Q1)

Minimal Viable Version (MVV)

Goal

Build a prototype that answers natural language questions about the video and return visual + textual context as a response from a video demo of a SaaS product.

Step-by-Step Approach

1. Initial User Interaction & Problem Discovery

- I would begin by talking to the user or stakeholder to understand the current process of documenting or explaining SaaS flows.
- From this, I would identify pain points like: manual screenshotting, inconsistent documentation, or hard-to-search video content.

2. Define Scope Based on User Goals

- Set the goal: Build a system where a user can upload a video + question, and get a clear, visual step-by-step answer.
- Keep it narrow for the first version: don't solve for all UI detection or detailed scene understanding yet.

3. Technical Steps for MVV

Usually, the simplest solution is the best solution and to build an MVV, I focussed on the simple solution to solve the user problem

Step 1: Extract Audio + Transcript

- Convert the video to a WAV file using ffmpeg.
- Use a Whisper model (e.g., whisper-tiny or medium) to transcribe the audio.
- Segment the transcript based on time windows or sentence boundaries (~7–10 sec).

Step 2: Extract Screenshots

- Use OpenCV to take screenshots at the start of each transcript segment.
- This creates a basic visual-to-text pairing.

Step 3: Construct Input Context

- Pair each segment of text with its corresponding screenshot.
- This creates a multimodal structure for an LLM input prompt.

Step 4: Generate Response

- Use an LLM to process the question along and return a textual answer with step-by-step explanation, and reference each screenshot.

Q2)

Ideal (Polished) Solution

After MVV: Return to the User

1. User Walkthrough Sessions

- Sit with 2–3 actual users and watch them use the MVV on a real SaaS video.
- Prompt them with open-ended questions:
  - "Was this answer clear?"
  - "Do these screenshots match your mental model of what's happening?"
  - "Would you feel confident using this in documentation?"

2. Feedback Collection and Prioritization

- Note issues: e.g., screenshots miss the right UI state, or transcript feels too general.
- Categorize feedback:
  - Quick Fixes: better segment timing, clearer formatting
  - Deep Enhancements: smarter screenshot selection, UI element detection
- Prioritize based on impact and effort.

3. Design-Driven Refinement

- Translate feedback into improved UX:
  - Add user controls for adjusting screenshots per segment
  - Let users highlight or annotate screenshots if labels are unclear
- Conduct short usability tests after every major iteration

Implemented Components

I am building a system inspired by Anthropic's *Computer Use* feature—aimed at interpreting user interactions with software through visual and contextual cues.

- **Scene Detection**:

Using the scenedetect library, I segment the video based on perceptible scene changes rather than fixed time intervals, improving the precision of screenshot capture.

- **UI Element Detection**:

  Leveraging Facebook's DETR and Microsoft's LayoutLM via Hugging Face, I extract and label on-screen UI components (buttons, dropdowns, text inputs, etc.) to build structured context for each screenshot.

- **Transcript Generation**:

  Using OpenAI Whisper, I transcribe the audio and align it with segmented video to create timestamped, meaningful chunks of user interaction.

- **Multimodal Fusion**:

  Each video segment is tied to its transcript, UI elements, and screenshot. This is formatted and passed into an LLM (Mistral or GPT) to answer questions or narrate what's happening in the demo.

Ideal System Breakdown

1. Smarter Screenshotting

- Incorporate **action-aware scene detection**:
  - Histogram/SSIM-based visual changes
  - Detect UI interactions like clicks or modal openings
- Capture frames only when meaningful changes occur

2. Advanced UI Understanding

- Run models like DETR for UI element detection and use OCR to extract and label visible text
- Annotate screenshots with bounding boxes and semantic labels (e.g., "Clicked 'Save'")

3. Multimodal Pipeline Enhancements

- Structure LLM input with:
  - Transcript segments
  - UI element metadata
  - Annotated screenshots
- Prompt LLM to infer actions, e.g., "What is the user doing in this segment?"

4. Enhanced Answer Generation

- Generate detailed walkthroughs with:

    - Sequential steps
    - Annotated screenshots (with red boxes and captions)
    - Natural language explanations (e.g., “User navigates to Settings and enables Dark Mode”)

5. UX Considerations

- Build a basic web interface for:
    - Uploading videos
    - Asking questions
    - Viewing answers inline with screenshots and timeline
- Add option to edit/correct screenshots or segment mappings (feedback loop)

6. Deployment Plan: Scale It, Maintain It

Backend**:** Use FastAPI or Flask to expose endpoints:

Containerization**:** Package components as Docker containers:

- Transcription container (Whisper)
- Vision container (CV + UI detection)
- LLM container (Mistral or OpenAI call wrapper)

Cloud Deployment:

- Host on GCP or AWS (ECS/EKS)
- Store video assets temporarily in S3 or GCS with lifecycle rules
- Use a queue-based system (e.g., Celery + Redis) to process videos asynchronously for better scalability

Frontend Deployment

- Use Vercel or Netlify for the frontend (Next.js or React)
- Integrate Auth (e.g., Firebase, Auth0) if required