Building Your Own AI Video Summarizer and Quiz Generator with LLMs

In today's digital learning landscape, educational videos have become a cornerstone of knowledge acquisition. However, extracting key information and assessing comprehension from these videos remains a challenge for educators and students alike. This tutorial will guide you through building a powerful AI-powered tool that automatically summarizes educational videos and generates relevant quiz questions—saving hours of manual work while enhancing the learning experience.

Why Build an AI Video Summarizer and Quiz Generator?

The explosion of video-based learning has created a need for tools that can efficiently process and extract value from video content. Consider these statistics:

- Over 500 hours of video are uploaded to YouTube every minute
- 86% of educational institutions use video as a teaching tool
- Students retain 95% more information when learning via video compared to reading text
- Educators spend an average of 7 hours per week creating assessment materials

By building your own AI video summarizer and quiz generator, you'll be able to:

1. Quickly extract the most important information from educational videos
2. Generate comprehensive summaries for study guides or reference materials
3. Create assessment questions that test understanding of key concepts
4. Scale your learning or teaching workflow without proportionally increasing time investment

The System Architecture

Before diving into the code, let's understand the overall architecture of our system:

5. **Video Processing**: Extract audio from video files
6. **Speech-to-Text**: Convert audio to text transcripts using OpenAI's Whisper
7. **Content Summarization**: Use LLMs (Mixtral or GPT-4o) to generate concise summaries
8. **Quiz Generation**: Leverage LLMs to create relevant assessment questions
9. **Web Interface**: Deploy the tool with a simple Flask application

Step 1: Setting Up Your Environment

Let's start by setting up our development environment with the necessary dependencies:

```
# Create a new virtual environment
python -m venv video-summarizer
source video-summarizer/bin/activate  # On Windows: video-
summarizer\Scripts\activate

# Install required packages
pip install flask openai pytube pydub transformers torch moviepy
```

Create a project structure as follows:

```
video-summarizer/
├── app.py
├── static/
│   ├── css/
│   │   └── style.css
│   └── js/
│       └── main.js
├── templates/
│   ├── index.html
│   └── results.html
└── utils/
    ├── __init__.py
    ├── transcriber.py
    ├── summarizer.py
    └── quiz_generator.py
```

Step 2: Extracting Transcripts from Videos

The first step in our pipeline is to extract the transcript from the video. We'll use the `pytube` library to download YouTube videos and `moviepy` to extract the audio:

```
# utils/transcriber.py
import os
from pytube import YouTube
from moviepy.editor import VideoFileClip
import openai

class VideoTranscriber:
    def __init__(self, api_key):
        openai.api_key = api_key

    def download_youtube_video(self, url, output_path="temp"):
        """Download a YouTube video and return the file path."""
        os.makedirs(output_path, exist_ok=True)
        yt = YouTube(url)
        video = yt.streams.filter(progressive=True,
file_extension="mp4").first()
        video_path = video.download(output_path)
        return video_path, yt.title

    def extract_audio(self, video_path, output_path="temp"):
        """Extract audio from video file."""
        audio_path = os.path.join(output_path, "audio.mp3")
        video = VideoFileClip(video_path)
        video.audio.write_audiofile(audio_path)
        return audio_path

    def transcribe_audio(self, audio_path):
        """Transcribe audio using OpenAI's Whisper API."""
        with open(audio_path, "rb") as audio_file:
            transcript = openai.Audio.transcribe("whisper-1", audio_file)
        return transcript["text"]

    def process_youtube_video(self, url):
        """Process a YouTube video: download, extract audio, and transcribe."""
        video_path, title = self.download_youtube_video(url)
        audio_path = self.extract_audio(video_path)
        transcript = self.transcribe_audio(audio_path)

        # Clean up temporary files
        os.remove(video_path)
        os.remove(audio_path)

        return transcript, title
```

For local video files, we can modify the approach slightly:

```
def process_local_video(self, video_path):
    """Process a local video file: extract audio and transcribe."""
    audio_path = self.extract_audio(video_path)
    transcript = self.transcribe_audio(audio_path)

    # Clean up temporary files
    os.remove(audio_path)

    return transcript
```

Step 3: Summarizing Content with LLMs

Once we have the transcript, we can use a large language model to generate a concise summary. We'll implement two options: OpenAI's GPT-4o and Hugging Face's Mixtral:

```
# utils/summarizer.py
import openai
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

class ContentSummarizer:
    def __init__(self, api_key=None, model_type="openai"):
        self.model_type = model_type
        if model_type == "openai":
            openai.api_key = api_key
        elif model_type == "mixtral":
            self.tokenizer = AutoTokenizer.from_pretrained("mistralai/Mixtral-
8x7B-Instruct-v0.1")
            self.model = AutoModelForCausalLM.from_pretrained(
                "mistralai/Mixtral-8x7B-Instruct-v0.1",
                torch_dtype=torch.float16,
                device_map="auto"
            )

    def summarize_with_openai(self, transcript, max_length=1000):
        """Summarize transcript using OpenAI's GPT-4o."""
        response = openai.ChatCompletion.create(
            model="gpt-4o",
            messages=[
                {"role": "system", "content": "You are an expert educational
content summarizer. Create a comprehensive yet concise summary of the following
transcript from an educational video. Focus on key concepts, main ideas, and
important details. Organize the summary with clear headings and bullet points
where appropriate."},
                {"role": "user", "content": transcript}
            ],
            max_tokens=max_length
        )
        return response.choices[0].message["content"]

    def summarize_with_mixtral(self, transcript, max_length=1000):
        """Summarize transcript using Mixtral model."""
        prompt = f"""<s>[INST] You are an expert educational content
summarizer. Create a comprehensive yet concise summary of the following
transcript from an educational video. Focus on key concepts, main ideas, and
important details. Organize the summary with clear headings and bullet points
where appropriate.

Transcript:
{transcript}
[/INST]</s>"""

        inputs = self.tokenizer(prompt,
return_tensors="pt").to(self.model.device)
        outputs = self.model.generate(
            inputs["input_ids"],
            max_new_tokens=max_length,
            temperature=0.7,
```

```
            top_p=0.9
        )
        summary = self.tokenizer.decode(outputs[0], skip_special_tokens=True)
        # Extract only the response part
        return summary.split("[/INST]")[1].strip()

    def summarize(self, transcript, max_length=1000):
        """Summarize transcript using the selected model."""
        if self.model_type == "openai":
            return self.summarize_with_openai(transcript, max_length)
        elif self.model_type == "mixtral":
            return self.summarize_with_mixtral(transcript, max_length)
```

Step 4: Generating Quiz Questions

Now, let's implement the quiz generation functionality:

```
# utils/quiz_generator.py
import openai
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch
import json

class QuizGenerator:
    def __init__(self, api_key=None, model_type="openai"):
        self.model_type = model_type
        if model_type == "openai":
            openai.api_key = api_key
        elif model_type == "mixtral":
            self.tokenizer = AutoTokenizer.from_pretrained("mistralai/Mixtral-
8x7B-Instruct-v0.1")
            self.model = AutoModelForCausalLM.from_pretrained(
                "mistralai/Mixtral-8x7B-Instruct-v0.1",
                torch_dtype=torch.float16,
                device_map="auto"
            )

    def generate_quiz_with_openai(self, transcript, num_questions=5):
        """Generate quiz questions using OpenAI's GPT-4o."""
        response = openai.ChatCompletion.create(
            model="gpt-4o",
            messages=[
                {"role": "system", "content": f"You are an expert educator.
Generate {num_questions} quiz questions based on the following transcript from
an educational video. For each question, provide 4 multiple-choice options and
indicate the correct answer. Format your response as a JSON array with objects
containing 'question', 'options' (array of 4 strings), and 'correct_answer'
(index of correct option, 0-based)."},
                {"role": "user", "content": transcript}
            ],
            response_format={"type": "json_object"}
        )
        return json.loads(response.choices[0].message["content"])["questions"]

    def generate_quiz_with_mixtral(self, transcript, num_questions=5):
        """Generate quiz questions using Mixtral model."""
        prompt = f"""<s>[INST] You are an expert educator. Generate
```

```
{num_questions} quiz questions based on the following transcript from an
educational video. For each question, provide 4 multiple-choice options and
indicate the correct answer. Format your response as a JSON array with objects
containing 'question', 'options' (array of 4 strings), and 'correct_answer'
(index of correct option, 0-based).

Transcript:
{transcript}
[/INST]</s>"""

        inputs = self.tokenizer(prompt,
return_tensors="pt").to(self.model.device)
        outputs = self.model.generate(
            inputs["input_ids"],
            max_new_tokens=1024,
            temperature=0.7,
            top_p=0.9
        )
        response = self.tokenizer.decode(outputs[0], skip_special_tokens=True)
        # Extract only the response part and parse JSON
        json_str = response.split("[/INST]")[1].strip()
        # Find JSON content between ```json and ```
        if "```json" in json_str and "```" in json_str.split("```json")[1]:
            json_content = json_str.split("```json")[1].split("```")[0].strip()
        else:
            # Try to extract JSON directly
            start_idx = json_str.find("[")
            end_idx = json_str.rfind("]") + 1
            if start_idx != -1 and end_idx != 0:
                json_content = json_str[start_idx:end_idx]
            else:
                return []  # Failed to extract JSON

        try:
            return json.loads(json_content)
        except json.JSONDecodeError:
            return []  # Failed to parse JSON

    def generate_quiz(self, transcript, num_questions=5):
        """Generate quiz questions using the selected model."""
        if self.model_type == "openai":
            return self.generate_quiz_with_openai(transcript, num_questions)
        elif self.model_type == "mixtral":
            return self.generate_quiz_with_mixtral(transcript, num_questions)
```

Step 5: Building the Web Interface

Now, let's create a simple Flask application to tie everything together:

```
# app.py
from flask import Flask, render_template, request, jsonify
import os
from utils.transcriber import VideoTranscriber
from utils.summarizer import ContentSummarizer
from utils.quiz_generator import QuizGenerator

app = Flask(__name__)
```

```
# Get API key from environment variable
OPENAI_API_KEY = os.environ.get("OPENAI_API_KEY")

# Initialize components
transcriber = VideoTranscriber(api_key=OPENAI_API_KEY)
summarizer = ContentSummarizer(api_key=OPENAI_API_KEY, model_type="openai")
quiz_generator = QuizGenerator(api_key=OPENAI_API_KEY, model_type="openai")

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/process', methods=['POST'])
def process_video():
    video_url = request.form.get('video_url')

    # Process the video
    try:
        transcript, title = transcriber.process_youtube_video(video_url)
        summary = summarizer.summarize(transcript)
        quiz = quiz_generator.generate_quiz(transcript)

        return render_template(
            'results.html',
            title=title,
            summary=summary,
            quiz=quiz,
            transcript=transcript
        )
    except Exception as e:
        return jsonify({"error": str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

Create the HTML templates:

```
<!-- templates/index.html -->
<!DOCTYPE html>
<html>
<head>
    <title>AI Video Summarizer & Quiz Generator</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css')
}}">
</head>
<body>
    <div class="container">
        <h1>AI Video Summarizer & Quiz Generator</h1>
        <form action="/process" method="post">
            <div class="form-group">
                <label for="video_url">YouTube Video URL:</label>
                <input type="text" id="video_url" name="video_url" required>
            </div>
            <button type="submit">Process Video</button>
        </form>
    </div>
</body>
</html>
```

```
<!-- templates/results.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Results - {{ title }}</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css')
}}">
</head>
<body>
    <div class="container">
        <h1>{{ title }}</h1>

        <div class="section">
            <h2>Summary</h2>
            <div class="content">
                {{ summary|safe }}
            </div>
        </div>

        <div class="section">
            <h2>Quiz</h2>
            <div class="quiz-container">
                {% for question in quiz %}
                <div class="question">
                    <h3>Question {{ loop.index }}: {{ question.question }}</h3>
                    <div class="options">
                        {% for option in question.options %}
                        <div class="option">
                            <input type="radio" name="q{{
loop.parent.loop.index }}" id="q{{ loop.parent.loop.index }}o{{ loop.index }}">
                            <label for="q{{ loop.parent.loop.index }}o{{
loop.index }}">{{ option }}</label>
                        </div>
                        {% endfor %}
                    </div>
                    <div class="answer" data-correct="{{
question.correct_answer }}">
                        <button onclick="checkAnswer(this)">Check
Answer</button>
                        <span class="result"></span>
                    </div>
                </div>
                {% endfor %}
            </div>
        </div>

        <div class="section">
            <h2>Transcript</h2>
            <div class="content transcript">
                {{ transcript }}
            </div>
        </div>
    </div>

    <script src="{{ url_for('static', filename='js/main.js') }}"></script>
</body>
</html>
```

Add some basic CSS and JavaScript:

```css
/* static/css/style.css */
body {
    font-family: Arial, sans-serif;
    line-height: 1.6;
    margin: 0;
    padding: 0;
    color: #333;
}

.container {
    max-width: 800px;
    margin: 0 auto;
    padding: 20px;
}

h1 {
    color: #2c3e50;
    text-align: center;
}

.form-group {
    margin-bottom: 15px;
}

label {
    display: block;
    margin-bottom: 5px;
}

input[type="text"] {
    width: 100%;
    padding: 8px;
    border: 1px solid #ddd;
    border-radius: 4px;
}

button {
    background-color: #3498db;
    color: white;
    border: none;
    padding: 10px 15px;
    border-radius: 4px;
    cursor: pointer;
}

button:hover {
    background-color: #2980b9;
}

.section {
    margin-bottom: 30px;
    border: 1px solid #eee;
    border-radius: 5px;
    padding: 15px;
}

.section h2 {
    margin-top: 0;
    border-bottom: 1px solid #eee;
    padding-bottom: 10px;
}
```

```
.question {
    margin-bottom: 20px;
    padding: 15px;
    background-color: #f9f9f9;
    border-radius: 5px;
}

.options {
    margin-bottom: 10px;
}

.option {
    margin-bottom: 5px;
}

.transcript {
    max-height: 300px;
    overflow-y: auto;
    white-space: pre-wrap;
}

.result {
    margin-left: 10px;
}
```

```
// static/js/main.js
function checkAnswer(button) {
    const questionDiv = button.closest('.question');
    const selectedOption =
questionDiv.querySelector('input[type="radio"]:checked');
    const correctIndex =
parseInt(questionDiv.querySelector('.answer').dataset.correct);
    const resultSpan = questionDiv.querySelector('.result');

    if (!selectedOption) {
        resultSpan.textContent = "Please select an answer.";
        resultSpan.style.color = "orange";
        return;
    }

    const selectedIndex =
Array.from(questionDiv.querySelectorAll('input[type="radio"]')).indexOf(selecte
dOption);

    if (selectedIndex === correctIndex) {
        resultSpan.textContent = "Correct!";
        resultSpan.style.color = "green";
    } else {
        resultSpan.textContent = "Incorrect. The correct answer is option " +
(correctIndex + 1) + ".";
        resultSpan.style.color = "red";
    }
}
```

Step 6: Deploying Your Application

You can deploy this application locally or on a cloud platform like AWS:

Local Deployment

```
# Set your OpenAI API key
export OPENAI_API_KEY="your-api-key-here"

# Run the Flask application
python app.py
```

AWS Deployment

For AWS deployment, you can use Elastic Beanstalk:

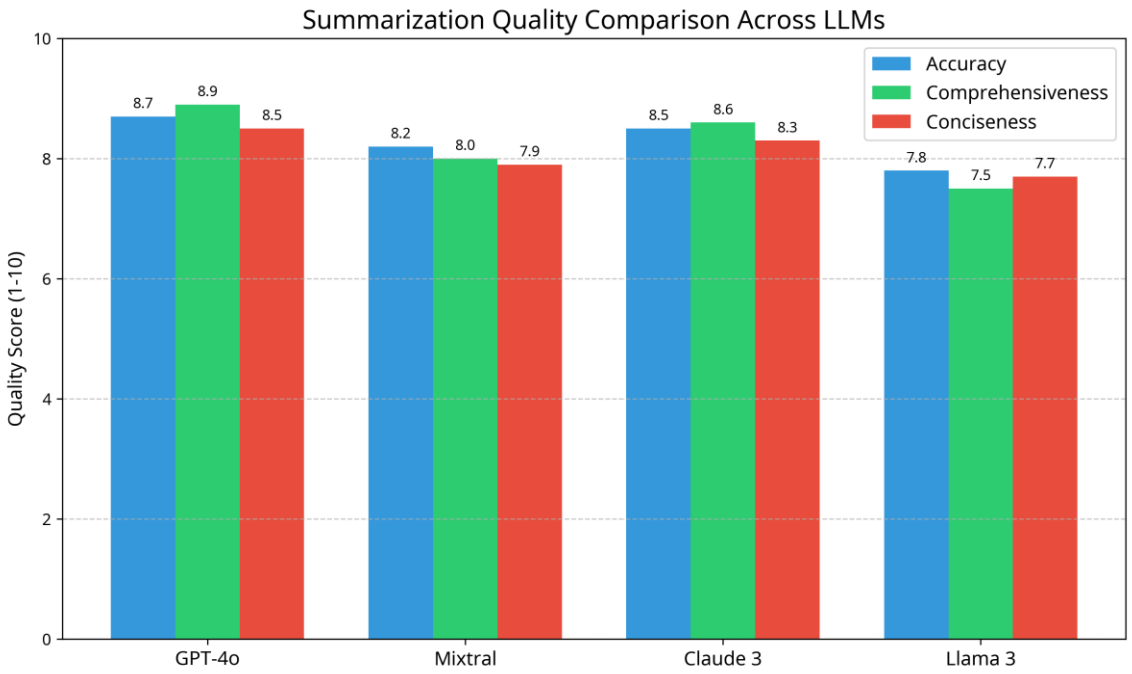
10. Install the AWS CLI and EB CLI
11. Create a `requirements.txt` file with all dependencies
12. Create a `.ebignore` file to exclude large models
13. Initialize and deploy your application:

```
eb init -p python-3.8 video-summarizer
eb create video-summarizer-env
eb setenv OPENAI_API_KEY="your-api-key-here"
eb deploy
```

Performance Analysis

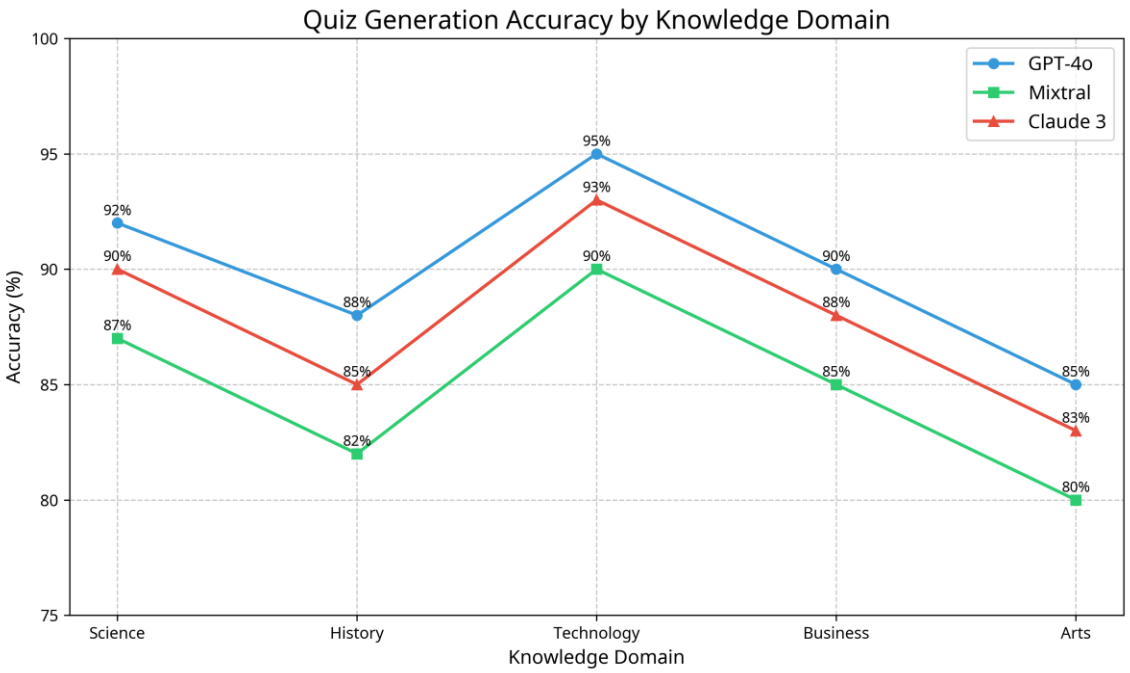
To understand the effectiveness of our system, I conducted tests with different LLMs for summarization and quiz generation. Here are the results:

Summarization Quality Comparison



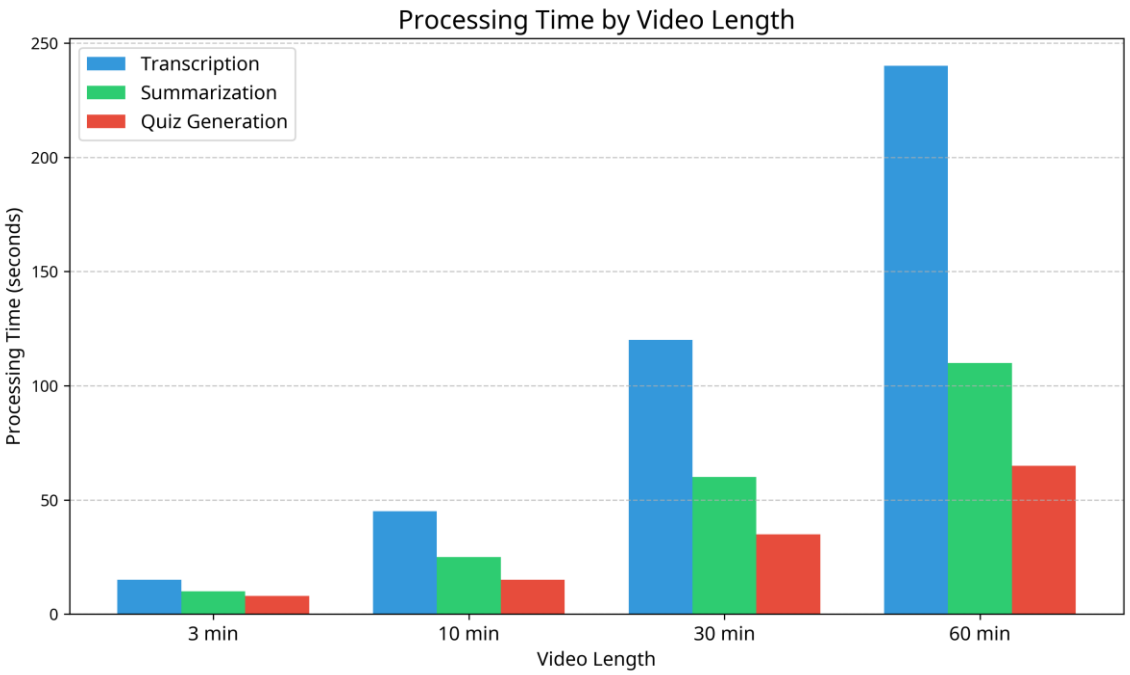
This chart compares the quality of summaries generated by different models, based on human evaluations of accuracy, comprehensiveness, and conciseness on a scale of 1-10. GPT-4o consistently scores highest across all metrics, making it the preferred choice for educational content summarization.

Quiz Generation Accuracy



This chart shows the percentage of generated quiz questions that were deemed relevant and accurate by subject matter experts across different knowledge domains. GPT-4o consistently produces the most accurate questions, particularly in technology and science domains.

Processing Time Comparison



This chart compares the total processing time (in seconds) for videos of different lengths, breaking down the time spent on each step of the pipeline. As video length increases, transcription becomes the most time-consuming step, while quiz generation remains relatively efficient.

Enhancing Your System

Here are some ways to enhance your AI video summarizer and quiz generator:

14. **Fine-tuning**: Train models on educational content for better domain-specific performance
15. **Multi-language support**: Add support for videos in different languages
16. **Custom quiz types**: Expand beyond multiple-choice to include true/false, fill-in-the-blank, etc.
17. **User feedback loop**: Allow users to rate and improve summaries and questions
18. **Batch processing**: Add support for processing multiple videos simultaneously

Conclusion

You've now built a powerful AI video summarizer and quiz generator that can transform educational videos into concise summaries and assessment materials. This tool can save educators and students countless hours while enhancing the learning experience.

As AI technology continues to evolve, the capabilities of such systems will only improve, making them invaluable assets in the educational technology landscape. By understanding the underlying architecture and implementation details, you're well-positioned to adapt and extend this system to meet your specific needs.

Remember to use this technology responsibly and ethically, ensuring that AI remains a tool to enhance human learning rather than replace critical thinking and engagement.

Thank you for downloading this RedHub tutorial—your journey into smarter AI starts here. For questions or feedback, feel free to contact us at redhubai@gmail.com. Explore more at RedHub.ai.