Vid2Insight: A Video Specialist Architecture

Aditya Mukhopadhyay[a], Ankit Kumar[b], Danish Akhlaq[c], Sanjay Kumar[d] and Trupti Rushikesh Kurambhatti[e]

Abstract. With the rise of instructional and product-related video content, there is a growing need for automated systems that can extract, structure, and repurpose information from multimedia streams. We present Vid2Insight, a modular video understanding system that constructs a multimodal knowledge base and supports downstream business and educational applications out of many such use cases. The pipeline ingests user-uploaded videos through a Streamlit interface, splits them into visual and audio segments, and generates detailed frame-level and transcript-level annotations using Gemini LLM.

The resulting multimodal data is stored as structured metadata and reused to optimize token consumption. On top of this knowledge base, we implement four specialised nodes: Product Documentation Generator, Executive Summary Generator, MCQ Generator, and Study Planner—using LangChain and LangGraph divided across two subgraphs. Each agent leverages LLM prompting and is validated via an LLM-as-a-judge approach to ensure factual and structural quality. Our system demonstrates strong alignment with expected outputs using BERTScore, ROUGE, and precision metrics, and offers a scalable framework for enterprise and educational video analytics. This analysis is placed at ingestion level & each specialist response level as well. Our architecture is currently designed to handle free-tier LLMs with rate limits for simple wide scale adoptions

1 Introduction

1.1 Context and Motivation

With the ever-growing volume of instructional and product-related videos available online, there is a pressing need for automated systems that can quickly and accurately extract and organize knowledge from such multimedia content. Traditional video indexing approaches rely heavily on manual annotation or simple speech-to-text pipelines, which often miss nuances present in visual frames or lack the flexibility to generate diverse textual outputs. Using recent advances in large language models (LLMs) and multimodal processing, our project aims to bridge this gap: we propose an end-to-end architecture that ingests arbitrary videos, constructs a unified knowledge base of audio and visual information, and then empowers specialized agents to fulfill distinct business needs.

Figure 1. AI generated Project Logo

1.2 Problem Statement

We identify two primary use cases that drive our design:

1. **Product Documentation Writer.** Many organizations require quick, accurate, and customizable product manuals or user guides derived from demonstration videos. With current Product Documentation creation procedure (with a Professional Product Document writer) being time consuming [6] with writers lacking key resources throughout the process [5] among many other problems.
2. **Student Tutor.** With ever-growing lengthy and detail oriented educational videos, learners need targeted assessments, summaries, study plans or a simple QnA interface to reinforce their understanding. Often a learner might need a subset of information but lack sufficient context to understand that subset. This often results in prolonged knowledge gathering phase before execution.

The overarching AI problem is twofold:

- *Information Extraction:* Seamlessly convert raw video and audio streams into a structured, comprehensive knowledge base.
- *Knowledge Utilization:* Generate accurate, context-aware textual outputs—product documents, executive summaries, assessments, and study plans—by querying the knowledge base through LLM-driven agents.

1.3 Six-Step Problem-Solving Approach

To justify and guide each design choice, we adopt a framework analogous to CRISP-DM:

1. **Framing the AI Problem.** We formalise our goals: (i) build a multimodal pipeline to extract and store video knowledge, and (ii) deploy LLM-based agents capable of producing targeted text outputs. Success metrics include the accuracy and completeness of the knowledge base and the quality of generated documents and chat responses.
2. **Data Gathering and Preparation.** We collect publicly available, non-confidential videos. Audio streams, video frames are separated and cleaned, deduplicated, and segmented into configurable chunks. This step ensures high-quality inputs for downstream LLM processing. This architecture also allows multiprocessing capabilities in production level systems; also making the solution LLM agnostic.
3. **Model Exploration and Selection.** Initially, we rely on prompt engineering with Google's Gemini LLM to obtain satisfactory transcripts and image captions. We record performance metrics

and leave open the possibility of fine-tuning or training custom models to further enhance accuracy. But, current prompts provide satisfactory responses limiting requirements to explore further.

4. **Solution Assembly and Evaluation.** We integrate the ingestion pipeline and agent suite, measuring key performance indicators at two stages: post-ingestion (transcript completeness and cohesiveness) and post-agent (textual fidelity and relevance). These metrics guide iterative refinements.
5. **Deployment.** The full architecture is containerized and deployed on the Google Cloud Platform. LangGraph orchestrates the data flow and agent interactions, ensuring scalability and fault tolerance.
6. **Monitoring and Retraining.** We schedule periodic evaluations of ingestion and agent metrics. Prompt templates are updated as needed, and future work includes fine-tuning models based on accumulated feedback.

This structured and iterative approach ensures that our system not only meets immediate business needs but also remains adaptable to evolving requirements and technological advances.

2 System Architecture

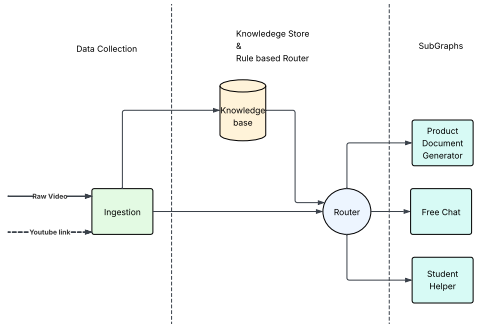

Figure 2. A High Level Architecture depicting different components.

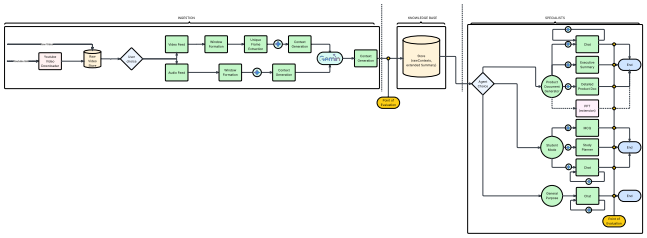

Figure 3. A Low Level Architecture depicting different subcomponents.

3 Methods and Evaluation Metrics

3.1 Data Acquisition and Preparation

The system accepts user-uploaded videos through a Streamlit-based frontend. To ensure ethical use and reproducibility, we have tested with publicly available content. Videos are processed one at a time, with intermediate data cached to avoid redundant computation and minimize LLM token usage. User can also provide their choice of window size which helps reduce llm token usage even further(depending on how dynamic the video content is)

3.2 Ingestion Pipeline

Each video is split into audio and visual streams. Using OpenCV and SceneDetect, the visual stream is windowed, and scene transitions are detected to extract distinct frames. In parallel, FFmpeg segments the audio stream to match the visual frame windows.

For each window, audio and image segments are passed through Google Gemini to generate transcripts and captions. A final video summary is synthesized using all collected content. All transcripts are stored locally and reused if the same video is reprocessed.

3.3 Knowledge Base Construction

Intermediate and final transcripts are stored as structured JSON files in an in-memory key-value store. This design enables rapid access and reuse, while future plans include migration to a traditional relational database for scalability.

3.4 Agent Design

Two specialized agents are implemented using LangChain and LangGraph, each responsible for distinct downstream tasks. Prompts are designed using few-shot strategies and include guardrails for structured output. Gemini generates the primary outputs, and Mistral serves as a judge, validating factual and structural consistency of the generated content.

3.5 Evaluation Protocol

We evaluate system performance using both reference-based and model-based metrics:

- **ROUGE** to assess surface-level similarity.
- **BERTScore** to evaluate semantic alignment.
- **LLM-as-a-Judge** acceptance rate, based on Mistral's binary evaluation of factual correctness and format.

Human evaluation is a planned future addition.

4 Results

Component	Precision	Recall	F1
Ingestion	91.5	91.5	91.5
Product Documentation	82.6	84.7	83.6
Executive Summary	86.8	87.4	87.1
Product Chat	87.8	89.3	88.5
MCQ Generator	84.7	86.6	85.7
Study Planner Agent	89.9	89.0	89.4
Student Chat	88.8	85.1	86.9
General Chat	86.4	86.0	86.2

Table 1. BERTScore (Precision, Recall, F1) for each component.

Component	Relevance (%)	Coherence (%)	Correctness (%)
Ingestion	80.0	80.0	78.0
Product Documentation	70.0	70.0	70.0
Executive Summary	70.0	50.0	70.0
Product Chat	69.0	69.0	65.0
MCQ Generator	85.0	90.0	70.0
Study Planner Agent	95.0	75.0	78.0
Student Chat	70.0	60.0	70.0
General Chat	85.0	60.0	75.0

Table 2. LLM-based human-like evaluation: Relevance, Coherence, and Correctness.

Component	ROUGE-1	ROUGE-2	ROUGE-L
Ingestion	49.5	26.7	48.1
Product Documentation	24.4	6.1	22.8
Executive Summary	37.0	12.5	37.0
Product Chat	30.9	10.6	28.7
MCQ Generator	38.8	22.5	37.6
Study Planner Agent	51.2	22.3	49.6
Student Chat	43.4	15.7	41.8
General Chat	42.0	15.7	40.3

Table 3. ROUGE metrics (F1 scores) for each component.

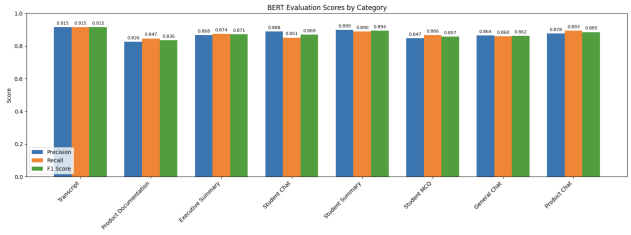

Figure 4. Bert Score Calculated for all Components.

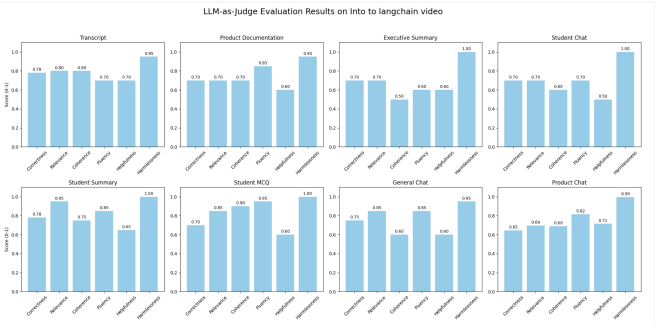

Figure 5. LLM as a judge score for all Components.

5 Demo and Code

Demo Video: Watch here
Live Application: Launch app
Code: Vid2Insight repo
Team's Whiteboard: Lucidchart board

6 Conclusion

This project demonstrates a modular and scalable architecture for extracting actionable insights from video content through a multimodal LLM-based pipeline. We addressed core challenges in knowledge extraction and usability by focusing on three key areas.

First, we implemented *token optimization* strategies by caching and reusing previously computed transcripts, thereby avoiding redundant LLM calls. This not only reduced computational costs but also improved latency and system responsiveness. Second, we built a *rich, structured knowledge base* that seamlessly integrates audio and visual context, enabling downstream agents to function in a plug-and-play fashion with minimal reconfiguration. Finally, we showcased how agents—such as the Product Documentation Generator—can support a *bidirectional, iterative interaction model*, simulating realistic back-and-forth conversations between developers and documentation authors.

Looking forward, the project offers several promising directions. We plan to move beyond prompt engineering by *fine-tuning specialised models* for audio-visual context understanding, particularly for technical domains. Additionally, *multi-video ingestion and parallelised pipelines* will help scale the system for enterprise use cases. Lastly, implementing a *versioned knowledge base* will allow the system to track content evolution, enabling features such as audit trails and historical document regeneration.

Overall, this work lays the foundation for a flexible, intelligent video understanding framework capable of adapting to diverse business and educational needs.

References

[1] G. DeepMind. Gemini api documentation, 2025. URL https://ai.google.dev/gemini-api/docs. Accessed: June 2025.

[2] R. Mansuy. Evaluating nlp models: A comprehensive guide to rouge, bleu, meteor, and bertscore metrics, 2023. URL https://ai.plainenglish.io/evaluating-the-performance-of-natural-language-processing-nlp-models-can-be-chall Accessed: June 2025.

[3] OpenAI. Chatgpt, 2025. Accessed via https://chat.openai.com, used DeepResearch to finetune our architecture and choose the best set of tools to build with, June 2025.

[4] OpenAI. Openai cookbook, 2025. URL https://cookbook.openai.com/examples/gpt_with_vision_for_video_understanding. Accessed: June 2025.

[5] C. Science. Technical documentation survey reveals top challenges and opportunities, 2022. URL https://review.content-science.com/technical-documentation-survey-reveals-top-challenges-and-opportunities/#:~:text=2. Accessed: June 2025.

[6] Smrita. How much time is spent on writing documentation versus developing rtl code?, 2023. URL http://sigasi.com/tech/how-much-time-spent-writing-documentation-versus-developing-rtl-code/#:~:text=measured%20in%20pages%2C%20and%20further,Minutes%2Fhours%20based%20method%20determines%20the. Accessed: June 2025.

[7] L. Team. Langchain documentation, 2025. URL https://docs.langchain.com/. Accessed: June 2025.

[8] L. Team. Langgraph documentation, 2025. URL https://docs.langchain.com/docs/langgraph/. Accessed: June 2025.

Project Contributors

Name	Contribution Area
Aditya Mukhopadhyay	Architecture, Student Subgraph, UI, Report
Ankit Kumar	Data collection, Ground Truth Creation, Evaluation
Danish Akhlaq	Architecture, Ingestion, Report
Sanjay Kumar	Graph Integration, Product Doc Subgraph, Integration Testing, Deployment
Trupti Rushikesh Kurambhatti	Ingestion, Testing, Prompt Engineering

Table 4. Team member contributions.

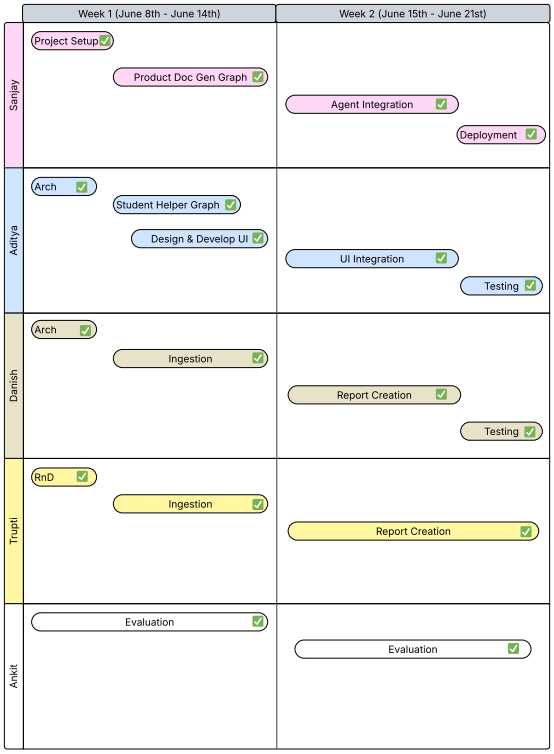

Figure 6. Week-by-week team work distribution.

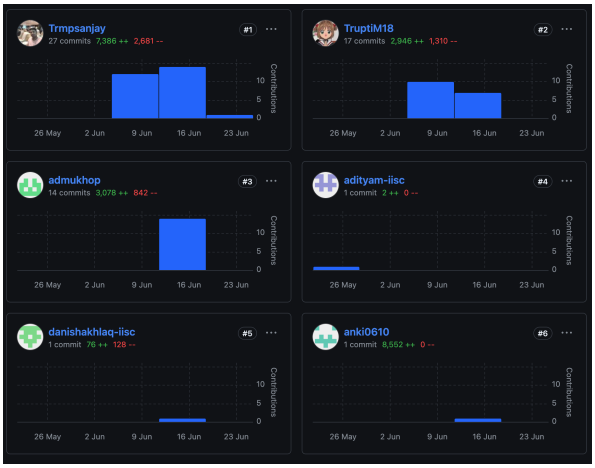

Figure 7. GitHub commit contributions over time. See Table 5 for GitHub username-to-name mapping.

GitHub Username	Contributor Name
adityam-iisc, admukhop	Aditya Mukhopadhyay
anki0610	Ankit Kumar
danishakhlaq-iisc	Danish Akhlaq
Trmpsanjay	Sanjay Kumar
TruptiM18	Trupti Rushikesh Kurambhatti

Table 5. Mapping of GitHub usernames to contributor names.

Appendix

A Iterative Design of Prompt Templates

Prompt engineering plays a pivotal role in ensuring reliable and task-specific responses from large language models (LLMs) throughout the Vid2Insight pipeline. Our system leverages four distinct prompt categories, each corresponding to a specific processing stage. This section outlines the progression, challenges, and justifications that shaped the current prompt strategies.

A.1 Prompt Type 1: Audio and Frame Transcription

The first category of prompts is employed during the ingestion phase, where audio and image frames are independently transcribed using Gemini LLM. The audio transcription prompt, largely adapted from Gemini's official documentation, proved satisfactory as it aligned with our goal of producing 1:1 verbatim conversions.

However, the frame-to-text prompt presented several technical challenges. While the prompt instructed the model to faithfully describe screen content, the responses often contained problematic tokens such as non-UTF-8 characters, improperly escaped code blocks, or premature end-of-text identifiers. These inconsistencies disrupted the downstream ingestion process, particularly when converting the LLM outputs to JSON format. Although the prompt included guardrails to avoid these characters, the LLM often prioritized literal transcription over structural safety.

To mitigate this, we incorporated traditional preprocessing filters to sanitize outputs post hoc, thus decoupling JSON parsing from LLM response reliability. In future iterations, we anticipate that a fine-tuned visual transcription model could resolve such structural inconsistencies natively.

A.2 Prompt Type 2: Unified Video Summary Generation

The second prompt stage is situated after the independent transcripts (audio and visual) have been generated. Here, the objective is to fuse windowed transcripts into a single cohesive video summary. As our system currently limits input video size to 200MB, the combined windowed context remains within typical LLM input limits, allowing for a single pass summarization.

Initial prompt versions failed to preserve the temporal structure of the original content, leading the model to arbitrarily reorder information. We addressed this by explicitly instructing the LLM to respect the chronological sequence of inputs and redesigned the payload to reflect this structure. This significantly improved the fidelity of generated summaries.

Our design also minimizes multi-pass LLM transformations, thereby reducing the risk of compounding hallucinations and contextual drift. The current summarization prompt achieves a balance between coherence and extractiveness, critical for downstream tasks.

A.3 Prompt Type 3: Specialist Agent Instructions

Specialist prompts are designed to generate structured content tailored to specific business or educational use cases—namely, product documentation, executive summaries, MCQ generation, and study planning.

For documentation and summaries, a single-shot prompting strategy is employed, where a sample output format is included to anchor model responses. The ingestion summary serves as the primary context. Over time, prompt refinements focused on increasing specificity, enhancing tone consistency, and reducing verbosity.

The MCQ generation prompt is more complex. It enforces a strict JSON schema to ensure compatibility with UI components. The output includes questions, options, correct answers, and topic coverage. This prompt underwent multiple iterations to balance generative freedom with structural constraints.

Future iterations will support richer user-defined input contexts—such as fixed document templates or learner profiles—to enable cold-start control and better user alignment.

A.4 Prompt Type 4: Interactive Chat Modules

The final prompt category powers our interactive chat modules. Each specialist chat agent (e.g., for study planning or product documentation) operates within a constrained domain and is initialized with relevant context artifacts—such as the latest executive summary or product document—alongside the user's message.

This setup enables a co-pilot experience, where users can modify content in the editor and issue related instructions in the chat. The chat prompt design supports such dynamic interactions while maintaining domain focus.

The MCQ feedback chat uses LLMs to critique responses and suggest learning improvements. Here, the context payload is programmatically generated, and prompts are structured to elicit actionable feedback rather than direct answers.

To support extended conversations, we employ LangChain's conversation memory module. It summarizes prior interactions to maintain context within input token limits while preserving coherence across sessions.

A.5 Prompt Type 5: LLM as a Judge

A critical part of our quality assurance pipeline involves the use of an "LLM as a judge" for each specialist response. This LLM receives two key inputs: (i) the response generated by a specialist agent, and (ii) the summarised context constructed during the ingestion phase.

The judging prompt is designed to assess responses across multiple criteria: factual accuracy, structural coherence, domain relevance, and compliance with expected output formats. The prompt is dynamic and adapts slightly based on the type of specialist being evaluated—for instance, the judgment of a product documentation differs subtly in emphasis compared to that of a student planner or MCQ generator.

One unique feature of this module is its capacity for recursive refinement. If the judge detects issues—such as topic drift, incorrect prioritization of content, or stylistic inconsistencies—it provides structured feedback and requests a revision from the corresponding specialist node. The system is thus able to iteratively self-correct through an internal feedback loop.

Over successive iterations, we observed specific challenges that necessitated adjustments to both the specialist prompts and the judge prompt. For example, early drafts of responses occasionally overemphasized trivial visual cues or ignored audio-based insights. These behaviors were addressed by reinforcing both specialist and judge's prompts.

Once the LLM judge approves a response, it is marked as final and proceeds to post-processing or user display. This mechanism not only enforces a baseline of output quality but also ensures modular accountability and scalable evaluation for multi-agent systems.

A.6 Conclusion

The evolution of our prompt design reflects a careful balance between automation, structure, and user guidance. Each prompt type has undergone targeted iterations to address context sensitivity, formatting robustness, and application-specific needs. Future work will incorporate user feedback loops, dynamic prompt tuning, and adaptive templates for improved personalization and reliability.

B Structured Payload Design and Evolution

A key design objective during system development was to define structured, extensible, and interpretable payloads that facilitate smooth transitions across pipeline stages. While several architectural decisions underwent refinement over time, our JSON-based payload formats were relatively stable and designed early in the process. This section documents the core data structures used for communication between ingestion, summarization, and specialist agents.

B.1 Audio Transcript Payload

The audio transcript payload was kept minimal, focusing on speaker tagging and utterance segmentation. The goal was to create a speaker-aware transcript stream that could be easily concatenated or merged into downstream contexts.

```
"audio_transcript": {
    "transcript": [
        {
            "speaker": "Speaker 1",
            "text": "..."
        }
    ]
}
```

B.2 Frame Transcript Payload

To capture the visual dimension of the video, frame-level transcription was conducted using image-to-text prompting. Each windowed segment stores references to frame files and associated textual interpretations. This will help us achieve the next iteration on presenting a product documentation with relevant screenshots.

```
"frame_transcript": {
    "titles": [
        "000/segment_0_frame_0.jpg"
    ],
    "details": [
        {
            "raw_text": "...",
            "explanation": "...",
            "transcript": "..."
        }
    ]
}
```

This structure accommodates both low-level OCR-style transcription and higher-level interpretation, allowing flexibility in rendering scene content.

B.3 Final Transcript Payload

All window-level audio and visual transcripts are combined into a unified knowledge context for further processing. The ingestion stage outputs a hierarchical payload where each segment contains detailed multimodal metadata, and a combined summary string is prepared for specialist modules.

```
{
  "video_id": "abcd",
  "combined_transcript": [
    {
      "combined_transcript": "..."
    }
  ],
  "segment_transcripts": {
    "000": {
      "audio_transcript": { ... },
      "frame_transcript": { ... }
    }
    // Additional segments omitted for brevity
  }
}
```

This JSON format enables targeted segment review during debugging, as well as streamlined access for downstream agents.

B.4 Specialist Agent Payloads

The combined transcript is passed to four specialist modules, each generating domain-specific outputs. The product documentation specialists (i.e, executive summary, product documentation) return outputs in Markdown format for structured rendering in the UI.

However, the MCQ generation module outputs a structured JSON payload to support interactive learning interfaces:

```
{
  "topics": ["Topic 1", "Topic 2", "Topic 3"],
  "questions": [
    {
      "question": "What is X?",
      "options": ["Option A", "Option B"],
      "correct_option": "Option B",
      "topics_covered": ["Topic 1", "Topic 2"]
    },
    ...
  ]
}
```

and for Student Planner, the output structure has been decided as such. This is to improve readability in the UI.

```
{
  "topics": ["Topic 1"],
  "summary": "...",
  "study_plan": [
    {
      "day": 1,
      "focus": "Intro to Topic 1",
      "activities": [
        "Read summary section",
        "Take notes",
```

```
        "Write a self-explanation"
      ]
    },
    {
      "day": 2,
      "focus": "...",
      "activities": [
        "Review notes",
        "Research examples",
        "Create a mind map"
      ]
    }
  ],
  "prerequisites": ["Concept A", "Term B"]
}
```

The payloads ensure content modularity and personalized progression for students, enabling back-end adaptability for user preferences.

B.5 LLM-as-a-Judge Feedback Payload

For every specialist response, a feedback mechanism is triggered using an LLM acting as a judge. This agent assesses factual correctness, structural completeness, and relevance, issuing corrective feedback or approval.

```
{
  "is_modification_required": true,
  "feedback": "..."
}
```

The judge has the authority to reject or approve the response. If modifications are requested, the specialist regenerates the content accordingly. This feedback loop enhances quality assurance and system self-correction.

B.6 Conclusion

The above payload structures reflect a deliberate focus on modularity, interpretability, and consistency across processing stages. By defining these schemas early in the design lifecycle, the system maintained robustness and extensibility, supporting varied downstream applications without structural conflicts.

C Scene-Detection & Segmentation Parameters

In designing the video ingestion pipeline, several approaches were explored and iteratively refined to balance fidelity, efficiency, and modularity. This section outlines the design evolution and technical rationale behind the current scene segmentation strategy.

C.1 Initial Approach: Direct Video-to-LLM Parsing

As a straightforward implementation, we first followed Gemini's documentation by directly passing the entire video stream into the LLM for transcription and captioning. While conceptually simple, this approach introduced two significant limitations:

- The parsing was limited to a fixed rate of 1 frame per second (fps), which proved to be suboptimal. For videos with little motion, this was too frequent, introducing redundant frames and inflating token usage. Conversely, for dynamic or fast-paced content, 1 fps was too sparse, missing key transitions or actions.
- The strategy tightly coupled the pipeline to Gemini's interface and assumptions. This reduced portability, as other LLMs might use different processing strategies or input constraints, thereby limiting the flexibility of our system.

Given these constraints, we moved away from this approach in favor of more granular and model-agnostic segmentation strategies.

C.2 Intermediate Approach: Fixed-Interval Frame Sampling

Next, we implemented a configurable sampling technique, where the video was split by selecting every n^{th} frame, with n settable by the user. While this offered improved control, it was still limited in key ways:

- The burden of choosing an appropriate n fell on the user, making the interface less intuitive and increasing the likelihood of poor configuration.
- The method remained naive in its handling of redundant content. Sampling every n^{th} frame does not account for the semantic similarity between adjacent frames, especially in largely static scenes.
- There was no guarantee that only informative or unique frames would be selected, meaning redundant or irrelevant data could still be sent to the LLM, inflating cost and latency.

C.3 Final Approach: Windowed Scene Detection with Frame Hashing

To address these limitations, our current pipeline adopts a hybrid technique:

- The video is first divided into temporal windows of configurable size.
- Within each window, we apply scene detection using the `scenedetect` Python library. This library compares frames using perceptual hashing, identifying significant visual changes and filtering out duplicates.
- Only distinct (non-redundant) frames from each window are passed to the LLM for captioning.

This method significantly reduces the number of redundant frames processed by the LLM, resulting in lower token usage and faster response times, particularly important in production environments with limited API quotas or tight latency constraints.

C.4 Future Work: Task-Specific Frame Relevance Modeling

Looking forward, we envision incorporating a fine-tuned model that learns to identify and select "important" frames based on the downstream task context:

- For example, in product documentation use cases, the model could prioritize frames where a tool or function is demonstrated.
- In educational videos, it might favor slides, diagrams, or on-screen writing.
- For interview analysis or human behavior modeling, it could emphasize frames showing facial expressions, gestures, or scene transitions.

Such task-aware frame selection would improve both efficiency and contextual relevance, enabling more accurate and tailored outputs for different business applications.

D Evaluation Methodology and Metrics

This appendix details the design, methodology, and interpretation of the evaluation process employed in our system. Given the modular nature of the architecture and its reliance on LLM-generated content, evaluation plays a critical role in ensuring output quality and guiding iterative refinements.

D.1 Evaluation Design and Execution Points

The evaluation pipeline was designed to assess the system at two critical junctures:

1. **Post-Ingestion Stage:** Evaluation of the audio and frame transcripts along with the final summary generated via LLM.
2. **Post-Specialist Stage:** Evaluation of the textual outputs from each agent module, including product documentation, executive summaries, student planners, and MCQ generators.

Although ideally evaluation could be applied after every LLM call, we strategically chose only two checkpoints to reduce token usage and request overhead. Given that most prompts were deterministic with guardrails and minimal chain-of-thought reasoning, the marginal gain from per-response evaluation was deemed unnecessary at this stage.

A dedicated flow was constructed to compute evaluation metrics separately and periodically. This strategy also allows future integration of human-in-the-loop feedback and drift tracking across temporal versions of the same video.

D.2 Ground Truth Curation Strategy

A significant component of the evaluation process was the manual curation of high-quality ground truth for each module:

- **Audio Transcript:** Derived from publicly available YouTube captions to maintain consistency with actual spoken content.
- **Frame Transcript:** Manually written based on selected frames, describing visual content in a structured format.
- **Combined Transcript:** Synthetically constructed from both the above inputs to act as a reference for overall semantic fidelity.
- **Specialist Outputs:** Hand-authored markdown documents for the product documentation and executive summary, as well as structured JSON payloads for student planners and MCQs.
- **Chat Responses:** Crafted reply sets to standard queries for each chat module, enabling targeted comparisons.

This multi-tier ground truth setup ensured robust reference points across modalities and downstream tasks.

D.3 Metrics and Interpretation

We employed three distinct classes of evaluation metrics to triangulate the system's performance:

D.3.1 ROUGE Scores

- **ROUGE-1, ROUGE-2, and ROUGE-L** F1 scores were calculated to measure n-gram and sentence-level overlap with the ground truth. Given the generative nature and multimodal input diversity, these scores were expectedly modest, reflecting lexical variation across responses.

D.3.2 BERTScore

- **Precision, Recall, and F1** components of the BERTScore were used to measure semantic similarity using contextual embeddings. These values remained consistently high, indicating strong content-level alignment between system outputs and human-authored references.

D.3.3 LLM-as-a-Judge Evaluation

- We used Mistral to evaluate outputs across six qualitative dimensions: *Correctness, Relevance, Coherence, Fluency, Helpfulness, and Harmlessness*.
- This metric acted as a soft surrogate for human judgment and was especially useful for assessing structural and stylistic quality.

D.4 Results and Insights

The ROUGE scores across components were moderate, which aligns with expectations for a system with layered generative stages, particularly in multimodal contexts where visual descriptions can vary widely. Conversely, high BERTScore values affirmed the semantic robustness of the system's outputs.

LLM-as-a-Judge scores were slightly below par on average, reflecting minor inconsistencies in tone, coverage, or structural conventions. These issues were within acceptable bounds and were largely mitigated by iterative prompt refinement and human review.

D.5 Future Evaluation Directions

We plan to introduce fine-tuned models optimized for task-specific video contexts. This evolution will necessitate richer evaluation strategies, potentially including:

- **BLEU and METEOR** scores for stylistic and lexical fluency.
- **Context-sensitive feedback** loops integrated with user behavior tracking.
- **Online evaluation strategies** where live user feedback directly informs prompt tuning or model adjustment.

This roadmap ensures the system remains responsive to evolving application requirements while maintaining rigorous evaluation standards.