

**INTEL UNNATI INDUSTRIAL TRAINING 2025**

**PROBLEM STATEMENT: AI-POWERED INTERACTIVE  
LEARNING ASSISTANT FOR CLASSROOMS**

**Team Name: Logic loops**

## ABSTRACT

In today's digital learning era, students increasingly rely on video-based content such as YouTube lectures to understand academic concepts. However, the sheer length and density of such videos often lead to information overload, difficulty in note-taking, and lack of effective revision tools. To address these challenges, **Gistify** is developed as an AI-powered web application that transforms any educational YouTube video into a concise, structured, and multilingual learning resource.

Gistify performs **automated speech transcription, AI-driven summarization, multi-language translation, MCQ generation, notes and glossary extraction**, and offers an **AI chatbot** for real-time doubt-solving. The application integrates advanced natural language processing and machine learning models such as **Whisper** for transcription, **T5 with OpenVINO optimization** for efficient summarization, **Deep Translator** for regional language support, and **LangChain with Groq**, which leverages **Large Language Models (LLMs)** to generate context-aware MCQs, notes, and glossaries.

Designed with a user-friendly Flask-based interface, Gistify allows students to interactively extract insights from videos, translate summaries into four Indian languages (Hindi, Tamil, Telugu, Malayalam), and download personalized notes and quizzes in PDF format. An AI chatbot, powered by an OpenVINO-accelerated **LLM** enables students to ask questions based on the video content and receive instant, accurate responses.

By bridging the gap between passive video consumption and active learning, Gistify redefines how students interact with online lectures and empowers them with tools to learn smarter, faster, and more effectively.

## INTRODUCTION:

Today, students rely on YouTube and other video platforms more than ever to learn new topics, catch up on missed lessons, or prepare for exams. While the content is easily accessible, actually learning from long, information-heavy videos can be time-consuming and overwhelming. Students often find it hard to take proper notes, stay focused through lengthy lectures, or quickly revise key points before an exam.

In many cases, these videos are not available in the student's native language, making it harder to understand or retain the content. Even if a student watches a video, they might still have doubts or wish they had a quick summary, important definitions, or practice questions based on the lecture.

That's where **Gistify** comes in. It was built to make this whole process easier and smarter. Instead of just watching and re-watching videos, Gistify helps students turn any YouTube lecture into a more useful learning format. It uses modern AI tools to break down the content into clear summaries, support different Indian languages, and even help students test their understanding with quizzes and instant doubt-solving.

This project is the result of combining different AI technologies—from speech recognition and summarization to large language models (LLMs)—and making them work together in a simple, usable web app that runs completely offline. The rest of this report explains how Gistify was built, what tools were used, and how it can change the way students learn from videos.

This report walks through the reasons behind building Gistify, the technologies involved, and how it was implemented to support smarter, faster, and more accessible video-based learning.

## PROBLEM STATEMENT:

While online platforms like YouTube have become a popular source of educational content, students face several challenges when trying to learn effectively from such videos. Most educational videos are long and unstructured, making it difficult for students to quickly grasp the key points. Rewatching or manually taking notes can be time-consuming and mentally exhausting, especially during exam preparation or when revising large topics.

Another major issue is the lack of accessibility. Many students come from different linguistic backgrounds and struggle to understand content that's not available in their native language. This creates a language barrier that limits learning, especially in a country as diverse as India.

Additionally, students often have doubts while watching videos but no immediate way to get those questions answered. There's no built-in support for interaction, revision, or practice through features like MCQs or glossaries. This passive form of learning reduces retention and slows down the overall understanding of the topic.



## PROPOSED SOLUTION:

To address the challenges students face with online educational videos, this project proposes a smart, AI-powered web application called **\*\*Gistify\*\***. The solution is designed to make video-based learning more efficient, accessible, and interactive by converting passive lecture content into personalized study material.

Gistify uses a combination of speech recognition, natural language processing, and large language models (LLMs) to process YouTube videos and deliver structured outputs. The system is built to work entirely offline using optimized models, ensuring privacy, low latency, and availability across different environments.

The solution offers the following core features:

**Automatic Transcription:** Converts video audio into clean text using Whisper, allowing the system to understand and process spoken content accurately.

**AI-Based Summarization:** Uses a T5 model optimized with OpenVINO to generate concise summaries from transcripts, saving students time and helping them focus on key concepts.

**Multilingual Translation:** Uses Deep Translator to convert summaries into five Indian languages (Hindi, Tamil, Telugu, Malayalam, and Kannada), enhancing accessibility for regional learners.

**Interactive Content Generation:** Leverages LangChain with Groq-backed LLMs to create context-aware multiple-choice questions (MCQs), brief notes, and glossaries based on the summary.

**AI Chatbot for Doubt-Solving:** Integrates a local chatbot powered by DistilBERT (optimized with OpenVINO) that answers questions using the summary as context, allowing students to clear doubts instantly.

**PDF Export:** Allows students to download summaries, notes, and glossaries as PDFs for offline use.

**User-Friendly Interface:** Built using Flask, the interface is clean, responsive, and simple to use, even for non-technical users.

This solution transforms raw video content into structured educational material, bridging the gap between passive viewing and active learning while supporting a wide range of users across linguistic and technological boundaries.

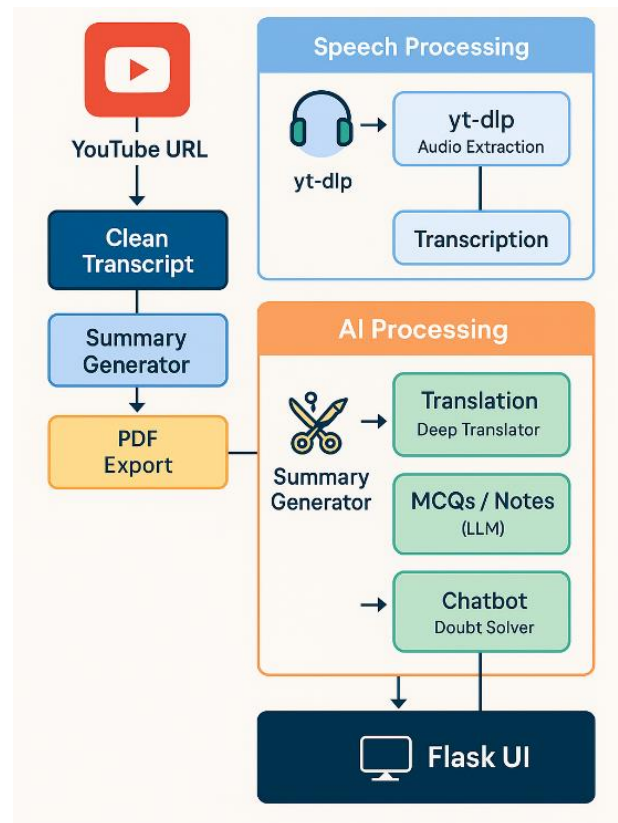
## **SYSTEM ARCHITECTURE:**

The architecture of Gistify follows a modular, pipeline-based design that processes educational YouTube videos into structured, multilingual, and interactive learning content. Each stage in the system plays a specific role—from extracting and transcribing video audio to generating summaries, translations, MCQs, and more. The entire pipeline is built using open-source and optimized tools, ensuring fast, offline, and efficient processing.

The architecture is divided into three key modules:

1. **Speech Processing** – Handles YouTube audio extraction using ``yt-dlp`` and transcribes it using the Whisper model.
2. **AI Processing**– Generates summaries using OpenVINO-optimized T5, translates them into regional languages, and creates MCQs, notes, and chatbot answers using LLMs.
3. **User Interface** – A Flask-based web interface that connects all modules and presents the output in a simple, user-friendly format with options for PDF download and interactive querying.

This layered design ensures scalability, flexibility, and ease of use—making Gistify a powerful educational tool for students of all backgrounds.



## TECHNOLOGIES USED:

### 1. Python:

- Acts as the backbone of the entire application, powering all AI and backend logic.
- Provides strong support for NLP, web development, and model integration using libraries like Hugging Face, Flask, etc.



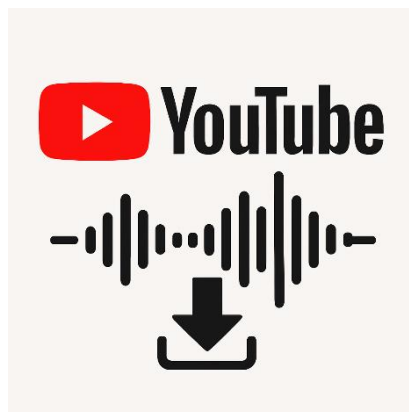
### 2. Flask:

- Used to develop the web interface where users input YouTube links and interact with outputs.
- Handles routes for summarization, translation, chatbot, and PDF generation efficiently.



### 3.YT-DLP:

- Extracts only the audio from a YouTube video using the provided link.
- Allows faster processing and smaller file size by avoiding full video download.



### 4.WHISPER:

- Converts extracted audio into accurate English transcripts using speech recognition.
- Works well even with background noise or unclear pronunciation.



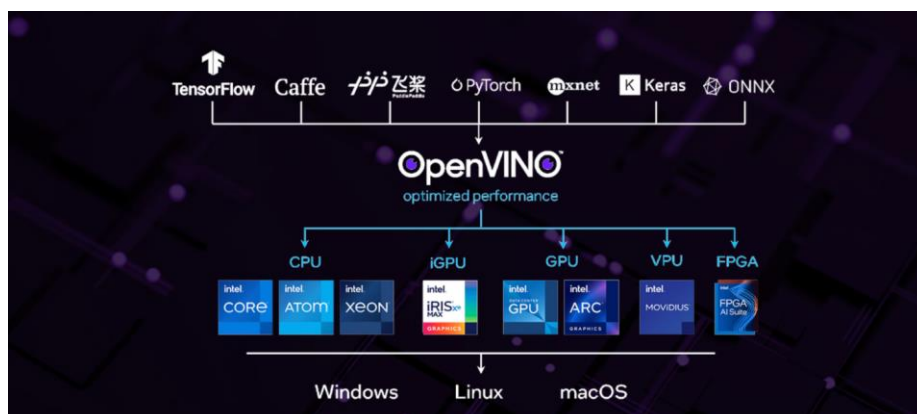
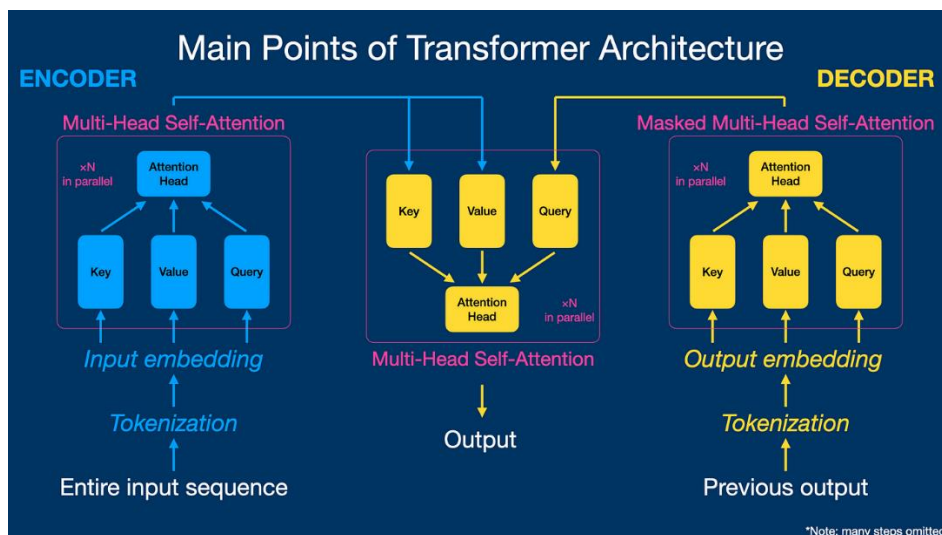
### 5.TEXT PRE-PROCESSING:

- Cleans transcript by removing repetitive phrases, filler words, and grammar issues.
- Ensures higher-quality inputs for summarization and translation.



## 6. T5 + OPENVINO:

- Generates concise summaries from transcripts using T5 transformer model.
- Optimized using Intel's OpenVINO to run faster on regular CPUs without GPU dependency.





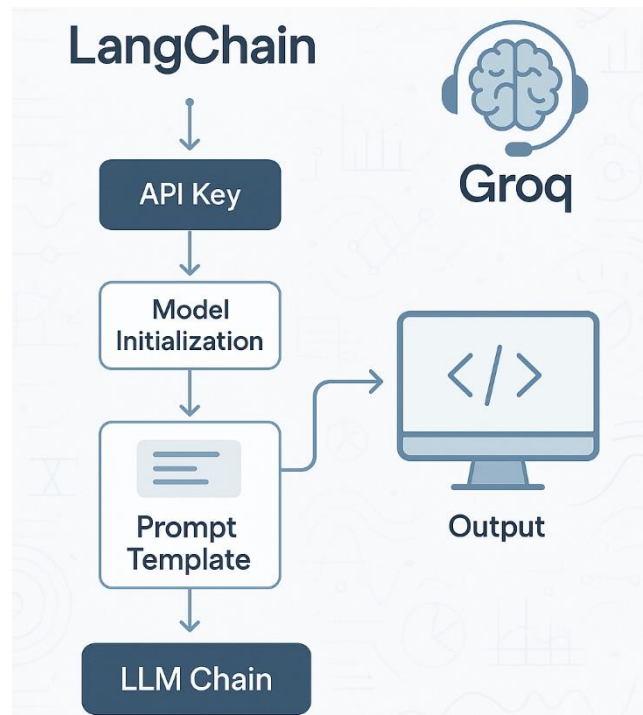
## 7. DEEP TRANSLATOR:

- Translates English summaries into Hindi, Tamil, Telugu, Malayalam, and Kannada.
- Simple, lightweight, and works efficiently with long text chunks.



## 8. LANGCHAIN + GROQ + LLM:

- Powers the generation of MCQs, notes, and glossary using large language models.
- Chatbot answers user questions in natural language using summary as context.



## 9. XHTML2PDF:

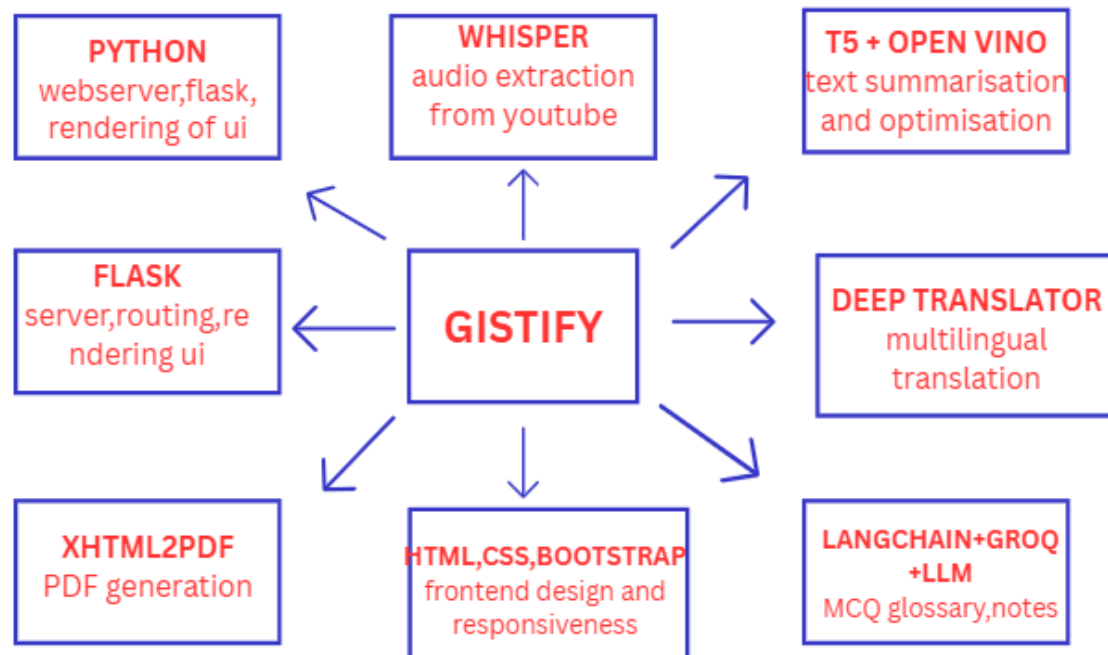
- Converts structured HTML content into downloadable PDF files.
- Gives students a way to study offline with summarized notes and MCQs.



## 10. HTML / CSS / BOOTSTRAP:

- Builds the front-end layout with responsiveness and consistent design.
- Ensures smooth user experience across devices like laptops and mobiles.





## SYSTEM WORKFLOW:

The Gistify application follows a modular and sequential workflow that transforms a YouTube video into a multilingual, structured study resource. Each component in the pipeline performs a specific task, with the output of one module serving as the input for the next.

The complete workflow is as follows:

### 1. YouTube URL Input

The user provides a YouTube video link through the web interface.

### 2. Audio Extraction (yt-dlp):

The backend uses `yt-dlp` to extract only the audio stream from the video, avoiding unnecessary downloads.

### 3. Speech-to-Text Transcription (Whisper):

The extracted audio is transcribed into clean English text using OpenAI's Whisper model.

### 4. Transcript Cleaning:

The raw transcript is processed to remove repetitive phrases, filler words, and formatting issues.

## 5. Summarization (T5 + OpenVINO):

The cleaned transcript is passed through a T5 model optimized with OpenVINO, producing a concise summary.

## 6. Translation (Deep Translator):

The summary is translated into five Indian languages: Tamil, Hindi, Telugu, Kannada, and Malayalam.

## 7. AI-Powered Content Generation (LLM via LangChain + Groq):

Multiple-choice questions (MCQs), key notes, and glossaries are generated using a large language model accessed through LangChain and Groq.

## 8. Chatbot (LLM-based):

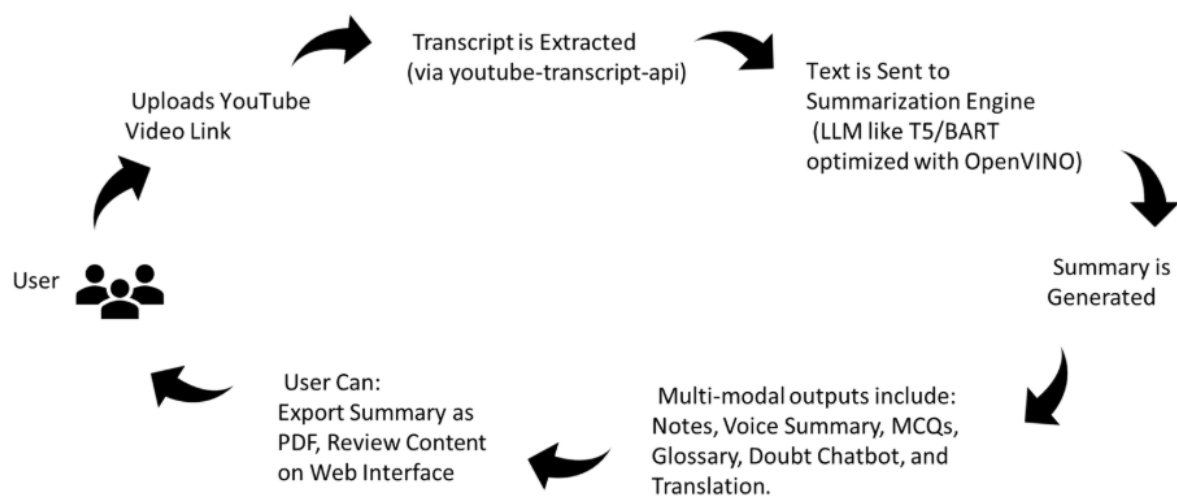
An AI chatbot is integrated into the UI, allowing students to ask questions and receive answers based on the summary context.

## 9. PDF Export (xhtml2pdf):

Summaries, notes, and glossary entries can be exported as downloadable PDF documents.

## 10. User Interface (Flask + Bootstrap):

All features are integrated and made accessible through a user-friendly, responsive web interface.



## IMPLEMENTATION:

### BACKEND:

#### 1.YOUTUBE AUDIO EXTRACTION & TRANSCRIPTION

##### Technology Used:

\* `yt\_dlp` to download audio

\* `Whisper` (base model) for multilingual transcription and English translation

##### Code Snippet:

```
def download_audio(youtube_url):  
    uid = uuid.uuid4().hex  
    output_template = f"audio_{uid}.{%(ext)s}"  
    ydl_opts = {  
        'format': 'bestaudio/best',  
        'outtmpl': output_template,  
        'postprocessors': [{ 'key': 'FFmpegExtractAudio', 'preferredcodec': 'mp3'}],  
    }  
    with yt_dlp.YoutubeDL(ydl_opts) as ydl:  
        info = ydl.extract_info(youtube_url, download=True)  
        return f"{ydl.prepare_filename(info).rsplit('.', 1)[0]}.mp3"  
  
def transcribe_audio(file_path):  
    return whisper_model.transcribe(file_path)["text"]
```

#### 2.TRANSCRIPT PRE-CLEANING

To improve summary quality, the transcript is cleaned using two functions:

\* `pre\_clean\_transcript()` removes repeated lines, words, and filler

\* `clean\_summary()` fixes punctuation and ensures sentence clarity

**Code Snippet:**

```
def pre_clean_transcript(text):  
    text = re.sub(r'\b(\w+)( \1\b)+', r'\1', text, flags=re.IGNORECASE)  
    # Removes repeated words like "got got got"  
    ...  
    return cleaned_text  
...
```

**3.SUMMARIZATION USING OPENVINO-OPTIMIZED T5**

**Model Used:** `t5-small`

**Optimized With:** `OVModelForSeq2SeqLM` from Intel's OpenVINO Toolkit

**Purpose:** Generate compressed summaries efficiently on CPU

**Code Snippet:**

```
tokenizer = AutoTokenizer.from_pretrained("t5-small")  
  
summary_model = OVModelForSeq2SeqLM.from_pretrained("t5-small",  
export=True)  
  
def summarize_text(text):  
    chunks = chunk_text(text)  
    final_summary = ""  
    for chunk in chunks:  
        inputs = tokenizer(chunk, return_tensors="pt", truncation=True)  
        outputs = summary_model.generate(**inputs, max_length=150, min_length=40)  
        summary_text = tokenizer.decode(outputs[0], skip_special_tokens=True)  
        final_summary += summary_text + " "  
    return final_summary.strip()
```

**4.SUMMARY TRANSLATION**

**Library Used:** `deep\_translator`

**Supported Languages:** Hindi, Tamil, Telugu, Malayalam

**Workflow:**

- \* Splits the summary into manageable chunks
- \* Translates each using GoogleTranslator
- \* Rejoins translated parts for display

**Code Snippet:**

```
def translate_text(full_text, target_lang):  
  
    chunks = [full_text[i:i+5000] for i in range(0, len(full_text), 5000)]  
  
    return ' '.join([GoogleTranslator(source='auto', target=target_lang).translate(chunk)  
for chunk in chunks])
```

## 5.MCQ, NOTES, GLOSSARY GENERATION

**LLM Used:** `LangChain` + `Groq` (LLaMA3 8B)

**Prompt Templates:**

`mcq\_prompt`, `notes\_prompt`, and `glossary\_prompt` guide the LLM

Outputs are structured, educational content generated from the summary

**Example Snippet (MCQ Generation):**

```
mcq_prompt = PromptTemplate(...template="Generate {num_questions} MCQs...")  
  
mcq_chain = LLMChain(llm=llm, prompt=mcq_prompt)  
  
@app.route("/generate-mcq")  
  
def generate_mcq():  
  
    raw_mcqs = mcq_chain.run({"context": summary, "num_questions": 5}).strip()  
  
    session["mcqs"] = parse_mcqs(raw_mcqs)
```

## 6.CHATBOT (DOUBT-SOLVING)

**Model Used:** `LangChain` + `Groq` (LLaMA3 8B)

**Purpose:** Answer user questions based on summary

### Workflow:

- \* Accepts a question via JavaScript fetch
- \* Uses the summary as context for answering

### Code Snippet:

```
@app.route("/chat", methods=["POST"])

def chat():

    question = request.json.get("question")

    summary = session.get("summary", "")

    prompt = f"Summary: {summary}\n\nQuestion: {question}\nAnswer:"

    reply = llm.invoke(prompt)

    return jsonify({"answer": reply.content.strip()})
```

## 7. PDF DOWNLOAD FUNCTIONALITY

**Library Used:** `xhtml2pdf (pisa)`

**Purpose:** Users can download the summary, notes, and glossary as a single PDF or MCQs separately.

### Code Snippet:

```
@app.route("/download-summary-notes-glossary")

def download_summary_notes_glossary():

    html = render_template_string(..., summary=summary, notes=notes,
    glossary=glossary)

    pisa.CreatePDF(io.StringIO(html), dest=stream)

    return send_file(stream, download_name="summary_notes_glossary.pdf",
    as_attachment=True)
```

## 8. SESSION MANAGEMENT:

All user interactions (summary, selected language, AI responses) are stored in Flask `session` objects to maintain continuity across routes.





## FRONTEND:

The frontend is implemented using HTML, CSS, and JavaScript, and connects seamlessly with Flask backend routes. The UI is clean, interactive, and highly functional for educational purposes.

### User Interface Overview

The interface allows users to:

- \* Paste a YouTube URL and click \*\* Summarize\*\*
- \* View the AI-generated summary
- \* Download summary + notes + glossary
- \* Listen to the summary with voice control
- \* Translate into regional languages
- \* Generate MCQs, Notes, and Glossary
- \* Practice MCQs and view scores
- \* Chat with the AI assistant using the embedded chatbot
- \* Get usage help via a \*\* Help\*\* popup

### 1.YOUTUBE INPUT & SUMMARIZE BUTTON:

```
<form method="POST" action="/">
```

```
  <input type="text" name="youtube_url" placeholder="Enter YouTube URL "
  required>
```

```
  <button type="submit"> Summarize</button>
```

```
</form>
```

On clicking Summarize, the page sends a POST request to the Flask server, which handles transcription and summarization.

### 2.DISPLAY OF AI-GENERATED SUMMARY:

```
<div class="summary-content" id="summaryText">{{ summary }}</div>
```

```
<a href="/download-summary-notes-glossary"> Download PDF</a>
```

The summary is dynamically filled using Jinja templating (`{{ summary }}`) and can be downloaded as PDF.

### 3.MANUAL TRANSLATION UI:

```
<form method="POST" action="/translate">

  <select name="language">

    <option value="tamil">Tamil</option>

    <option value="hindi">Hindi</option>

    ...

  </select>

  <button type="submit">🌐 Translate</button>

</form>
```

Translation is triggered only after the summary is generated. It sends the selected language via POST to the `/translate` route.

### 4.AI FEATURE BUTTONS:

```
<form action="/generate-mcq" method="get"><button>📝 Generate
MCQs</button></form>

<form action="/generate-notes" method="get"><button>📖 Generate
Notes</button></form>

<form action="/generate-glossary" method="get"><button>📖 Generate
Glossary</button></form>
```

These buttons call specific backend routes to activate LangChain-based content generation.

### 5.TEXT-TO-SPEECH INTEGRATION:

JavaScript functions allow reading the summary aloud:

```
function readSummary() {

  const text = document.getElementById("summaryText").textContent;

  utterance = new SpeechSynthesisUtterance(text);

  speechSynthesis.cancel();
```

```
speechSynthesis.speak(utterance);  
}
```

Additional buttons support Pause, Resume, and Stop actions.

## 6.INTERACTIVE MCQ QUIZ:

Rendered using a loop over the `mcqs` list:

```
{% for q in mcqs %}  
  
<div class="mcq-question">Q{{ loop.index }}: {{ q.question }}</div>  
  
{% for option in q.options %}  
  
  <input type="radio" name="q{{ loop.index }}" value="{{ option }}">  
  
  {{ option }}  
  
{% endfor %}  
  
{% endfor %}  
...
```

JavaScript calculates the score and shows answers immediately after submission:

```
document.getElementById("quizForm").addEventListener("submit", function(e) {  
  
  // scoring logic  
  
});
```

## 7.DOUBT-SOLVING CHATBOT:

The chatbot uses a floating UI with a text input and send button.

```
<form id="chatForm">  
  
  <input type="text" id="userQuestion" placeholder="Ask a question...">  
  
  <button type="submit">➤</button>  
  
</form>
```

JavaScript handles chat requests:

```
fetch("/chat", {
```

```

    method: "POST",

    headers: { "Content-Type": "application/json" },

    body: JSON.stringify({ question })

  })

  .then(res => res.json())

  .then(data => appendMessage("bot", data.answer));

```

## 8.HELP POPUP UI:

Provides first-time user guidance with toggle animation:

```

<div class="help-icon" onclick="toggleHelpPopup()"> ? </div>

<div id="helpPopup">

  <h3>How to Use Gistify</h3>

  <ul>

    <li>📌 Paste a YouTube link</li>

  </ul>

</div>

```

## 9.JS TOGGLES DISPLAY:

```

function toggleHelpPopup() {

  const popup = document.getElementById("helpPopup");

  popup.style.display = popup.style.display === "block" ? "none" : "block";

}

```

## 10.FOOTER & AESTHETIC DESIGN:

Styled with Montserrat fonts, icons, and custom CSS, the design offers:

- \* Capsule-style summary cards
- \* Interactive hover buttons
- \* Section separation using visual headers

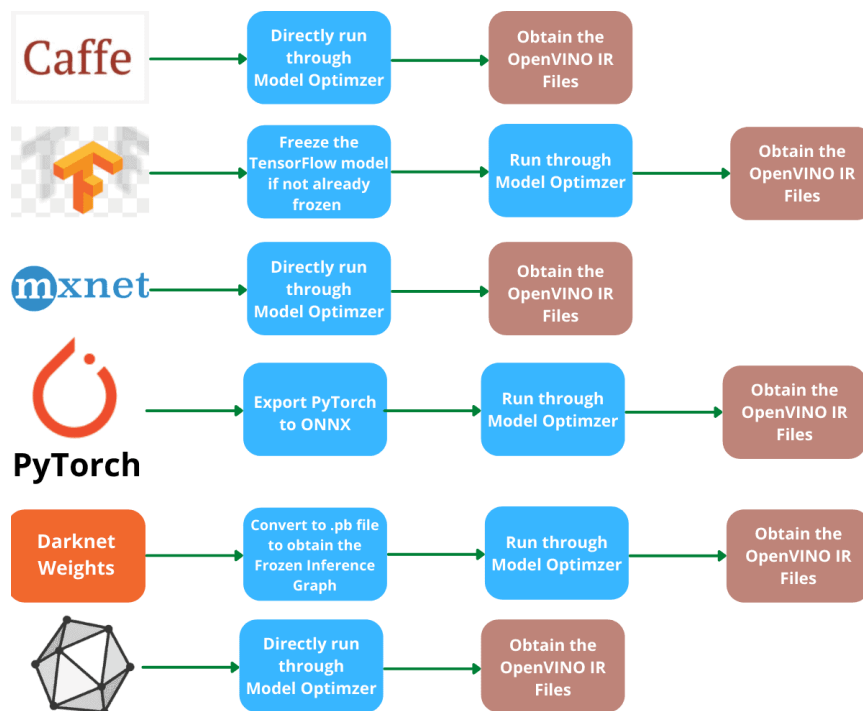
\* Friendly emoji-based labeling

## OPENVINO TOOLKIT INTEGRATION IN GISTIFY:

### OpenVINO:

OpenVINO™ (Open Visual Inference and Neural Network Optimization) is a powerful toolkit developed by Intel to optimize and accelerate the performance of deep learning models, particularly for **inference on CPU** and edge devices.

It allows you to take a pre-trained model (like from HuggingFace) and convert it into a format that runs faster, with lower memory usage, and without needing a GPU.



### USAGE OF OPENVINO IN GISTIFY:

In Gistify, OpenVINO is used to optimize the T5 summarization model during the step where the cleaned transcript is converted into a short summary.

### Code Location:

```
from optimum.intel.openvino import OVModelForSeq2SeqLM

summary_model = OVModelForSeq2SeqLM.from_pretrained("t5-small",
export=True)
```

### **Inference Usage:**

```
inputs = tokenizer(chunk, return_tensors="pt", truncation=True)
```

### **Why OpenVINO Was Integrated in Gistify?**

The integration of the **\*\*OpenVINO Toolkit\*\*** in Gistify plays a crucial role in enhancing the performance, efficiency, and user experience of the summarization module. Below are the key reasons for using OpenVINO along with the benefits it brings to the application:

#### **Faster Inference:**

By optimizing the T5 summarization model, OpenVINO significantly reduces the time required to generate summaries, ensuring a smooth and responsive experience for users.

#### **CPU Optimization:**

Unlike traditional deep learning models that rely heavily on GPUs, the OpenVINO-optimized model performs inference efficiently on regular CPUs. This makes the app accessible and usable on standard laptops and desktops.

#### **Fully Local Deployment:**

All summarization happens locally on the user's machine without depending on external APIs or cloud services. This not only reduces latency but also ensures better data privacy and offline capability.

#### **Efficient Handling of Large Transcripts:**

Educational videos can be long and dense. OpenVINO ensures that the summarizer can handle large transcript chunks without performance drops or memory issues.

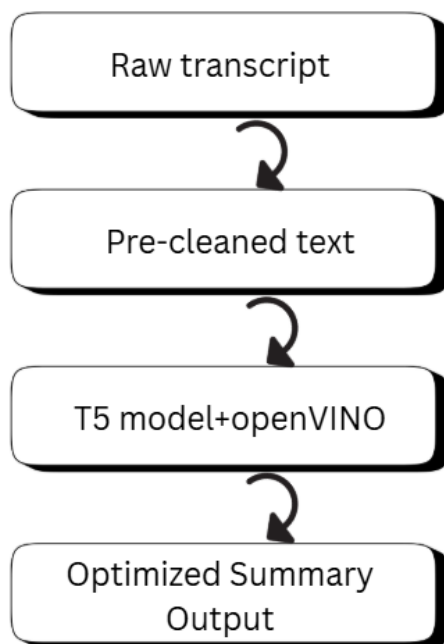
#### **Scalable for Multi-User Environments:**

The lightweight and fast nature of the OpenVINO-optimized model allows Gistify to scale better, supporting simultaneous users without overloading system resources.

PERFORMANCE METRICS COMPARISON:

| Metric                       | pytorch                         | ONNX runtime | OpenVINO toolkit  |
|------------------------------|---------------------------------|--------------|-------------------|
| Inference time               | 10-12 seconds                   | 6-8 seconds  | 3-4 seconds       |
| GPU required                 | Recommended                     | optional     | Not required      |
| CPU performance              | Slower                          | Medium       | Optimized         |
| RAM usage (for T5)           | ~2GB                            | ~1.4GB       | ~500-800 MB       |
| Supports Local Deployment    | Partial(needs weights download) | Yes          | Yes               |
| Integration Effort           | Medium                          | Medium       | Easy with optimum |
| Edge Device Ready            | No                              | Limited      | Yes               |
| Overall speed and efficiency | Moderate                        | Better       | Best              |

## SUMMARIZATION PIPELINE USING T5 + OPENVINO:



## EVALUATION AND RESULTS:

The performance of Gistify was evaluated based on inference speed, resource utilization, and the quality of output across features such as summarization, MCQ generation, translation, and chatbot responsiveness. The system was tested on a standard CPU-based laptop, ensuring accessibility for everyday student devices.

## PERFORMANCE BENCHMARKS:

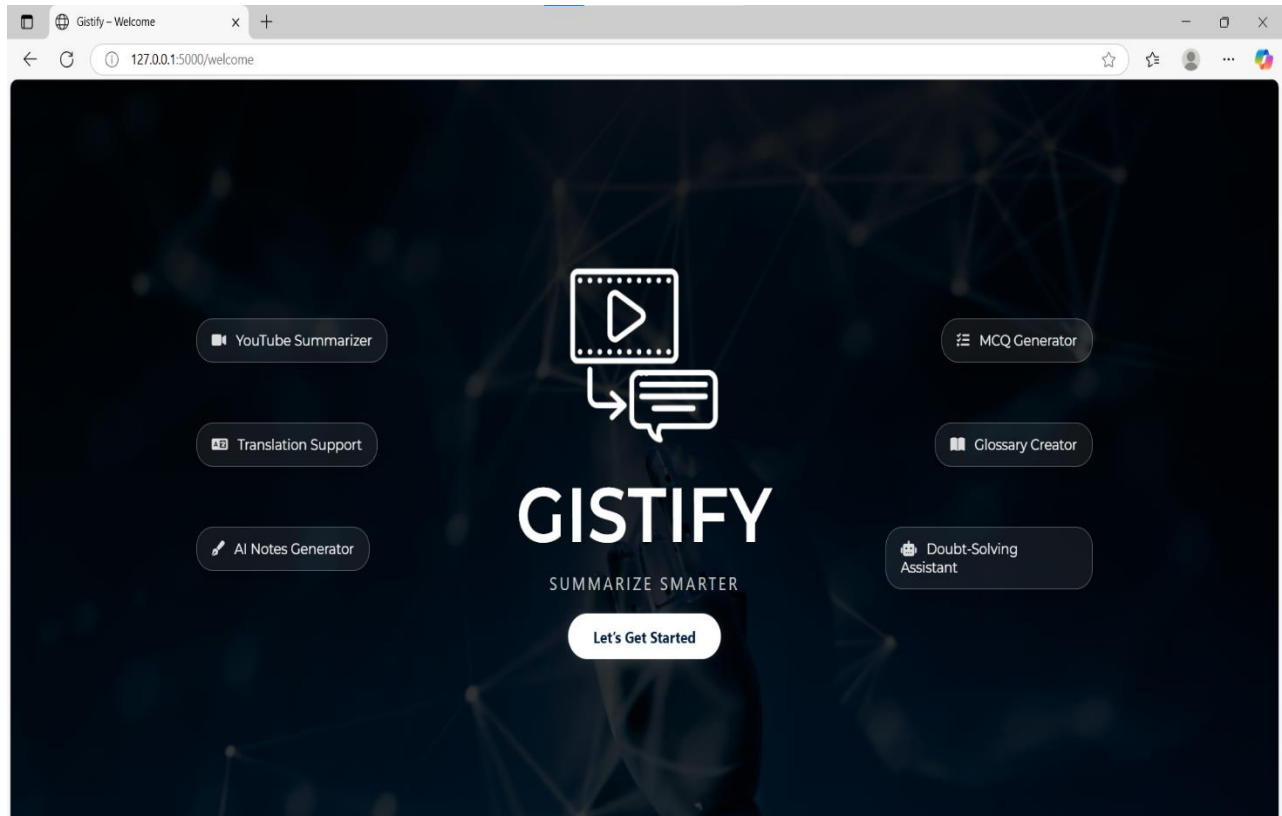
| Metric           | PyTorch Model    | OpenVINO Optimization |
|------------------|------------------|-----------------------|
| Inference Time   | 10-12 seconds    | 3-4 seconds           |
| GPU dependency   | Yes(Recommended) | No(CPU-only)          |
| RAM usage        | ~2 GB            | ~500-800 MB           |
| Summary Accuracy | Moderate         | High(clean+concise)   |



## OUTPUT SCREENSHOTS:

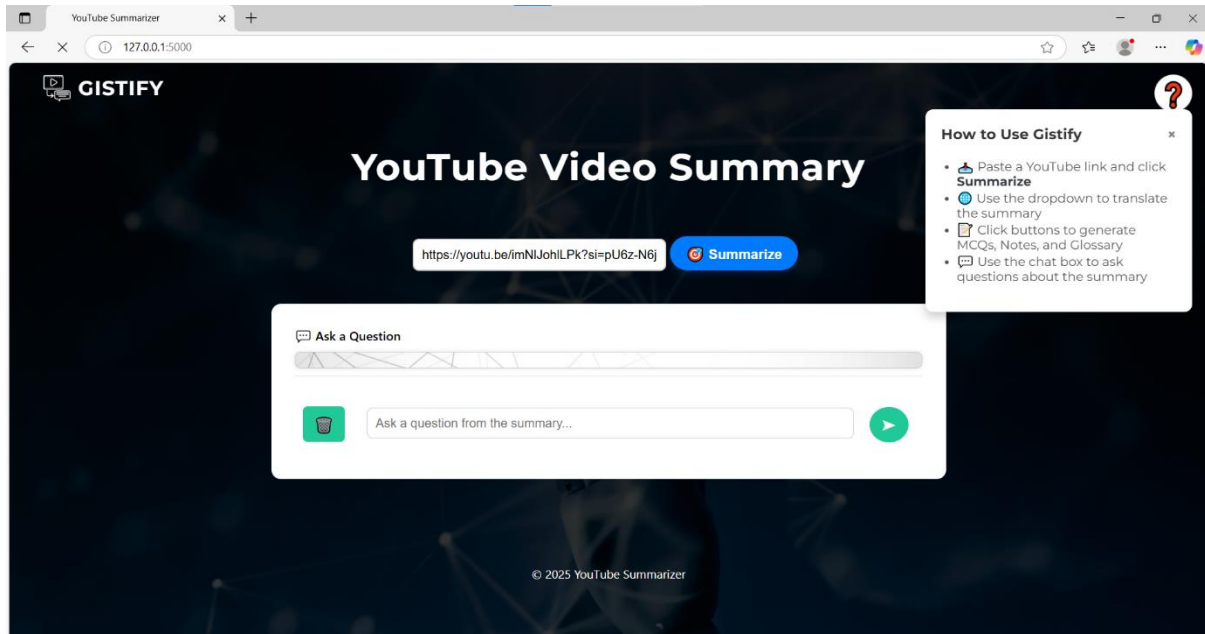
### 1. Welcome Page (Front UI):

- A clean and welcoming landing page introduces users to Gistify.
- Clicking “Let’s Get Started” smoothly redirects to the main summarizer interface.



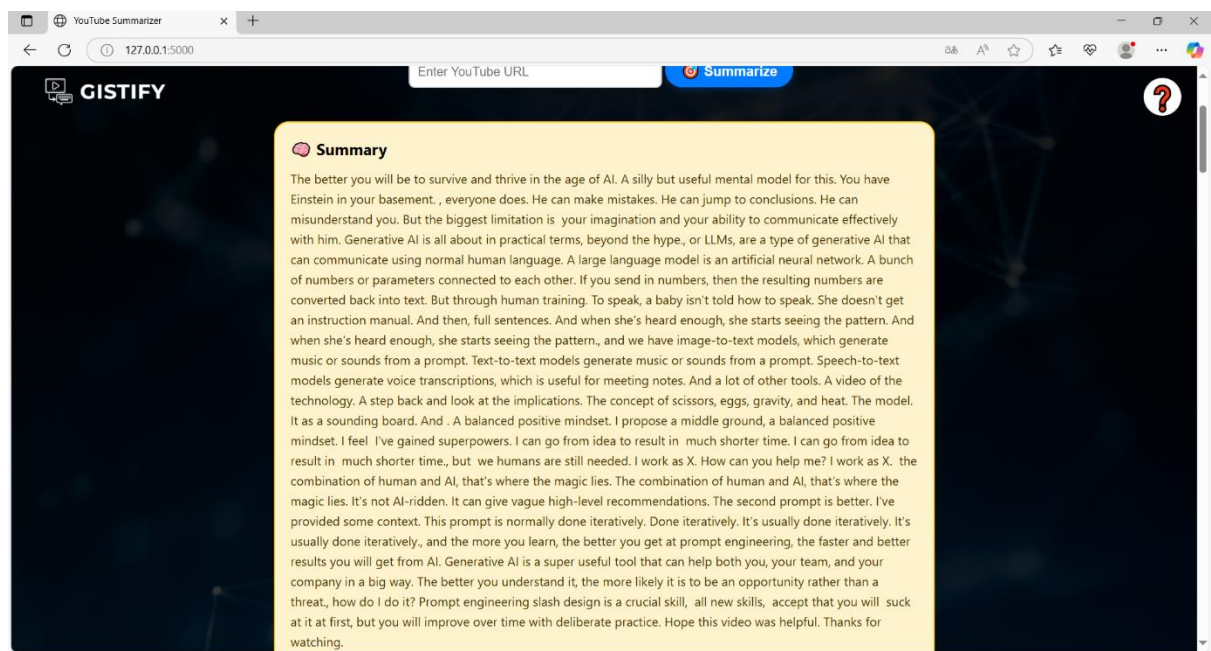
## 2. Youtube URL Submission:

- Users enter any educational YouTube link to begin processing.
- Gistify extracts and transcribes the video automatically using Whisper.



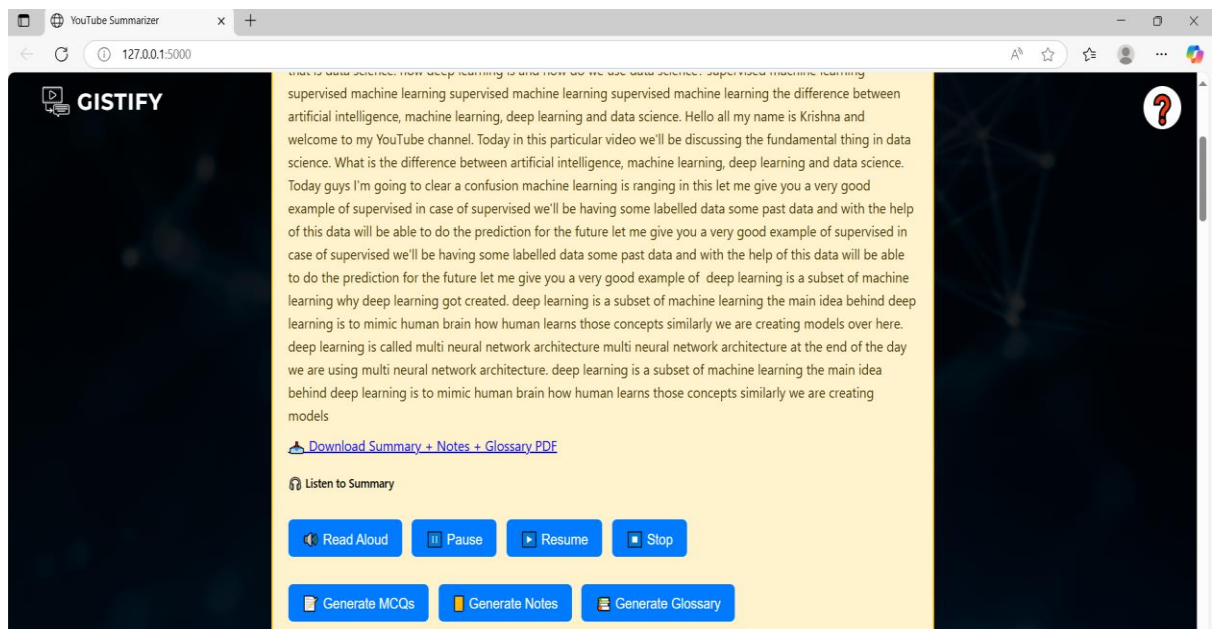
## 3.Summary Output:

- Summarization is powered by T5 model optimized with OpenVINO for fast CPU inference.
- The summary provides a clean, accurate gist of long lectures instantly.



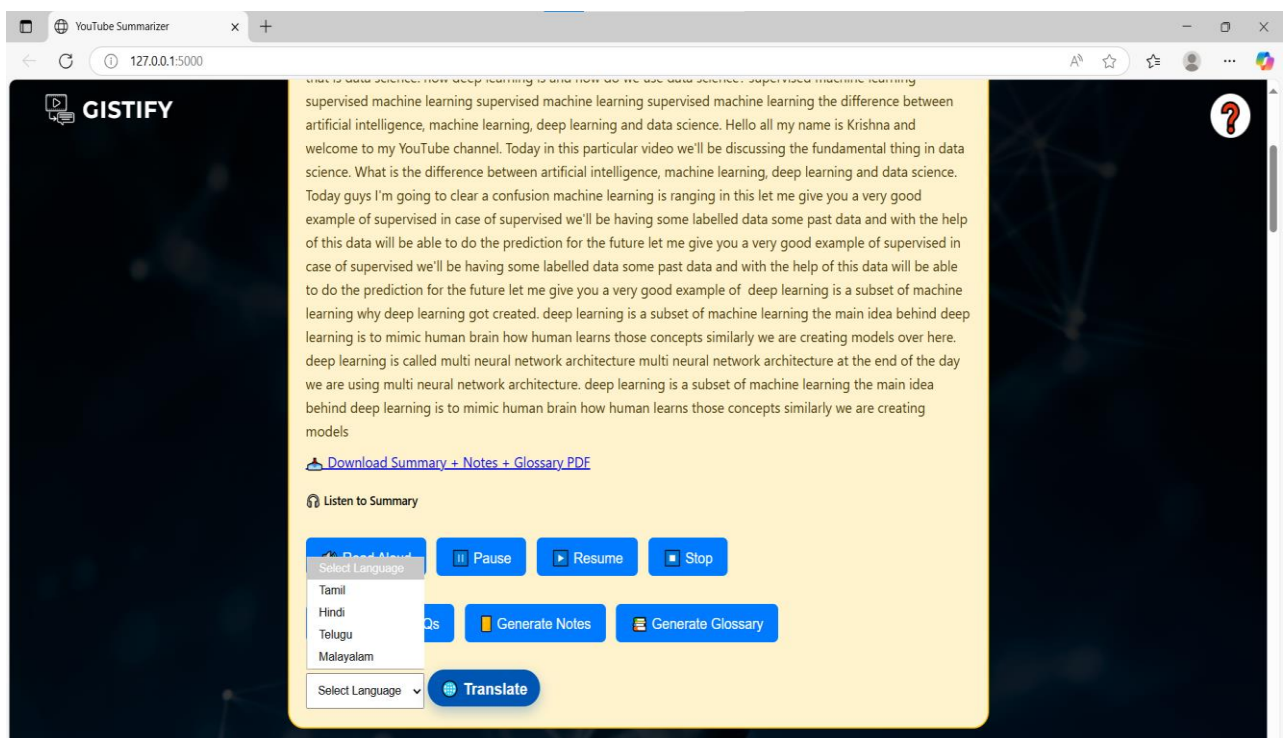
#### 4.Audio playback of summary:

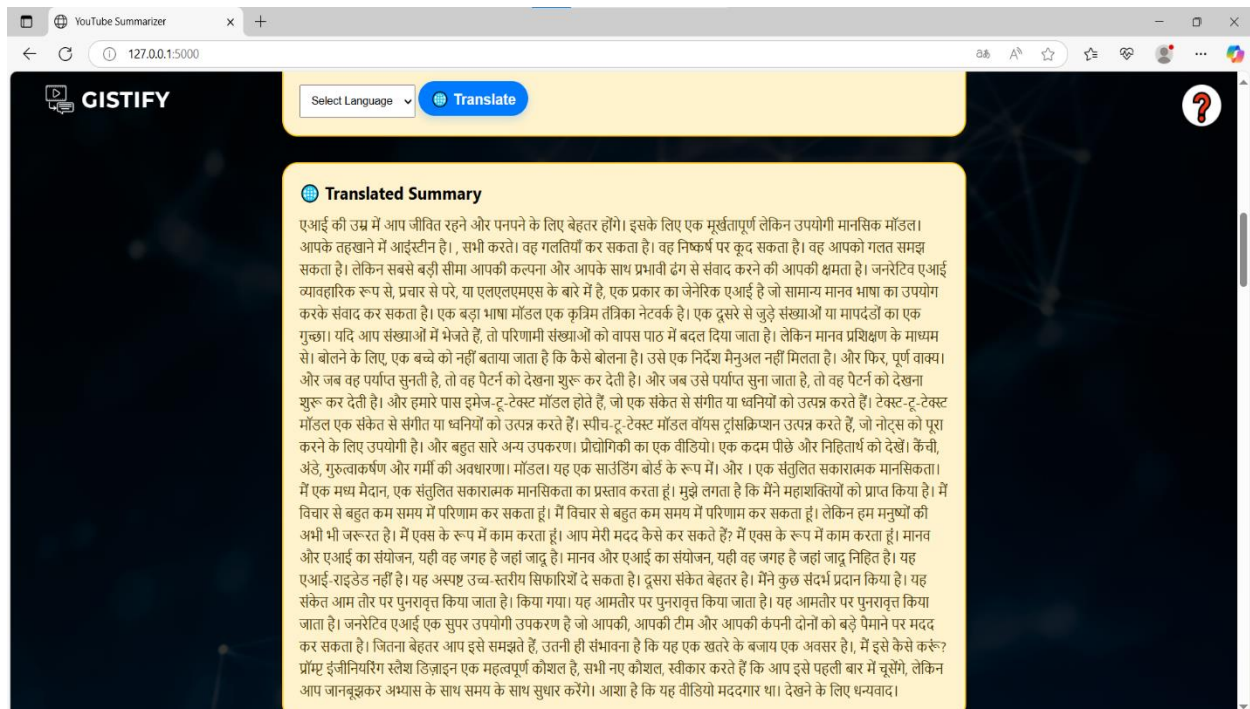
- Users can listen to the summarized content via a voice playback button.
- This improves accessibility and supports auditory learners.



#### 5.Translation feature:

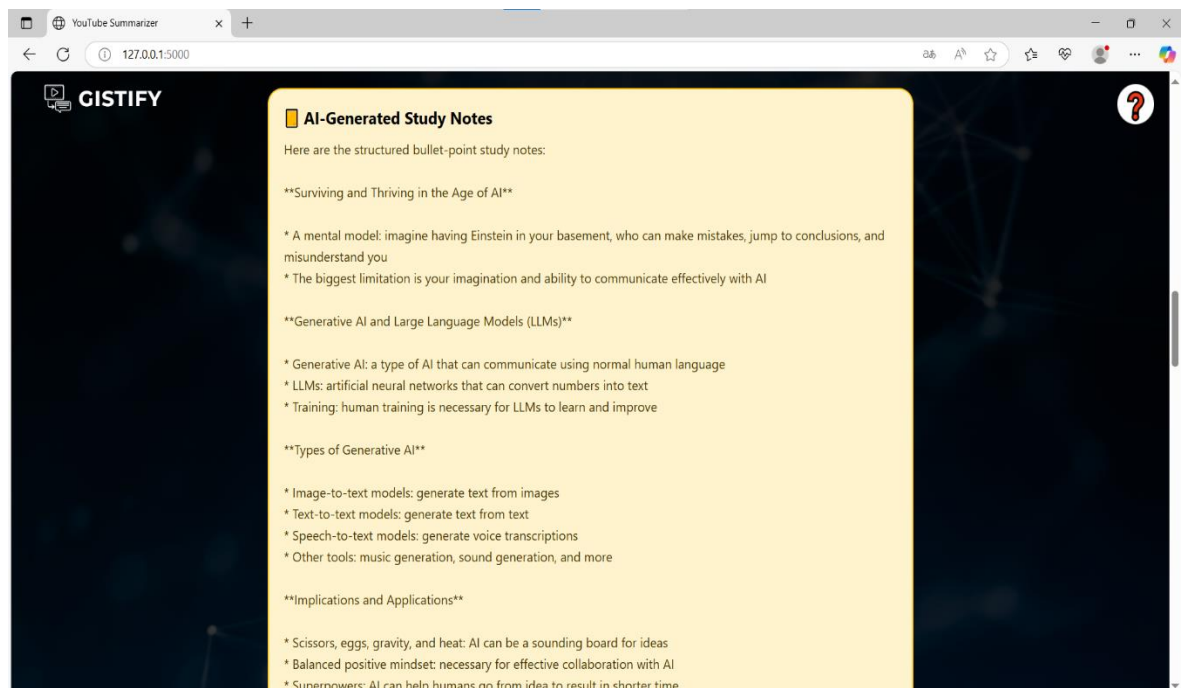
- Gistify supports 5 Indian languages using Deep Translator.
- Summaries are translated chunk-wise to handle long texts efficiently.





## 6. Generate notes:

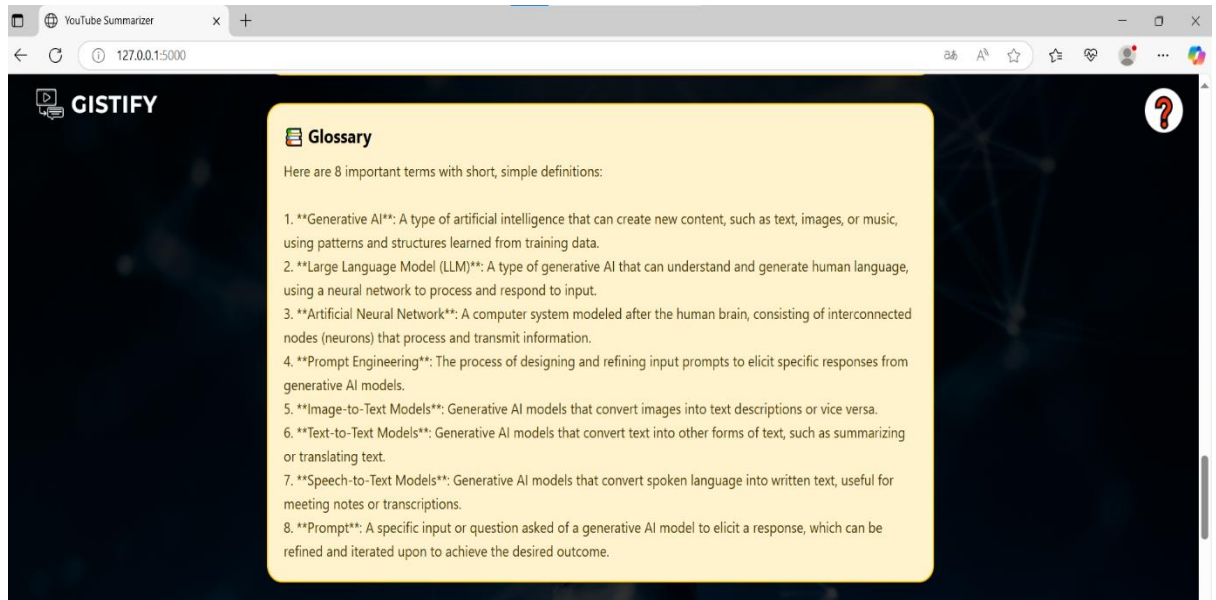
- Notes are generated as structured bullet points from the summary using LLM.
- These help in quick revision and better concept retention.





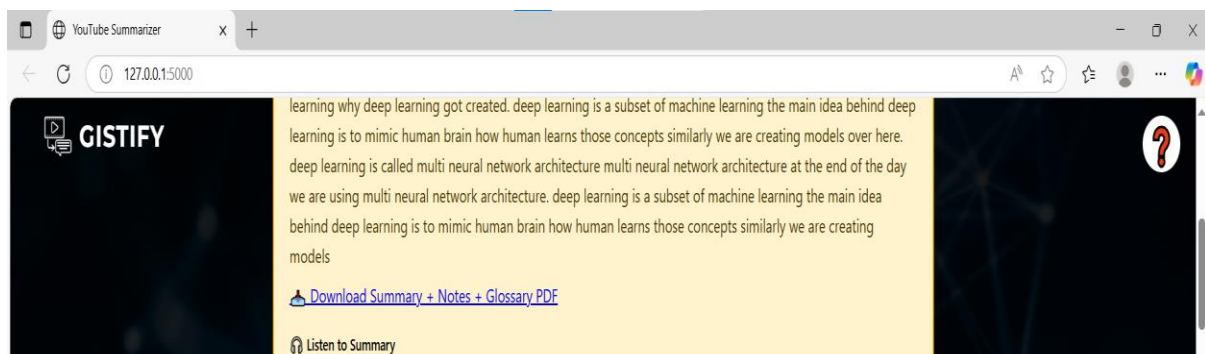
## 7. Generate glossary:

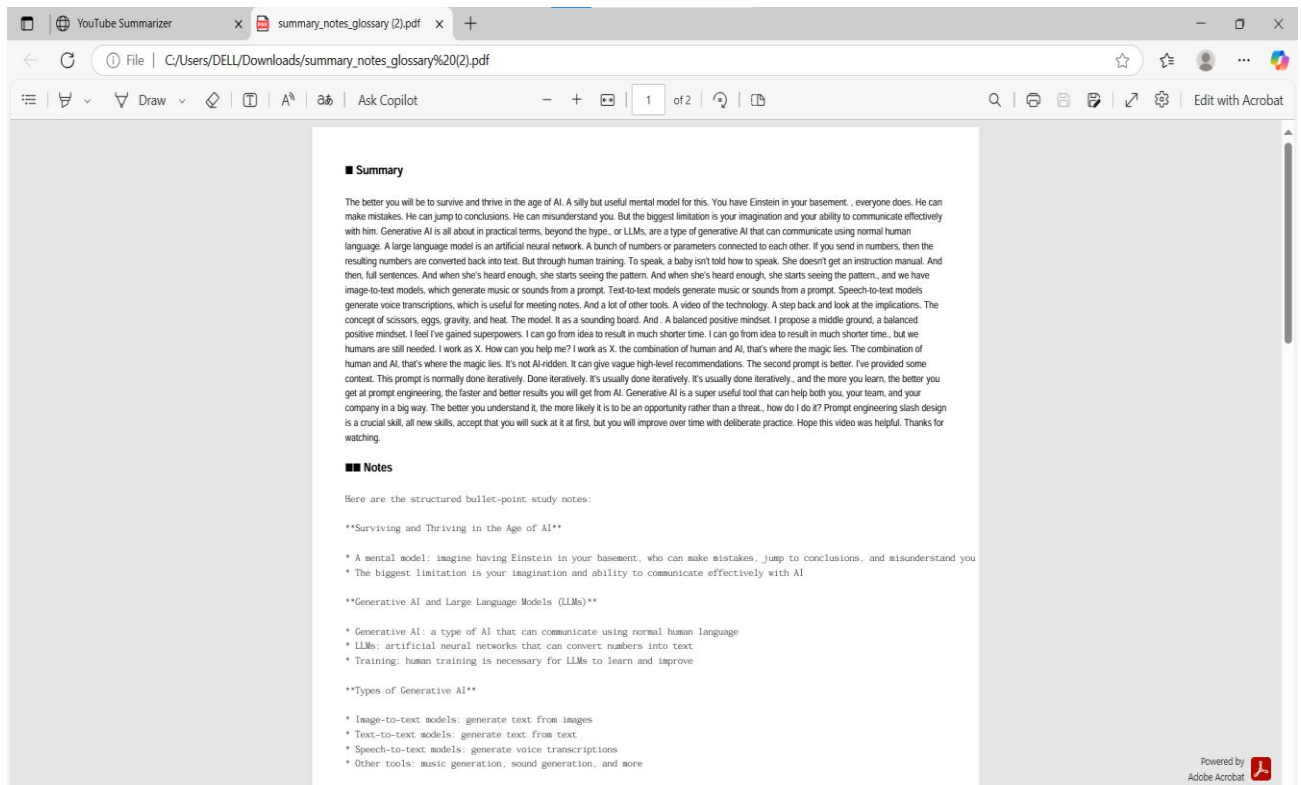
- Key terms from the lecture are extracted and explained in simple language.
- Ideal for understanding domain-specific vocabulary.



## 8. Download Summary + Notes + Glossary as PDF:

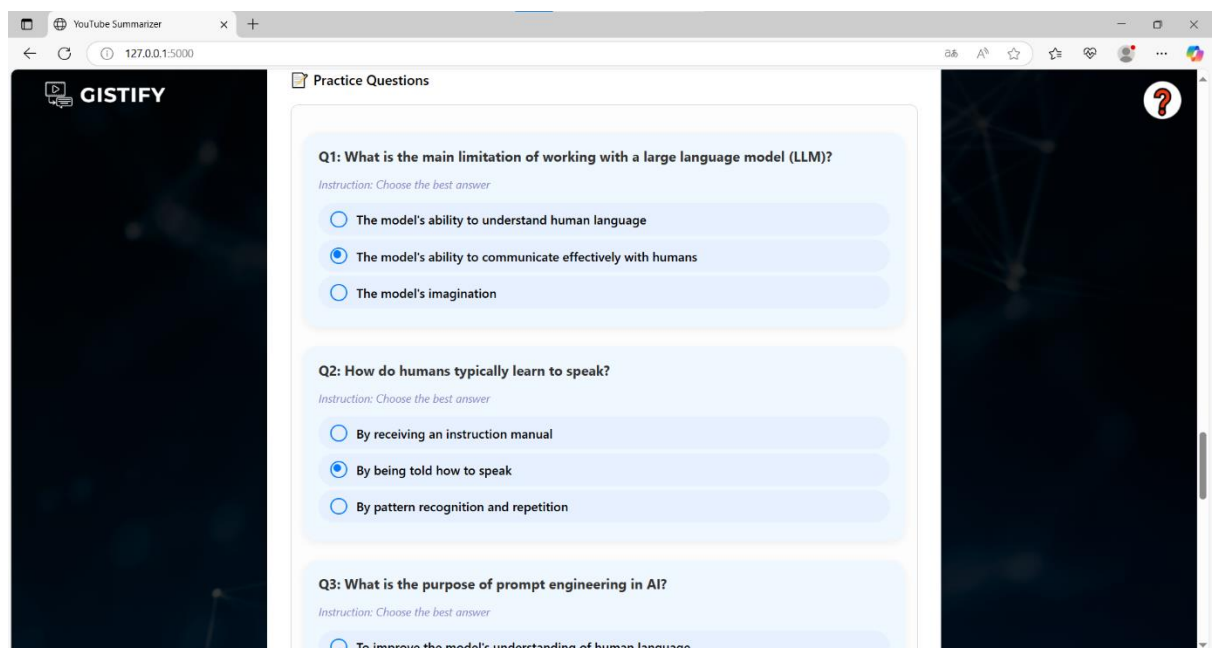
- Users can export the summary, notes, and glossary in a single PDF.
- Useful for offline study and sharing.

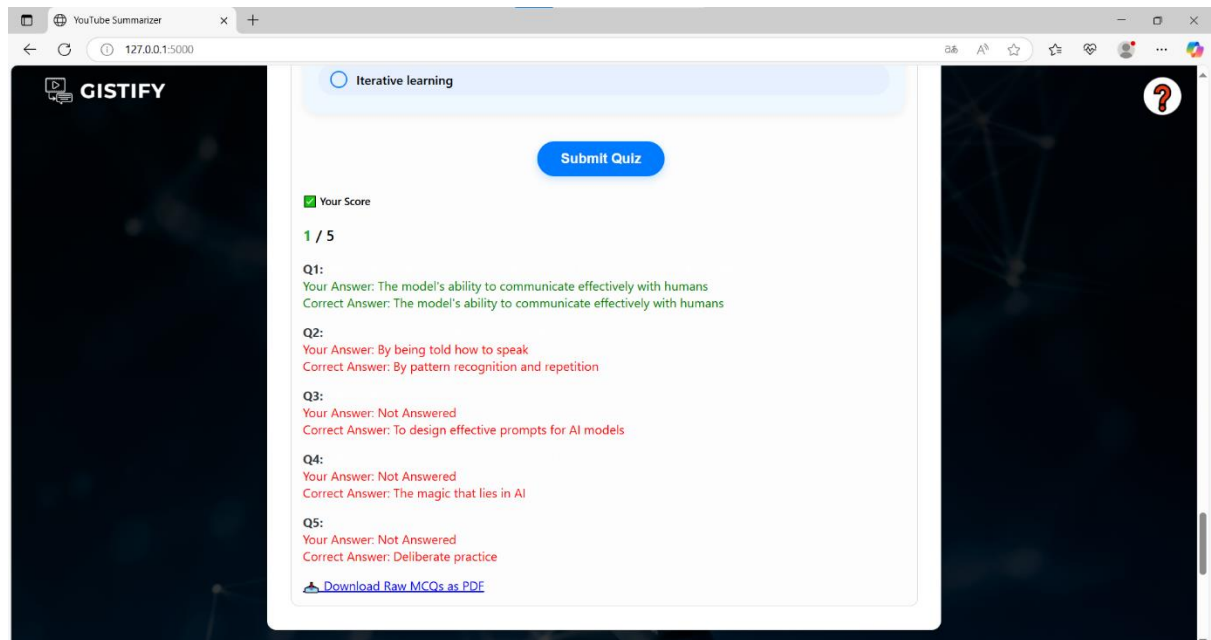




## 9. MCQ Generation:

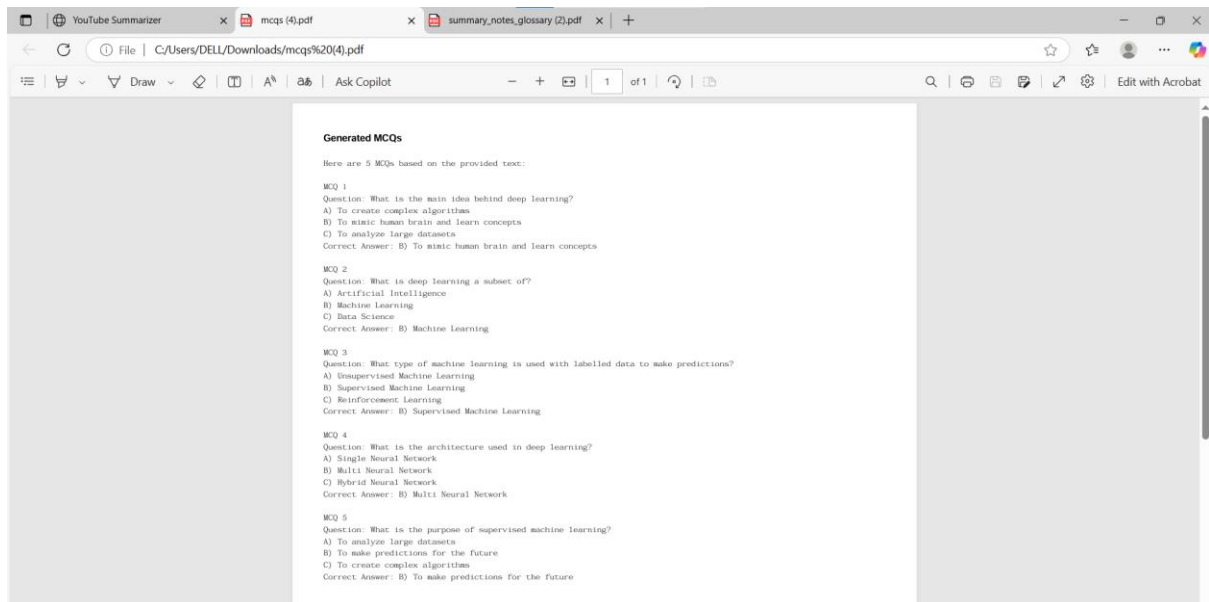
- AI-generated 3-option MCQs based on the summary aid in self-assessment.
- Correct answers are shown after submission to help evaluate learning.





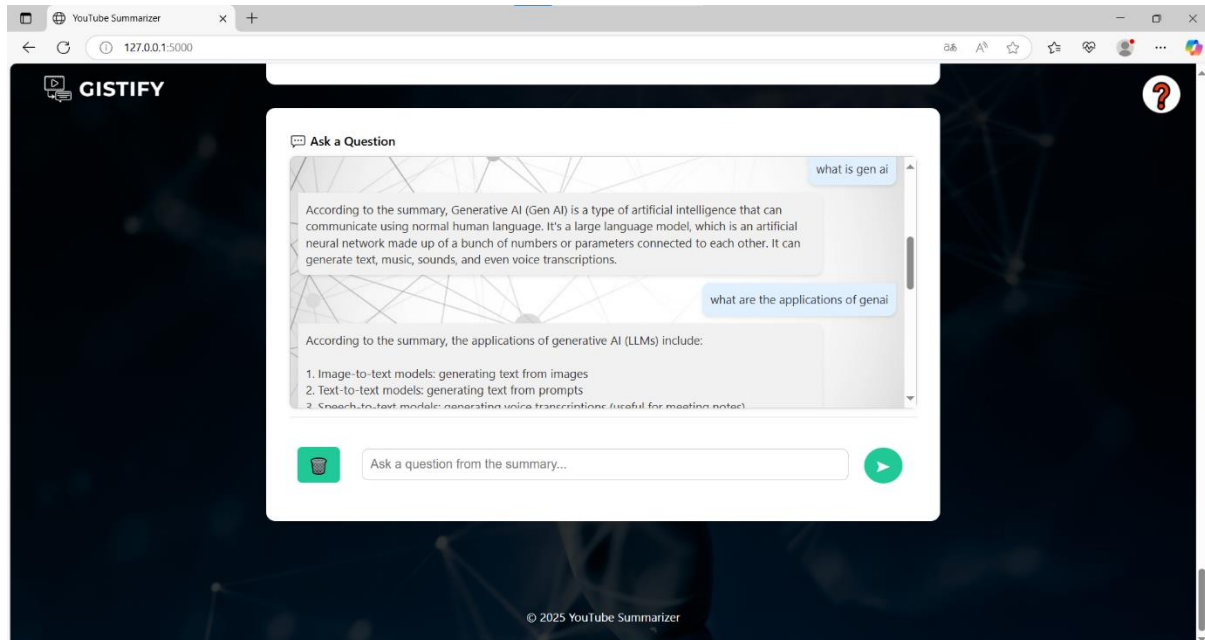
## 10.Download MCQs as PDF:

- Users can export the entire quiz in a clean PDF format.
- Supports test preparation and easy revision.



## 11. Chatbot (AI Doubt Solver)

- The chatbot uses the summary as context to answer student queries.
- Works fully offline using a local LLM (LLaMA 3 via LangChain).



## CHALLENGES FACED AND SOLUTIONS:

During the development of Gistify, several technical and usability-related challenges emerged. This section outlines key obstacles encountered throughout the project, along with the practical solutions implemented to ensure a robust and responsive application.

### 1.Slow Summarization Inference on CPU:

#### Challenge:

Initially, the summarization model (T5 from Hugging Face) took over **10–12 seconds** to generate summaries on an average laptop without GPU support. This delay affected the user experience and made the interface feel sluggish.

#### Solution:

To address this, the model was converted and optimized using **OpenVINO Toolkit**, significantly reducing the inference time to **3–4 seconds**. This allowed faster processing even on **CPU-only systems**, making Gistify usable on low-end devices without sacrificing accuracy.



## 2. Translation Failures for Long Summaries:

### Challenge:

When translating long summaries into regional languages, the application frequently encountered errors due to **API limits** and **text truncation**, especially with tools like Google Translate or Deep Translator.

### Solution:

The summary was automatically **split into 5000-character chunk**, allowing each segment to be translated individually. The translated chunks were then **merged seamlessly**, enabling accurate translation of long summaries without losing data or context.

## 3. Noisy and Repetitive Transcripts:

### Challenge:

Whisper-generated transcripts often contained **repeated phrases**, stutters, and filler content such as “so”, “you know”, “like”, which negatively affected summary quality and clarity.

### Solution:

Custom NLP cleaning functions (`pre_clean_transcript()` and `clean_summary()`) were developed using **regular expressions and tokenization**, which:

- \* Removed repeated words and sentences.
- \* Filtered out filler phrases.
- \* Fixed punctuation and spacing issues.

As a result, the input to the summarizer became more structured, and the output summary was significantly more readable.

## 4. Generic and Off-Topic Chatbot Responses:

### Challenge:

When using a large language model (LLM) for the chatbot, the responses were often **too generic or irrelevant**, especially when students asked video-specific doubts.

### Solution:

By leveraging **LangChain** with **context-based prompt engineering**, the LLM was directed to **focus only on the generated summary** while answering. This ensured the chatbot responses were **relevant, accurate, and grounded in the video content**, improving reliability.

## 5. Repeated Translation Not Triggering for Same URL:

### Challenge:

If the same YouTube URL was submitted again, the session variables prevented re-triggering of translation or other features, confusing users who expected fresh results.

### Solution:

The application was modified to **clear session states** on each new submission, ensuring the translation, summarization, and AI features could **run afresh every time**, regardless of URL repetition.

## 6. Heavy Resource Consumption for MCQ and Notes Generation

### Challenge:

Using LLMs like LLaMA-3 via Groq sometimes required **a few seconds to respond**, especially when generating multiple MCQs or long glossaries.

### Solution:

The LangChain chains were optimized to run **with lower temperature** and **shorter max tokens**, balancing response quality with performance. Additionally, Groq's high-speed LLM hosting ensured responses stayed within acceptable time ranges.

## CONCLUSION:

Gistify demonstrates how artificial intelligence and natural language processing can revolutionize the way students interact with video-based educational content. By automating the extraction of meaningful summaries, generating interactive MCQs, offering multilingual support, and enabling an AI-powered chatbot, Gistify bridges the gap between passive content consumption and active learning.

The use of optimized models like Whisper and OpenVINO-T5 ensures that the application performs efficiently on local machines, making it accessible even without high-end hardware. Additionally, the integration of LangChain and LLMs brings intelligent content generation capabilities that align well with real classroom needs.

Overall, Gistify stands as a smart, scalable, and user-friendly solution that supports learners in understanding, revising, and engaging with complex video content more effectively. It not only simplifies the study process but also lays the foundation for future enhancements in AI-driven educational tools.

**TEAM MEMBERS DETAILS:**

**NAME:** Shamsu Nisha N

**BRANCH:** B Tech CSE

**COLLEGE:** B S Abdur Rahman Crescent Institute of Science and Technology

**NAME:** Sharmila D

**BRANCH:** B Tech CSE

**COLLEGE:** B S Abdur Rahman Crescent Institute of Science and Technology

**NAME:** Tharani V S

**BRANCH:** B Tech CSE

**COLLEGE:** B S Abdur Rahman Crescent Institute of Science and Technology