How can we automate the process of editing videos?

Kaan Gögcay

June 12, 2025

Abstract

This research investigates the automation of explainer video production using structured client data as input. The goal is to develop a fully automated pipeline that can transform raw data into professionally edited, branded video content with minimal human intervention. Through an extensive evaluation of commercial tools and open-source technologies, the study identifies the limitations of existing platforms and demonstrates the feasibility of a Python-based workflow. The proposed system integrates AI-driven scripting, audio recognition, and dynamic media overlays to generate personalized videos programmatically. The findings highlight the potential of hybrid, modular solutions for scalable and intelligent video generation.

Contents

1 Introduction **1**

2 Research Approach **2**
2.1 Tooling and Technology Exploration . . . 2
2.1.1 Descript . . . 2
2.1.2 Adobe Premiere Pro . . . 2
2.1.3 DaVinci Resolve . . . 2
2.1.4 Python-Based Workflow . . . 2
2.2 Automated Video Generation Pipeline . . . 3
2.2.1 Input Handling . . . 3
2.2.2 Prompt Templates . . . 3
2.2.3 Script Generation . . . 3
2.2.4 Video Segment Generation . . . 3
2.2.5 Editing Videos . . . 3
2.2.6 Post-Processing and Concatenation . . . 4
2.2.7 Automated Video Job Execution via REST API . . . 4
2.2.8 Experimentation and Iterative Evaluation . . . 4

3 Conclusion **4**

1 Introduction

As demand grows for personalized, scalable, and engaging video content, organizations are increasingly seeking ways to automate video production. Manually editing videos remains time-consuming, costly, and inconsistent—especially when generating explainer videos from structured data such as client financial profiles or insurance records. This presents a unique opportunity for automation.

This research explores how client data can be transformed into a fully edited, branded explainer video—entirely through automation. The primary objective is to identify the tools, technologies, and workflows needed to build an end-to-end pipeline that transforms raw data into polished video output with minimal human intervention.

To achieve this, the study combines tool analysis, hands-on experimentation, and iterative feedback. It evaluates both commercial editing platforms and open-source libraries, and ultimately proposes a

Python-based pipeline capable of dynamically structuring, generating, and editing video content based on AI-generated scripts and audio recognition.

2 Research Approach

2.1 Tooling and Technology Exploration

This section investigates the feasibility of automating video editing processes using technologies that can be programmatically controlled. The goal was to identify solutions capable of dynamic content generation, particularly in response to detected audio input.

The research began with an exploratory review of existing software to evaluate their potential for automation. This was followed by a technical analysis of open-source libraries for their suitability in scripting-based workflows. Extensive hands-on experimentation was conducted to test the limitations and possibilities of available tools. Feedback from mentors and stakeholders was regularly gathered to iteratively improve the approach.

2.1.1 Descript

Descript is an AI-powered editing platform with transcript-based editing. Although well-suited for semi-automated workflows, its API exposes only limited functionality, making full automation difficult. Moreover, it lacks mechanisms for dynamically inserting media based on spoken content, limiting its usefulness in a fully automated system.

2.1.2 Adobe Premiere Pro

Adobe Premiere Pro supports scripting via ExtendScript (JavaScript), but lacks native support for speech recognition or transcription within its automation capabilities. Additionally, it requires desktop installation, complicating deployment in server-based or cloud-native environments.

2.1.3 DaVinci Resolve

DaVinci Resolve shares similar limitations with Adobe Premiere Pro. While offering strong manual editing tools, it provides minimal scripting support and lacks audio-driven features critical for automation.

2.1.4 Python-Based Workflow

An alternative, more flexible solution is a custom-built Python-based pipeline using libraries such as `moviepy`, `OpenCV`, and `Whisper`. While these libraries are limited when used individually, combined they allow for highly customizable workflows with nearly full automation capabilities. Supported features include:

- Cropping video segments
- Concatenating multiple videos
- Speech-to-text transcription using AI
- Conditional media overlays triggered by audio content
- Background removal using chroma keying
- Basic video transitions such as fade-ins and fade-outs

Each of these capabilities was tested through practical experiments and validated with feedback from mentors and users.

While tools like Descript or Premiere Pro may still serve specific sub-tasks, they fall short for end-to-end automation. A custom Python-based approach, though more technically demanding, enables the level of flexibility and contextual awareness needed for dynamic, audio-driven video creation.

2.2 Automated Video Generation Pipeline

This section describes the implementation of a fully automated video creation workflow, from data input to the generation of a complete, polished video.

2.2.1 Input Handling

Input is provided as JSON files containing client-specific data. These files represent either financial or insurance product profiles. Users upload the JSON via a POST request to the REST API, specifying the use case type (e.g., "Financial" or "Insurance"). The data is parsed and integrated into a prompt used for video script generation.

2.2.2 Prompt Templates

A local data store contains prompt templates for each use case. For the financial use case, prompts are as follows:

- **Explainer prompt Template:** "Could you explain the financial products found in this client data? Only mention the most important information so the video doesn't get too lengthy (max 45s)."
- **Intro prompt Template:** "Write a very short video introduction mentioning what will be covered in the video (max 10s)."
- **Outro prompt Template:** "Write a very short outro highlighting a 'Summary of key points' or a 'Checklist of recommended actions' (max 15s)."

Based on the use case selected in the API request, the appropriate prompt set is retrieved.

2.2.3 Script Generation

The full prompt for generating the explainer script is composed by combining the client data with the explainer prompt template.

- `explainer_prompt = explainer_prompt_template + JSON_data`

This `explainer_prompt` is then given to a generative AI. As response we will obtain an explainer video script. this video script will be used to create the prompt of the intro and the outro.

- `intro_prompt = intro_prompt_template + explainer_video_script`
- `outro_prompt = outro_prompt_template + explainer_video_script`

Once these prompts are created we can feed them to the same generative AI to receive an intro script and an outro script.

2.2.4 Video Segment Generation

With the explainer, intro, and outro scripts ready, we can generate the corresponding video segments using a text-to-video model. The intro script is fed into the model to produce the intro video, and similarly, the explainer and outro scripts generate their respective segments.

2.2.5 Editing Videos

After generating the individual video segments, further editing is applied to enhance visual presentation and engagement. In the explainer segment, the speaker is placed in a small bubble in the bottom-left corner, with a neutral or softly branded background. This is done using the library OpenCV. Using OpenCV, we can crop the video of the speaker to a circle and make the rest a green screen. Next, we choose a background image or video. Then we take the cropped speaker and position it in the bottom-left of the background, and we remove the green screen with chroma keying. This way, we can have the speaker in a bubble in the bottom-left. This is possible using the libraries OpenCV for the cropping and green-screening, and moviepy for accessing the video clips in code and overlaying the video timelines over one another.

After generating the individual video segments, further editing is applied to enhance visual presentation and engagement. In particular, the explainer segment features the speaker displayed in a small circular bubble positioned at the bottom-left corner, overlaid on a neutral or softly branded background.

This effect is achieved using a combination of the `OpenCV` and `moviepy` libraries. First, `OpenCV` is used to crop the speaker's video feed into a circular shape and apply a green screen to the surrounding area. Then, a background image or video is selected. Using chroma keying, the green screen is removed and the circular speaker cutout is composited onto the chosen background. Finally, `moviepy` is employed to access and manipulate video timelines, allowing seamless overlay of the processed speaker footage onto the background layer.

To enrich the content, visual elements such as graphs or other media appear next to the speaker when relevant topics are mentioned. This is achieved by transcribing the video word-for-word using the `whisper` library. When the script contains content that can be visualized, such as data trends or product breakdowns, corresponding visuals are generated and timed to appear during specific sentences. Since whisper also returns the timestamp of each word we can use those to show and hide media at the appropriate time.

Since the graph generated uses the original video script and whisper returns the transcribed video, there can be minor differences in the video script. To make sure we still show and hide media at the correct time we can use the Knuth-Morris-Pratt (KMP) to compare similar strings to one another.

2.2.6 Post-Processing and Concatenation

After generating and editing the video segments, they are concatenated using Python-based tools. Transitions are added between segments for smoother viewing, and brand elements, such as logos or color overlays, can be consistently applied throughout the video.

2.2.7 Automated Video Job Execution via REST API

To make the video generation process accessible and scalable, a RESTful API endpoint will be provided. This allows users to initiate video creation by submitting a job request, along with the required client data and use case parameters.

Upon submission, the system returns a unique video ID and a password. These credentials can be used to poll the status of the video generation process and download the final output once it's complete. Polling was chosen as the preferred method for status tracking, as it aligns well with the asynchronous nature of video rendering and offers simplicity in client-side integration.

For security and access control, an API key is required for all interactions—including job initiation, status polling, and video retrieval. This ensures that only authorized users can utilize the system and access generated content.

2.2.8 Experimentation and Iterative Evaluation

Each step in the pipeline was validated through continuous testing. After every development milestone, feedback was gathered from mentors and company stakeholders to evaluate the output. This cycle was repeated across areas such as branding, prompt design, video quality, and timing—ensuring alignment with stakeholder expectations and improving the system iteratively.

3 Conclusion

This research demonstrates that it is technically feasible to create a fully automated video editing pipeline using Python and open-source libraries. The system not only supports video generation and editing but also provides an accessible API interface, allowing users to initiate video creation tasks, track their progress, and retrieve finished videos. While commercial tools offer valuable functionality for manual workflows, achieving full automation with contextual and audio-driven adaptability requires a custom-built solution integrating multiple technologies.