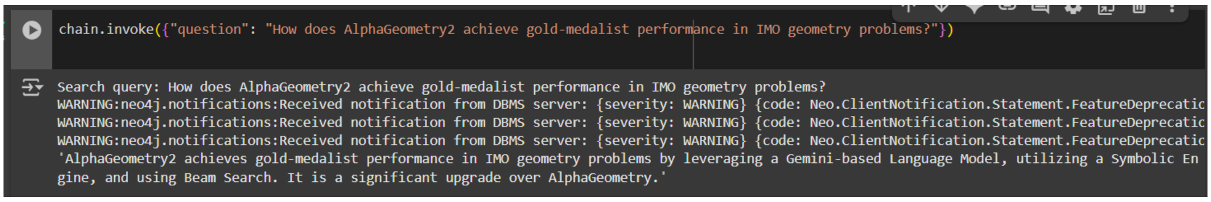
```
chain.invoke({"question": "What exploration strategies do large language models use in open-ended tasks?"})

Search query: What exploration strategies do large language models use in open-ended tasks?
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecat
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecat
'Based on the provided text, large language models use random exploration, uncertainty-driven exploration, and empowerment as exploration strat
gies.'
```



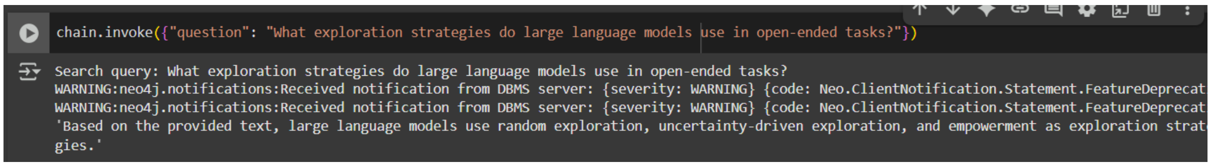
```
chain.invoke({"question": "What is the purpose of Continuous Concept Mixing (CoCoMix) in LLM pretraining?"})

Search query: What is the purpose of Continuous Concept Mixing (CoCoMix) in LLM pretraining?
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {cate
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {cate
'The provided text does not mention "Continuous Concept Mixing (CoCoMix)".'
```

Fig 6.1 Q&A before enrichment

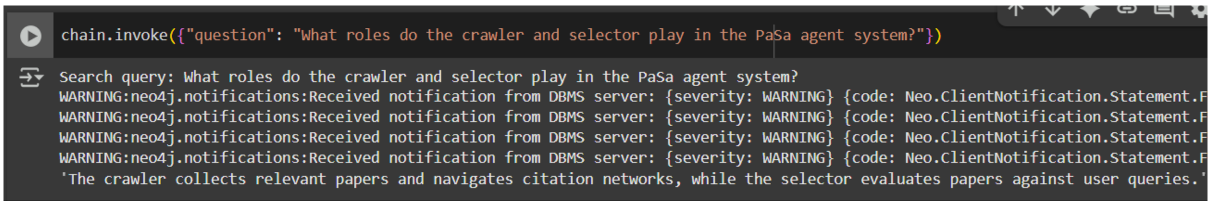
After:



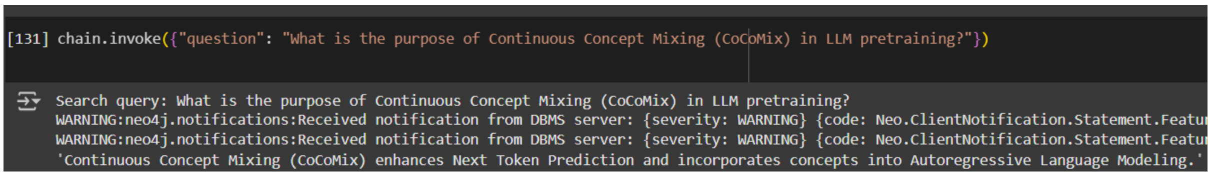
```
chain.invoke({"question": "How does AlphaGeometry2 achieve gold-medalist performance in IMO geometry problems?"})

Search query: How does AlphaGeometry2 achieve gold-medalist performance in IMO geometry problems?
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecatic
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecatic
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecatic
'AlphaGeometry2 achieves gold-medalist performance in IMO geometry problems by leveraging a Gemini-based Language Model, utilizing a Symbolic En
gine, and using Beam Search. It is a significant upgrade over AlphaGeometry.'
```



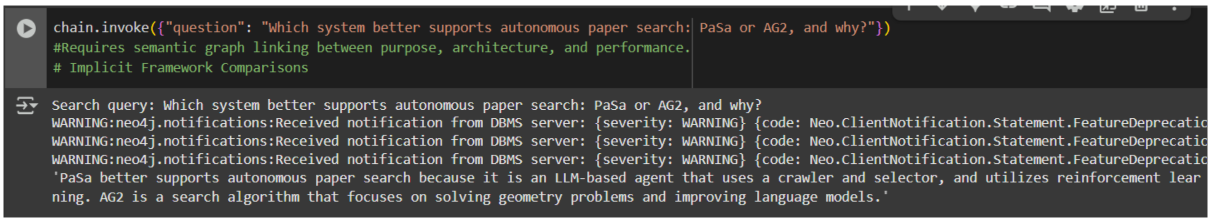
```
chain.invoke({"question": "What exploration strategies do large language models use in open-ended tasks?"})

Search query: What exploration strategies do large language models use in open-ended tasks?
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecat
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecat
'Based on the provided text, large language models use random exploration, uncertainty-driven exploration, and empowerment as exploration strat
gies.'
```



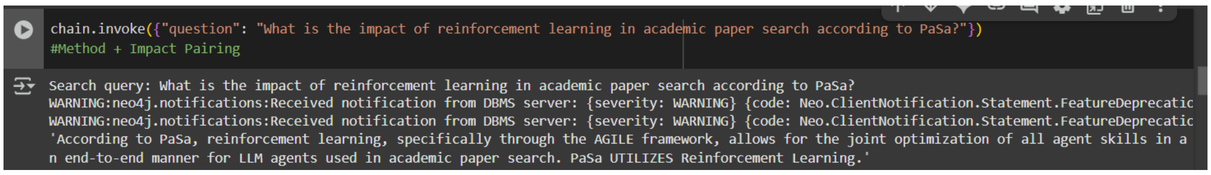
```
chain.invoke({"question": "What roles do the crawler and selector play in the PaSa agent system?"})

Search query: What roles do the crawler and selector play in the PaSa agent system?
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.F
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.F
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.F
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.F
'The crawler collects relevant papers and navigates citation networks, while the selector evaluates papers against user queries.'
```



```
[131] chain.invoke({"question": "What is the purpose of Continuous Concept Mixing (CoCoMix) in LLM pretraining?"})

Search query: What is the purpose of Continuous Concept Mixing (CoCoMix) in LLM pretraining?
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.Featu
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.Featu
'Continuous Concept Mixing (CoCoMix) enhances Next Token Prediction and incorporates concepts into Autoregressive Language Modeling.'
```



```
chain.invoke({"question": "Which system better supports autonomous paper search: PaSa or AG2, and why?"})
#Requires semantic graph linking between purpose, architecture, and performance.
# Implicit Framework Comparisons

Search query: Which system better supports autonomous paper search: PaSa or AG2, and why?
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecatic
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecatic
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecatic
'PaSa better supports autonomous paper search because it is an LLM-based agent that uses a crawler and selector, and utilizes reinforcement lear
ning. AG2 is a search algorithm that focuses on solving geometry problems and improving language models.'
```



```
chain.invoke({"question": "What is the impact of reinforcement learning in academic paper search according to PaSa?"})
#Method + Impact Pairing

Search query: What is the impact of reinforcement learning in academic paper search according to PaSa?
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecatic
WARNING:neo4j.notifications:Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecatic
'According to PaSa, reinforcement learning, specifically through the AGILE framework, allows for the joint optimization of all agent skills in a
n end-to-end manner for LLM agents used in academic paper search. PaSa UTILIZES Reinforcement Learning.'
```

Fig 6.2 Q&A after enrichment

Figure 6: Sample question testing: Before vs After Enrichment

Discussions

Analysis of the Results:

System Performance:
The system's ability to extract and organize data from research paper explanation videos was impressive. When incorporated into a multi-stage pipeline, the Gemini 2.0 Flash model efficiently condensed complex multimodal content and converted it into structured text for the creation of knowledge graphs. Together with Neo4j, the LangChain-based prompts correctly mapped entity relationships and semantic connections between research concepts. The gradual blending of new information during the enrichment stage demonstrated strong consistency, with enriched graphs representing a wider range of interrelated and varied AI subjects.

Graph Enrichment Quality:
Previously constructed graphs were successfully enhanced by the enrichment mechanism with little loss of information or redundancy. Even when related nodes and relationships had different wording in different summaries, semantic similarity matching based on Gemini embeddings assisted in combining them. As new summaries were added, the batchwise enrichment logs (kg_log) demonstrated increased concept density and interconnectivity. However, when key phrases were unclear or semantic overlap was too low, some enrichment failures happened.

Processing Pipeline Reliability:
Downloading, transcribing, summarizing, and gathering metadata were all smoothly coordinated by the MCP-based architecture. The video.mkv, summary.json, metadata.json, and transcript.txt files were consistently present in every YouTube video folder, facilitating repeatable downstream processing. The scalability and resilience of the pipeline were demonstrated through the batch processing of more than 400 files.

Challenges Encountered:

Unstructured Video Data Complexity:
Using YouTube videos as primary data presented a number of difficulties. In certain instances, transcription quality was impacted by changes in background noise, accent, and narration speed. Although summarization was handled well by the Gemini LLM, poor input quality could result in the loss of important details because it depends on the accuracy of the transcribed text.

Prompt Sensitivity and Consistency:
Prompt formulation during entity extraction and relationship creation caused a slight variation in LLM performance. Occasionally, subtle phrasing changes resulted in inconsistent or missing nodes across related documents. Although this problem was lessened, it was not completely resolved by fine-tuning the LangChain prompt templates.

Graph Enrichment Failures:
False negatives happened when there was concept overlap but the wording was too different, even though semantic similarity metrics performed well in many enrichment cases. Furthermore, it was more difficult to accurately link entities across documents when summaries lacked metadata like paper titles, authors, or publication year.

Potential Improvements:

Structured Metadata Injection:
By adding structured metadata (such as the arXiv ID, title, and author list) to the summaries before graph construction, future iterations could improve the quality of the graph. Better entity resolution and linking would be made possible by this.

LLM Prompt Robustness:
Consistency across various summaries could be enhanced by creating more resilient and modular LangChain prompt templates. It may be possible to lessen variation and hallucination by standardizing output formats and including few-shot examples.

Graph Quality Monitoring and Metrics:
Graph density, average node degree, and node centrality are examples of quantitative metrics that can be used to monitor structural changes and verify the effectiveness of enrichment. Future filtering and optimization techniques can also be guided by these statistics.

Interactive Dashboards for Graph Insight:
End users' interpretability and usability would be enhanced by utilizing Neo4j Aura's dashboard feature to visualize node types, relationship counts, and key concepts. The enrichment pipeline can incorporate prompts for creating these dashboards.

Chapter 6 Conclusion

This project's Intelligent Video Processing and Knowledge Enrichment System provides a new and modular approach to turning unstructured, multimodal research content—specifically, YouTube videos that describe research papers pertaining to artificial intelligence—into structured, queryable knowledge graphs. The system bridges the gap between informal video explanations and formalized AI knowledge representation by utilizing a Multi-Agent Control Protocol (MCP) framework for coordinated video ingestion and summarization, in conjunction with Gemini 2.0 Flash LLM for content understanding and LangChain + Neo4j for knowledge graph construction and enrichment.

By creating structured JSON summaries and turning them into rich semantic graphs, the implementation showed that the system could manage a sizable corpus of more than 400 videos with a research focus. By employing embedding-based similarity to guarantee semantic coherence, the enrichment component made it possible to gradually incorporate new information into preexisting graphs.

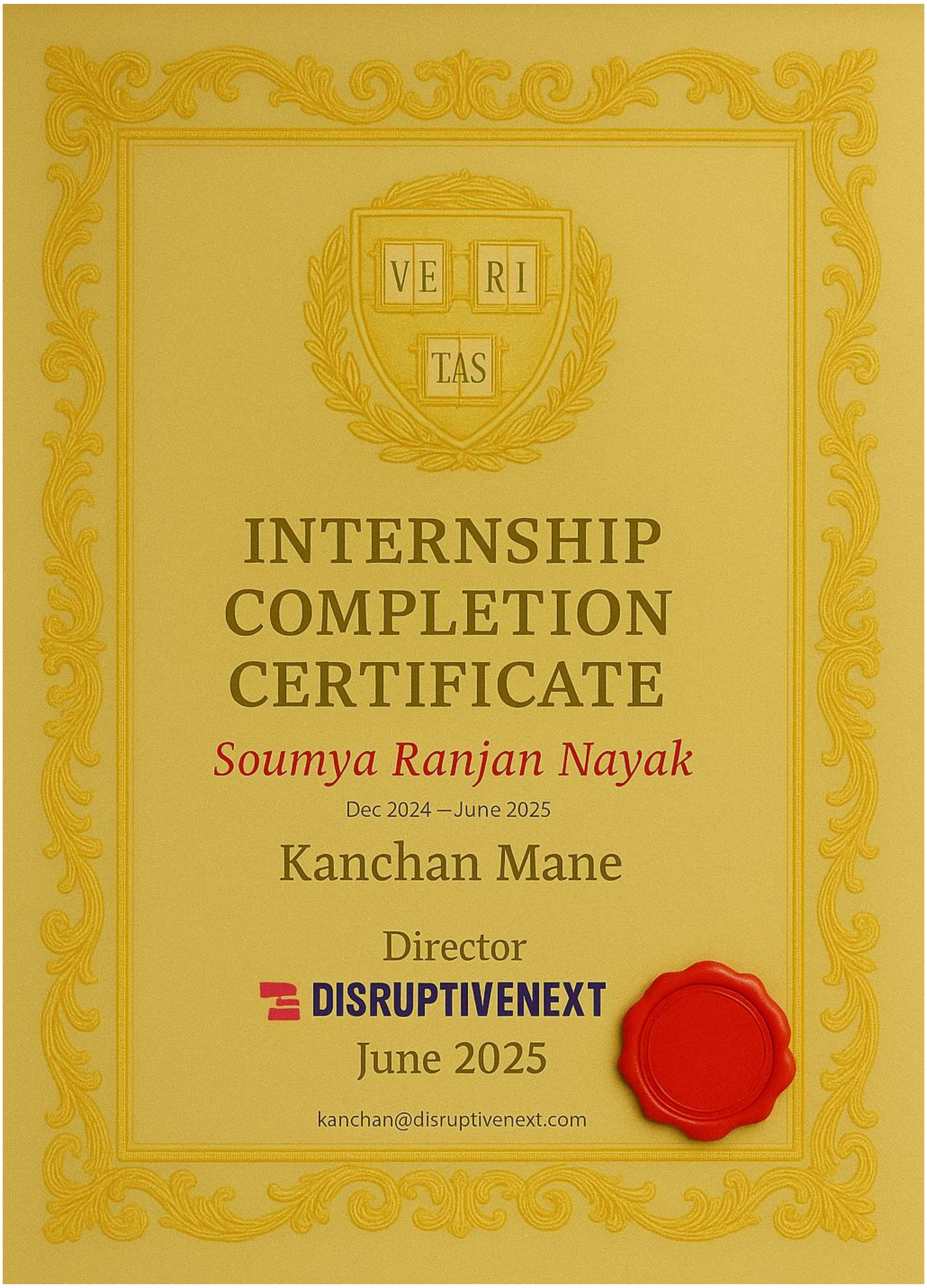
The architecture's viability and scalability were validated by the outcomes of both small- and large-scale tests. The system supported fault tolerance through effective batch processing and error handling, which made it appropriate for actual academic or research indexing workflows. The pipeline demonstrated resilience and flexibility in managing diverse input formats and content complexity, even with minor drawbacks in handling edge cases (e.g., unclear transcription, irregular summary structure).

All things considered, the system makes a significant contribution to the field of automated knowledge extraction, especially when dealing with multimodal AI content. Intelligent tutoring systems, research aggregators, and educational platforms that want to create knowledge-aware infrastructure from video-based learning materials can all use it.

References

1. Lu, Yuxing, and Jinzhuo Wang. "KARMA: Leveraging Multi-Agent LLMs for Automated Knowledge Graph Enrichment." *arXiv preprint arXiv:2502.06472* (2025).
2. Das, Taniya, Louis Mahon, and Thomas Lukasiewicz. "Detection-Fusion for Knowledge Graph Extraction from Videos." *arXiv preprint arXiv:2501.00136* (2024).
3. Deng, Jiaxin, et al. "A unified model for video understanding and knowledge embedding with heterogeneous knowledge graph dataset." *Proceedings of the 2023 ACM International Conference on Multimedia Retrieval*. 2023.
4. Zhang, Y., & Chen, L. (2024). *Document GraphRAG: Knowledge Graph-Guided Document Retrieval and Generation for Complex QA*. Proceedings of the AAAI Conference on Artificial Intelligence.
5. Chen, Xiang, et al. "Hybrid transformer with multi-level fusion for multimodal knowledge graph completion." *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval*. 2022.
6. Hu, Yuntong, et al. "Grag: Graph retrieval-augmented generation." *arXiv preprint arXiv:2405.16506* (2024).
7. Ma, Chuangtao, et al. "Large Language Models Meet Knowledge Graphs for Question Answering: Synthesis and Opportunities." *arXiv preprint arXiv:2505.20099* (2025)..
8. Sarmah, Bhaskarjit, et al. "Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction." *Proceedings of the 5th ACM International Conference on AI in Finance*. 2024.
9. Nelson, Ivar, and Oscar Andersson. "Enhancing Code Review at Scale with Generative AI and Knowledge Graphs: An Agentic GraphRAG Framework for Enterprise Code Review." (2025).
10. Zhou, Yingli, et al. "In-depth Analysis of Graph-based RAG in a Unified Framework." *arXiv preprint arXiv:2503.04338* (2025).
11. Xiang, Zhishang, et al. "When to use graphs in rag: A comprehensive analysis for graph retrieval-augmented generation." *arXiv preprint arXiv:2506.05690* (2025).
12. https://modelcontextprotocol.io/introduction
13. https://python.langchain.com/docs/how_to/graph_constructing/
14. https://arxiv.org/
15. https://www.youtube.com/@ArxivPapers)
16. https://neo4j.com/product/auradb/
17. https://ai.meta.com/tools/faiss/

Appendix



VE RI TAS

INTERNSHIP COMPLETION CERTIFICATE

Soumya Ranjan Nayak

Dec 2024 – June 2025

Kanchan Mane

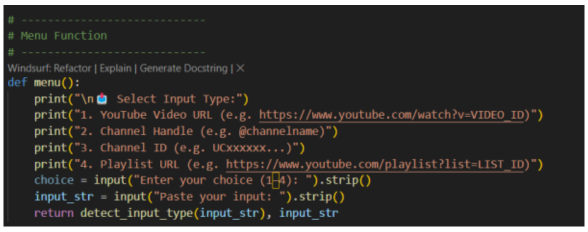
Director

DISRUPTIVENEXT

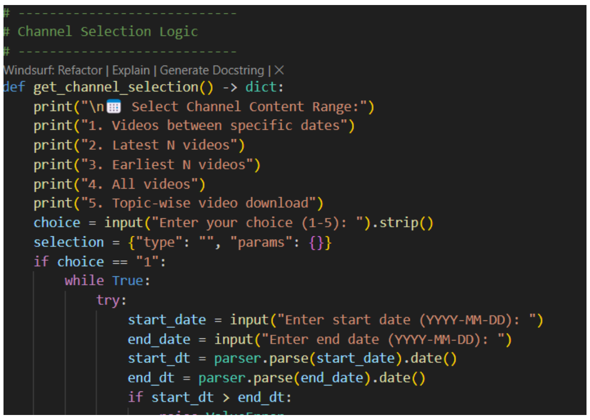
June 2025

kanchan@disruptivenext.com

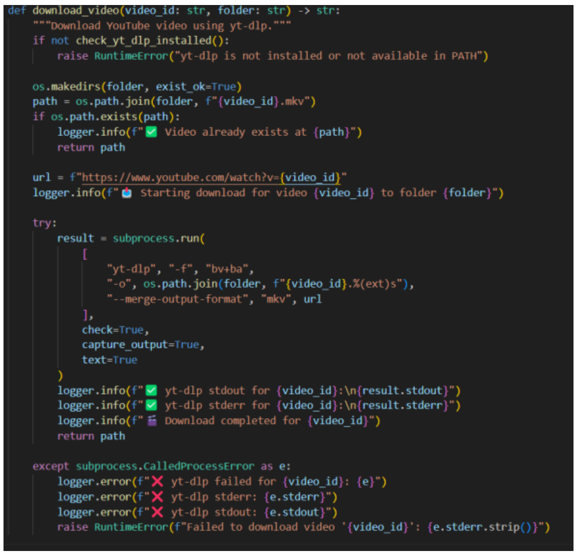
YouTube Content Downloading:



```python
# ---------------------------
# Menu Function
# ---------------------------
Windsurf: Refactor | Explain | Generate Docstring | ×
def menu():
    print("\n📥 Select Input Type:")
    print("1. YouTube Video URL (e.g. https://www.youtube.com/watch?v=VIDEO_ID)")
    print("2. Channel Handle (e.g. @channelname)")
    print("3. Channel ID (e.g. UCxxxxxx...)")
    print("4. Playlist URL (e.g. https://www.youtube.com/playlist?list=LIST_ID)")
    choice = input("Enter your choice (1-4): ").strip()
    input_str = input("Paste your input: ").strip()
    return detect_input_type(input_str), input_str
```

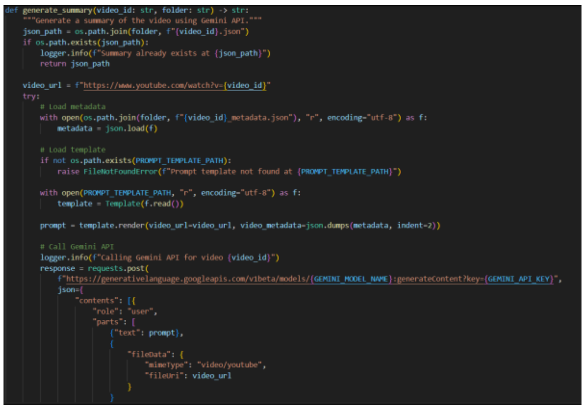


```python
# ---------------------------
# Channel Selection Logic
# ---------------------------
Windsurf: Refactor | Explain | Generate Docstring | ×
def get_channel_selection() -> dict:
    print("\n📅 Select Channel Content Range:")
    print("1. Videos between specific dates")
    print("2. Latest N videos")
    print("3. Earliest N videos")
    print("4. All videos")
    print("5. Topic-wise video download")
    choice = input("Enter your choice (1-5): ").strip()
    selection = {"type": "", "params": {}}
    if choice == "1":
        while True:
            try:
                start_date = input("Enter start date (YYYY-MM-DD): ")
                end_date = input("Enter end date (YYYY-MM-DD): ")
                start_dt = parser.parse(start_date).date()
                end_dt = parser.parse(end_date).date()
                if start_dt > end_dt:
```

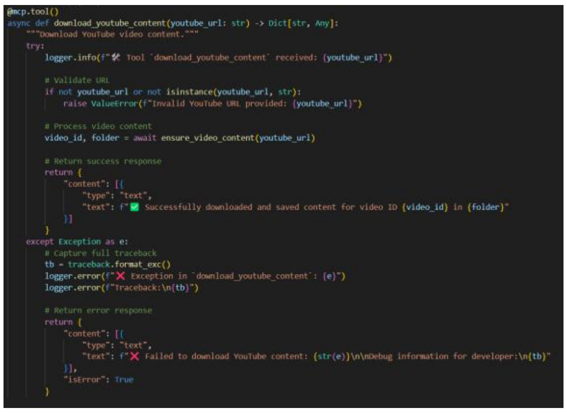


```python
def download_video(video_id: str, folder: str) -> str:
    """Download YouTube video using yt-dlp."""
    if not check_yt_dlp_installed():
        raise RuntimeError("yt-dlp is not installed or not available in PATH")

    os.makedirs(folder, exist_ok=True)
    path = os.path.join(folder, f"{video_id}.mkv")
    if os.path.exists(path):
        logger.info(f"✅ Video already exists at {path}")
        return path

    url = f"https://www.youtube.com/watch?v={video_id}"
    logger.info(f"📥 Starting download for video {video_id} to folder {folder}")

    try:
        result = subprocess.run(
            [
                "yt-dlp", "-f", "bv+ba",
                "-o", os.path.join(folder, f"{video_id}.%(ext)s"),
                "--merge-output-format", "mkv", url
            ],
            check=True,
            capture_output=True,
            text=True
        )
        logger.info(f"✅ yt-dlp stdout for {video_id}:\n{result.stdout}")
        logger.info(f"✅ yt-dlp stderr for {video_id}:\n{result.stderr}")
        logger.info(f"🎬 Download completed for {video_id}")
        return path

    except subprocess.CalledProcessError as e:
        logger.error(f"❌ yt-dlp failed for {video_id}: {e}")
        logger.error(f"❌ yt-dlp stderr: {e.stderr}")
        logger.error(f"❌ yt-dlp stdout: {e.stdout}")
        raise RuntimeError(f"Failed to download video '{video_id}': {e.stderr.strip()}")
```

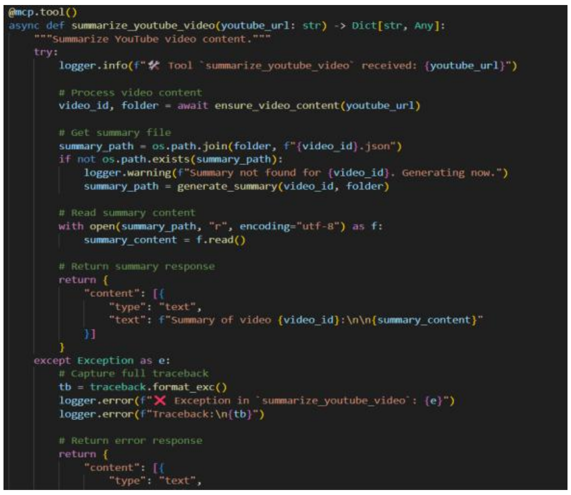
YouTube MCP server tools:



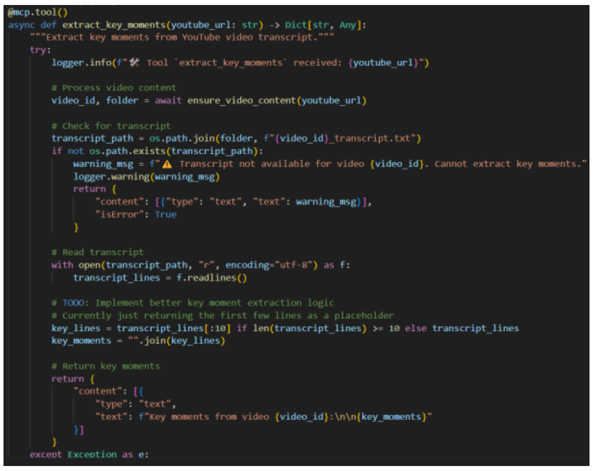
```python
def generate_summary(video_id: str, folder: str) -> str:
    """Generate a summary of the video using Gemini API."""
    json_path = os.path.join(folder, f"{video_id}.json")
    if os.path.exists(json_path):
        logger.info(f"Summary already exists at {json_path}")
        return json_path

    video_url = f"https://www.youtube.com/watch?v={video_id}"
    try:
        # Load metadata
        with open(os.path.join(folder, f"{video_id}_metadata.json"), "r", encoding="utf-8") as f:
            metadata = json.load(f)

        # Load template
        if not os.path.exists(PROMPT_TEMPLATE_PATH):
            raise FileNotFoundError(f"Prompt template not found at {PROMPT_TEMPLATE_PATH}")

        with open(PROMPT_TEMPLATE_PATH, "r", encoding="utf-8") as f:
            template = Template(f.read())

        prompt = template.render(video_url=video_url, video_metadata=json.dumps(metadata, indent=2))

        # Call Gemini API
        logger.info(f"Calling Gemini API for video {video_id}")
        response = requests.post(
            f"https://generativelanguage.googleapis.com/v1beta/models/{GEMINI_MODEL_NAME}:generateContent?key={GEMINI_API_KEY}",
            json={
                "contents": [{
                    "role": "user",
                    "parts": [
                        {"text": prompt},
                        {
                            "fileData": {
                                "mimeType": "video/youtube",
                                "fileUri": video_url
                            }
                        }
```



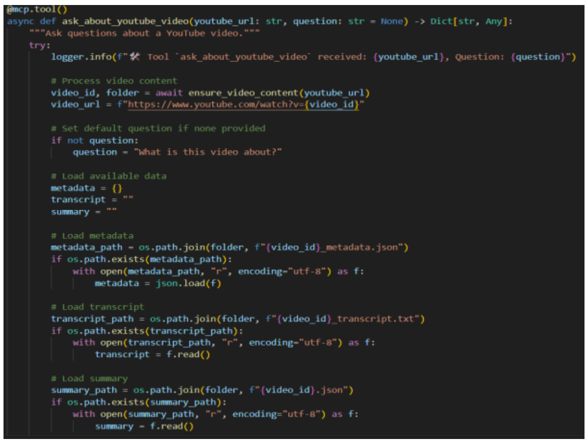
```python
@mcp.tool()
async def download_youtube_content(youtube_url: str) -> Dict[str, Any]:
    """Download YouTube video content."""
    try:
        logger.info(f"🛠 Tool `download_youtube_content` received: {youtube_url}")

        # Validate URL
        if not youtube_url or not isinstance(youtube_url, str):
            raise ValueError(f"Invalid YouTube URL provided: {youtube_url}")

        # Process video content
        video_id, folder = await ensure_video_content(youtube_url)

        # Return success response
        return {
            "content": [{
                "type": "text",
                "text": f"✅ Successfully downloaded and saved content for video ID {video_id} in {folder}"
            }]
        }
    except Exception as e:
        # Capture full traceback
        tb = traceback.format_exc()
        logger.error(f"❌ Exception in `download_youtube_content`: {e}")
        logger.error(f"Traceback:\n{tb}")

        # Return error response
        return {
            "content": [{
                "type": "text",
                "text": f"❌ Failed to download YouTube content: {str(e)}\n\nDebug information for developer:\n{tb}"
            }],
            "isError": True
        }
```



```python
@mcp.tool()
async def summarize_youtube_video(youtube_url: str) -> Dict[str, Any]:
    """Summarize YouTube video content."""
    try:
        logger.info(f"🛠 Tool `summarize_youtube_video` received: {youtube_url}")

        # Process video content
        video_id, folder = await ensure_video_content(youtube_url)

        # Get summary file
        summary_path = os.path.join(folder, f"{video_id}.json")
        if not os.path.exists(summary_path):
            logger.warning(f"Summary not found for {video_id}. Generating now.")
            summary_path = generate_summary(video_id, folder)

        # Read summary content
        with open(summary_path, "r", encoding="utf-8") as f:
            summary_content = f.read()

        # Return summary response
        return {
            "content": [{
                "type": "text",
                "text": f"Summary of video {video_id}:\n\n{summary_content}"
            }]
        }
    except Exception as e:
        # Capture full traceback
        tb = traceback.format_exc()
        logger.error(f"❌ Exception in `summarize_youtube_video`: {e}")
        logger.error(f"Traceback:\n{tb}")

        # Return error response
        return {
            "content": [{
                "type": "text",
```

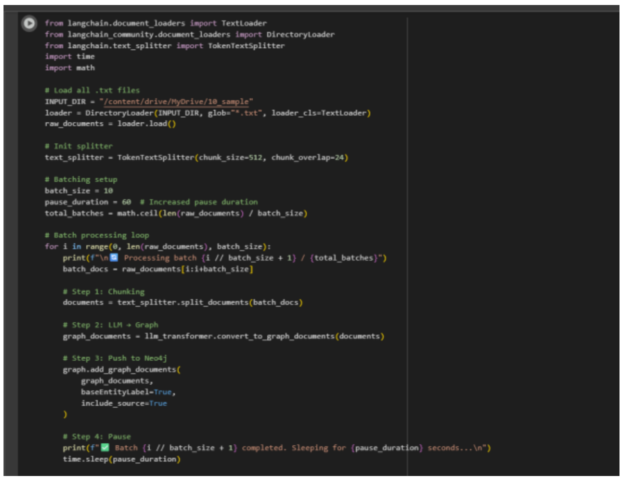


```python
@mcp.tool()
async def extract_key_moments(youtube_url: str) -> Dict[str, Any]:
    """Extract key moments from YouTube video transcript."""
    try:
        logger.info(f"🛠 Tool `extract_key_moments` received: {youtube_url}")

        # Process video content
        video_id, folder = await ensure_video_content(youtube_url)

        # Check for transcript
        transcript_path = os.path.join(folder, f"{video_id}_transcript.txt")
        if not os.path.exists(transcript_path):
            warning_msg = f"⚠ Transcript not available for video {video_id}. Cannot extract key moments."
            logger.warning(warning_msg)
            return {
                "content": [{"type": "text", "text": warning_msg}],
                "isError": True
            }

        # Read transcript
        with open(transcript_path, "r", encoding="utf-8") as f:
            transcript_lines = f.readlines()

        # TODO: Implement better key moment extraction logic
        # Currently just returning the first few lines as a placeholder
        key_lines = transcript_lines[:10] if len(transcript_lines) >= 10 else transcript_lines
        key_moments = "".join(key_lines)

        # Return key moments
        return {
            "content": [{
                "type": "text",
                "text": f"Key moments from video {video_id}:\n\n{key_moments}"
            }]
        }
    except Exception as e:
```



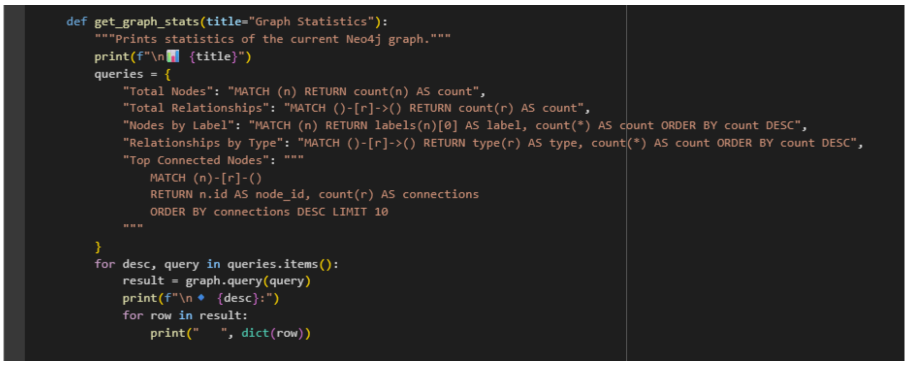
```python
@mcp.tool()
async def ask_about_youtube_video(youtube_url: str, question: str = None) -> Dict[str, Any]:
    """Ask questions about a YouTube video."""
    try:
        logger.info(f"🛠 Tool `ask_about_youtube_video` received: {youtube_url}, Question: {question}")

        # Process video content
        video_id, folder = await ensure_video_content(youtube_url)
        video_url = f"https://www.youtube.com/watch?v={video_id}"

        # Set default question if none provided
        if not question:
            question = "What is this video about?"

        # Load available data
        metadata = {}
        transcript = ""
        summary = ""

        # Load metadata
        metadata_path = os.path.join(folder, f"{video_id}_metadata.json")
        if os.path.exists(metadata_path):
            with open(metadata_path, "r", encoding="utf-8") as f:
                metadata = json.load(f)

        # Load transcript
        transcript_path = os.path.join(folder, f"{video_id}_transcript.txt")
        if os.path.exists(transcript_path):
            with open(transcript_path, "r", encoding="utf-8") as f:
                transcript = f.read()

        # Load summary
        summary_path = os.path.join(folder, f"{video_id}.json")
        if os.path.exists(summary_path):
            with open(summary_path, "r", encoding="utf-8") as f:
                summary = f.read()
```

Knowledge Graph Creation:

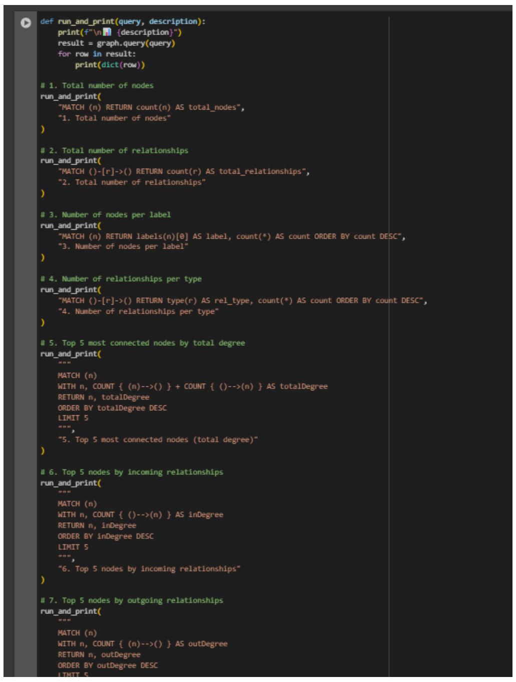


```python
from langchain.document_loaders import TextLoader
from langchain_community.document_loaders import DirectoryLoader
from langchain.text_splitter import TokenTextSplitter
import time
import math

# Load all .txt files
INPUT_DIR = "/content/drive/MyDrive/10_sample"
loader = DirectoryLoader(INPUT_DIR, glob="*.txt", loader_cls=TextLoader)
raw_documents = loader.load()

# Init splitter
text_splitter = TokenTextSplitter(chunk_size=512, chunk_overlap=24)

# Batching setup
batch_size = 10
pause_duration = 60  # Increased pause duration
total_batches = math.ceil(len(raw_documents) / batch_size)

# Batch processing loop
for i in range(0, len(raw_documents), batch_size):
    print(f"\n🔄 Processing batch {i // batch_size + 1} / {total_batches}")
    batch_docs = raw_documents[i:i+batch_size]

    # Step 1: Chunking
    documents = text_splitter.split_documents(batch_docs)

    # Step 2: LLM → Graph
    graph_documents = llm_transformer.convert_to_graph_documents(documents)

    # Step 3: Push to Neo4j
    graph.add_graph_documents(
        graph_documents,
        baseEntityLabel=True,
        include_source=True
    )

    # Step 4: Pause
    print(f"✅ Batch {i // batch_size + 1} completed. Sleeping for {pause_duration} seconds...\n")
    time.sleep(pause_duration)
```

Graph Analysis:

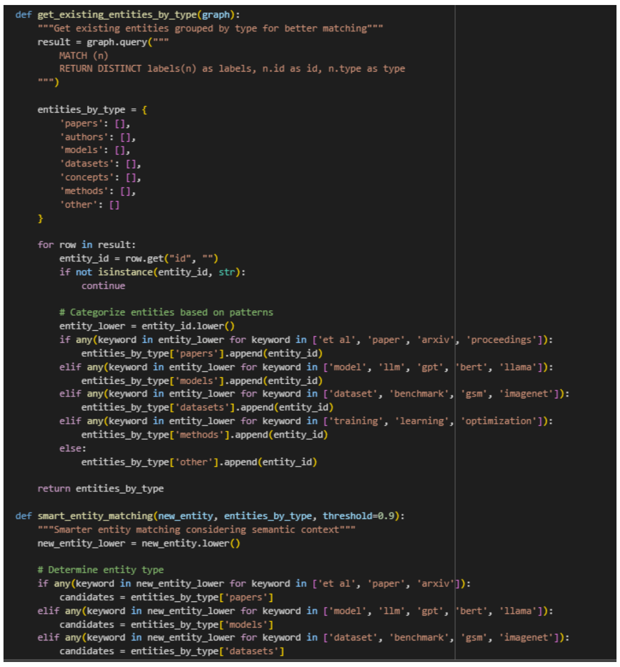


```python
def get_graph_stats(title="Graph Statistics"):
    """Prints statistics of the current Neo4j graph."""
    print(f"\n📊 {title}")
    queries = {
        "Total Nodes": "MATCH (n) RETURN count(n) AS count",
        "Total Relationships": "MATCH ()-[r]->() RETURN count(r) AS count",
        "Nodes by Label": "MATCH (n) RETURN labels(n)[0] AS label, count(*) AS count ORDER BY count DESC",
        "Relationships by Type": "MATCH ()-[r]->() RETURN type(r) AS type, count(*) AS count ORDER BY count DESC",
        "Top Connected Nodes": """
            MATCH (n)-[r]-()
            RETURN n.id AS node_id, count(r) AS connections
            ORDER BY connections DESC LIMIT 10
        """
    }
    for desc, query in queries.items():
        result = graph.query(query)
        print(f"\n🔹 {desc}:")
        for row in result:
            print("   ", dict(row))
```

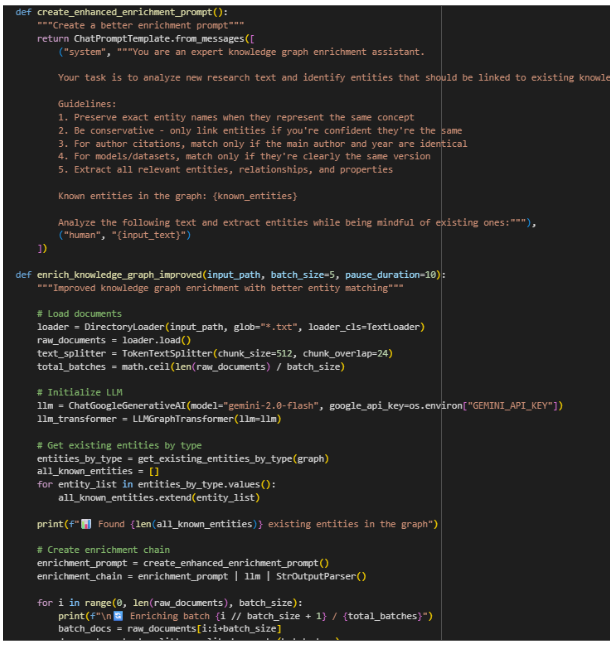


```python
def run_and_print(query, description):
    print(f"\n📊 {description}")
    result = graph.query(query)
    for row in result:
        print(dict(row))

# 1. Total number of nodes
run_and_print(
    "MATCH (n) RETURN count(n) AS total_nodes",
    "1. Total number of nodes"
)

# 2. Total number of relationships
run_and_print(
    "MATCH ()-[r]->() RETURN count(r) AS total_relationships",
    "2. Total number of relationships"
)

# 3. Number of nodes per label
run_and_print(
    "MATCH (n) RETURN labels(n)[0] AS label, count(*) AS count ORDER BY count DESC",
    "3. Number of nodes per label"
)

# 4. Number of relationships per type
run_and_print(
    "MATCH ()-[r]->() RETURN type(r) AS rel_type, count(*) AS count ORDER BY count DESC",
    "4. Number of relationships per type"
)

# 5. Top 5 most connected nodes by total degree
run_and_print(
    """
    MATCH (n)
    WITH n, COUNT { (n)-->() } + COUNT { ()-->(n) } AS totalDegree
    RETURN n, totalDegree
    ORDER BY totalDegree DESC
    LIMIT 5
    """,
    "5. Top 5 most connected nodes (total degree)"
)

# 6. Top 5 nodes by incoming relationships
run_and_print(
    """
    MATCH (n)
    WITH n, COUNT { ()-->(n) } AS inDegree
    RETURN n, inDegree
    ORDER BY inDegree DESC
    LIMIT 5
    """,
    "6. Top 5 nodes by incoming relationships"
)

# 7. Top 5 nodes by outgoing relationships
run_and_print(
    """
    MATCH (n)
    WITH n, COUNT { (n)-->() } AS outDegree
    RETURN n, outDegree
    ORDER BY outDegree DESC
    LIMIT 5
```

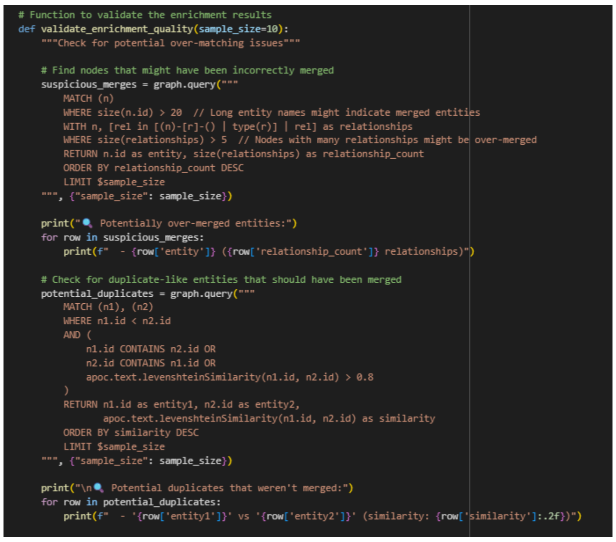
Knowledge Graph Enrichment: Manual Categorisation:



```python
def get_existing_entities_by_type(graph):
    """Get existing entities grouped by type for better matching"""
    result = graph.query("""
        MATCH (n)
        RETURN DISTINCT labels(n) as labels, n.id as id, n.type as type
    """)

    entities_by_type = {
        'papers': [],
        'authors': [],
        'models': [],
        'datasets': [],
        'concepts': [],
        'methods': [],
        'other': []
    }

    for row in result:
        entity_id = row.get("id", "")
        if not isinstance(entity_id, str):
            continue

        # Categorize entities based on patterns
        entity_lower = entity_id.lower()
        if any(keyword in entity_lower for keyword in ['et al', 'paper', 'arxiv', 'proceedings']):
            entities_by_type['papers'].append(entity_id)
        elif any(keyword in entity_lower for keyword in ['model', 'llm', 'gpt', 'bert', 'llama']):
            entities_by_type['models'].append(entity_id)
        elif any(keyword in entity_lower for keyword in ['dataset', 'benchmark', 'gsm', 'imagenet']):
            entities_by_type['datasets'].append(entity_id)
        elif any(keyword in entity_lower for keyword in ['training', 'learning', 'optimization']):
            entities_by_type['methods'].append(entity_id)
        else:
            entities_by_type['other'].append(entity_id)

    return entities_by_type

def smart_entity_matching(new_entity, entities_by_type, threshold=0.9):
    """Smarter entity matching considering semantic context"""
    new_entity_lower = new_entity.lower()

    # Determine entity type
    if any(keyword in new_entity_lower for keyword in ['et al', 'paper', 'arxiv']):
        candidates = entities_by_type['papers']
    elif any(keyword in new_entity_lower for keyword in ['model', 'llm', 'gpt', 'bert', 'llama']):
        candidates = entities_by_type['models']
    elif any(keyword in new_entity_lower for keyword in ['dataset', 'benchmark', 'gsm', 'imagenet']):
        candidates = entities_by_type['datasets']
```

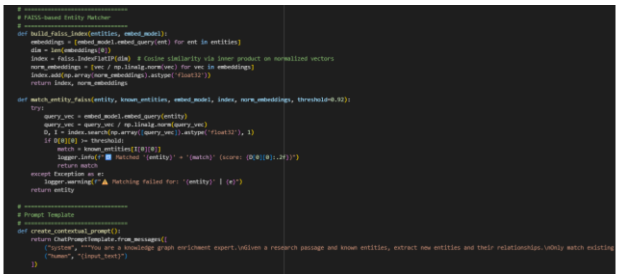


```python
def create_enhanced_enrichment_prompt():
    """Create a better enrichment prompt"""
    return ChatPromptTemplate.from_messages([
        ("system", """You are an expert knowledge graph enrichment assistant.

        Your task is to analyze new research text and identify entities that should be linked to existing knowle

        Guidelines:
        1. Preserve exact entity names when they represent the same concept
        2. Be conservative - only link entities if you're confident they're the same
        3. For author citations, match only if the main author and year are identical
        4. For models/datasets, match only if they're clearly the same version
        5. Extract all relevant entities, relationships, and properties

        Known entities in the graph: {known_entities}

        Analyze the following text and extract entities while being mindful of existing ones:"""),
        ("human", "{input_text}")
    ])

def enrich_knowledge_graph_improved(input_path, batch_size=5, pause_duration=10):
    """Improved knowledge graph enrichment with better entity matching"""

    # Load documents
    loader = DirectoryLoader(input_path, glob="*.txt", loader_cls=TextLoader)
    raw_documents = loader.load()
    text_splitter = TokenTextSplitter(chunk_size=512, chunk_overlap=24)
    total_batches = math.ceil(len(raw_documents) / batch_size)

    # Initialize LLM
    llm = ChatGoogleGenerativeAI(model="gemini-2.0-flash", google_api_key=os.environ["GEMINI_API_KEY"])
    llm_transformer = LLMGraphTransformer(llm=llm)

    # Get existing entities by type
    entities_by_type = get_existing_entities_by_type(graph)
    all_known_entities = []
    for entity_list in entities_by_type.values():
        all_known_entities.extend(entity_list)

    print(f"📊 Found {len(all_known_entities)} existing entities in the graph")

    # Create enrichment chain
    enrichment_prompt = create_enhanced_enrichment_prompt()
    enrichment_chain = enrichment_prompt | llm | StrOutputParser()

    for i in range(0, len(raw_documents), batch_size):
        print(f"\n🔄 Enriching batch {i // batch_size + 1} / {total_batches}")
        batch_docs = raw_documents[i:i+batch_size]
```

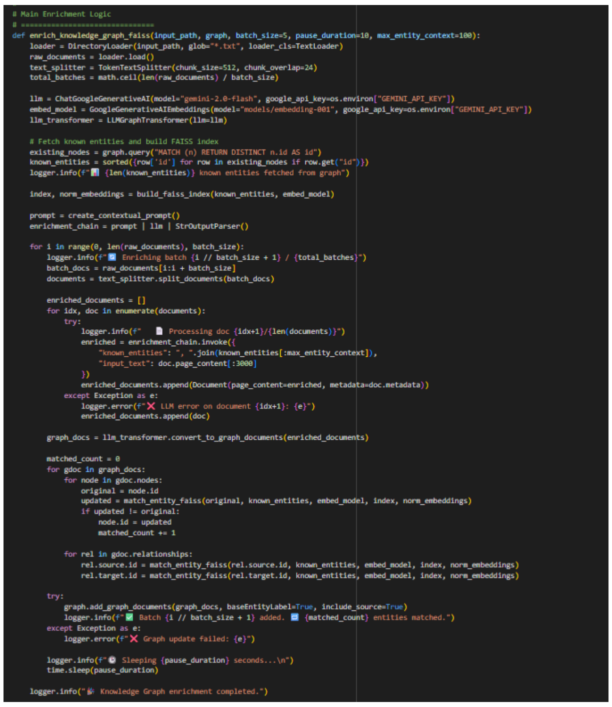


```python
# Function to validate the enrichment results
def validate_enrichment_quality(sample_size=10):
    """Check for potential over-matching issues"""

    # Find nodes that might have been incorrectly merged
    suspicious_merges = graph.query("""
        MATCH (n)
        WHERE size(n.id) > 20  // Long entity names might indicate merged entities
        WITH n, [rel in [(n)-[r]-() | type(r)] | rel] as relationships
        WHERE size(relationships) > 5  // Nodes with many relationships might be over-merged
        RETURN n.id as entity, size(relationships) as relationship_count
        ORDER BY relationship_count DESC
        LIMIT $sample_size
    """, {"sample_size": sample_size})

    print("🔍 Potentially over-merged entities:")
    for row in suspicious_merges:
        print(f"  - {row['entity']} ({row['relationship_count']} relationships)")

    # Check for duplicate-like entities that should have been merged
    potential_duplicates = graph.query("""
        MATCH (n1), (n2)
        WHERE n1.id < n2.id
        AND (
            n1.id CONTAINS n2.id OR
            n2.id CONTAINS n1.id OR
            apoc.text.levenshteinSimilarity(n1.id, n2.id) > 0.8
        )
        RETURN n1.id as entity1, n2.id as entity2,
               apoc.text.levenshteinSimilarity(n1.id, n2.id) as similarity
        ORDER BY similarity DESC
        LIMIT $sample_size
    """, {"sample_size": sample_size})

    print("\n🔍 Potential duplicates that weren't merged:")
    for row in potential_duplicates:
        print(f"  - '{row['entity1']}' vs '{row['entity2']}' (similarity: {row['similarity']:.2f})")
```

Knowledge Graph Enrichment: LLM Biased



```python
# ==============================
# FAISS-based Entity Matcher
# ==============================
def build_faiss_index(entities, embed_model):
    embeddings = [embed_model.embed_query(ent) for ent in entities]
    dim = len(embeddings[0])
    index = faiss.IndexFlatIP(dim)  # Cosine similarity via inner product on normalized vectors
    norm_embeddings = [vec / np.linalg.norm(vec) for vec in embeddings]
    index.add(np.array(norm_embeddings).astype('float32'))
    return index, norm_embeddings

def match_entity_faiss(entity, known_entities, embed_model, index, norm_embeddings, threshold=0.92):
    try:
        query_vec = embed_model.embed_query(entity)
        query_vec = query_vec / np.linalg.norm(query_vec)
        D, I = index.search(np.array([query_vec]).astype('float32'), 1)
        if D[0][0] >= threshold:
            match = known_entities[I[0][0]]
            logger.info(f"🔁 Matched '{entity}' → '{match}' (score: {D[0][0]:.2f})")
            return match
    except Exception as e:
        logger.warning(f"⚠️ Matching failed for: '{entity}' | {e}")
    return entity

# ==============================
# Prompt Template
# ==============================
def create_contextual_prompt():
    return ChatPromptTemplate.from_messages([
        ("system", """You are a knowledge graph enrichment expert.\nGiven a research passage and known entities, extract new entities and their relationships.\nOnly match existing
        ("human", "{input_text}")
    ])
```



```python
# Main Enrichment Logic
# ==============================
def enrich_knowledge_graph_faiss(input_path, graph, batch_size=5, pause_duration=10, max_entity_context=100):
    loader = DirectoryLoader(input_path, glob="*.txt", loader_cls=TextLoader)
    raw_documents = loader.load()
    text_splitter = TokenTextSplitter(chunk_size=512, chunk_overlap=24)
    total_batches = math.ceil(len(raw_documents) / batch_size)

    llm = ChatGoogleGenerativeAI(model="gemini-2.0-flash", google_api_key=os.environ["GEMINI_API_KEY"])
    embed_model = GoogleGenerativeAIEmbeddings(model="models/embedding-001", google_api_key=os.environ["GEMINI_API_KEY"])
    llm_transformer = LLMGraphTransformer(llm=llm)

    # Fetch known entities and build FAISS index
    existing_nodes = graph.query("MATCH (n) RETURN DISTINCT n.id AS id")
    known_entities = sorted({row['id'] for row in existing_nodes if row.get("id")})
    logger.info(f"📊 {len(known_entities)} known entities fetched from graph")

    index, norm_embeddings = build_faiss_index(known_entities, embed_model)

    prompt = create_contextual_prompt()
    enrichment_chain = prompt | llm | StrOutputParser()

    for i in range(0, len(raw_documents), batch_size):
        logger.info(f"🔄 Enriching batch {i // batch_size + 1} / {total_batches}")
        batch_docs = raw_documents[i:i + batch_size]
        documents = text_splitter.split_documents(batch_docs)

        enriched_documents = []
        for idx, doc in enumerate(documents):
            try:
                logger.info(f"   📄 Processing doc {idx+1}/{len(documents)}")
                enriched = enrichment_chain.invoke({
                    "known_entities": ", ".join(known_entities[:max_entity_context]),
                    "input_text": doc.page_content[:3000]
                })
                enriched_documents.append(Document(page_content=enriched, metadata=doc.metadata))
            except Exception as e:
                logger.error(f"❌ LLM error on document {idx+1}: {e}")
                enriched_documents.append(doc)

        graph_docs = llm_transformer.convert_to_graph_documents(enriched_documents)

        matched_count = 0
        for gdoc in graph_docs:
            for node in gdoc.nodes:
                original = node.id
                updated = match_entity_faiss(original, known_entities, embed_model, index, norm_embeddings)
                if updated != original:
                    node.id = updated
                    matched_count += 1

            for rel in gdoc.relationships:
                rel.source.id = match_entity_faiss(rel.source.id, known_entities, embed_model, index, norm_embeddings)
                rel.target.id = match_entity_faiss(rel.target.id, known_entities, embed_model, index, norm_embeddings)

        try:
            graph.add_graph_documents(graph_docs, baseEntityLabel=True, include_source=True)
            logger.info(f"✅ Batch {i // batch_size + 1} added. 🔁 {matched_count} entities matched.")
        except Exception as e:
            logger.error(f"❌ Graph update failed: {e}")

        logger.info(f"🕒 Sleeping {pause_duration} seconds...\n")
        time.sleep(pause_duration)

    logger.info("🎉 Knowledge Graph enrichment completed.")
```

Undertaking from the PG student while submitting his/her final dissertation to his respective institute

Ref. No. 23070243063

I , the following student

Sr. No.	**Sequence of students names on a dissertation**	**Students name**	**Name of the Institute & Place**	**Email & Mobile**
1.	First Author	Soumya Ranjan Nayak	SIG	Email: 23070243063@sig.ac.in Mobile: 6371347272

Note: Put additional rows in case of more number of students

hereby give an undertaking that the dissertation **Intelligent Video Processing And Knowledge Enrichment System** been checked for its Similarity Index/Plagiarism through Turnitin software tool; and that the document has been prepared by me and it is my original work and free of any plagiarism. It was found that:

1.	The Similarity Index (SI) was: *(Note: SI range: 0 to 10%; if SI is >10%, then authors cannot communicate ms;* ***attachment of SI report is mandatory)***	3 %
2.	The ethical clearance for research work conducted obtained from: *(Note: Name the consent obtaining body; if 'not appliable' then write so)*	NA
3.	The source of funding for research was: *(Note: Name the funding agency; or write 'self' if no funding source is involved)*	Self
4.	Conflict of interest: *(Note: Tick √ whichever is applicable)*	No
5.	The material (adopted text, tables, figures, graphs, etc.) as has been obtained from other sources, has been duly acknowledged in the manuscript: *(Note: Tick √ whichever is applicable)*	Yes

In case if any of the above-furnished information is found false at any point in time, then the University authorities can take action as deemed fit against all of us.

Soumya Ranjan Nayak
Signature of the student

Mr. Sahil Shah
Signature of SIU Guide/Mentor

Date: 12-07-2025

Endorsement by
Academic Integrity Committee (AIC)

Place: Pune

Note: It is mandatory that the Similarity Index report of plagiarism (only first page) should be appended to the UG/PG dissertation

23070243063_final_project_report.docx

ORIGINALITY REPORT

3%	3%	1%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	louisdl.louislibraries.org Internet Source	1%
2	Submitted to Symbiosis International University Student Paper	1%
3	scholarworks.waldenu.edu Internet Source	<1%
4	www.researchsquare.com Internet Source	<1%
5	trace.tennessee.edu Internet Source	<1%
6	www.theseus.fi Internet Source	<1%
7	S. Sutor, F. Matusek, R. Reda. "WSSU: High Performance Wireless Self-Contained, Surveillance Unit; an Ad Hoc Video Surveillance System", 2008 Fourth Advanced International Conference on Telecommunications, 2008 Publication	<1%