AI-Powered Short-Form Video Automation System

Technical proposal · Roberto Aguirre · Registered Partner, Anthropic

> I already run a system that does most of what you describe: topic in, vertical video out, with a generated presenter, dynamic footage, burned captions and automated publishing. It runs on a schedule and produces the avatar led videos I publish on LinkedIn. So my proposal is not an estimate of how long it would take me to learn this. It is an adaptation of something that works, which is why a complete system fits inside your stated budget and lands in two weeks. Below: the architecture, what I reuse, and what genuinely has to be built for your case.

1. What already exists, and what I would build

CAPABILITY	STATUS	DETAIL
Vertical 1080x1920 composition	Built	Single FFmpeg filtergraph per stage. A golden test renders a real MP4 and asserts output dimensions and duration on every commit.
Generated presenter (avatar)	Built	HeyGen API v2, fully automated. Replaced the manual dashboard step. Includes automated identity QA on the rendered face.
Script generation	Built	Brief to script, with structural validators that reject a script before it reaches TTS.
Voiceover and voice QA	Built	Segment based generation with context stitching, per segment QA and selective regeneration of only the defective take.
Automatic subtitles	Built	ASS subtitles burned in the same pass as the overlays, not a second encode.
Review and approval gates	Built	Seven state machine with validated transitions. Human approval points before publishing.
Automated publishing	Partial	YouTube and Instagram Reels are working, including the Instagram requirement of serving the file from a public URL. TikTok is not built.
Swappable LLM provider	Built	A routing layer maps task class to provider and model, resolved by environment then per tenant then global default. Changing provider is configuration, not code.
Encrypted credential storage	Built	AES-256-GCM vault per tenant, with self serve OAuth so the client connects their own accounts.
Docker deployment	Built	Video runner image with the base pinned by digest, so the render environment is reproducible.
TikTok publishing	To build	Content Posting API, including the audit process TikTok requires before an app can post publicly.
Local TTS	To build	Today the pipeline is wired to a paid provider. The surface to abstract is small: two functions call it, everything downstream is provider agnostic.
Local LLM runtime	To build	The routing layer exists and is in production; adding a local provider means implementing one adapter behind an interface that already has consumers.
Korean output	To build	Spanish and English are in production with per language QA profiles. Korean needs its own voice selection and QA profile.

2. Worked example: long form source to finished vertical video

A concrete run of the pipeline, end to end, so you can judge the output rather than the description.

STAGE	WHAT HAPPENED
Input	A third party conference talk, given as a plain URL. Nothing else. `youtu.be/jji0ycFJQW0` · *Inteligencia artificial: el riesgo del sedentarismo cognitivo*, TEDx
Agents	Transcribed the talk, extracted the central argument, and wrote a short form script around that single idea rather than summarising the whole talk.

Generation	Voiceover, generated illustration clips, motion, transitions and burned captions, composed into a vertical video.
Output	A finished vertical video, published. `youtu.be/9lJZ6gQcQqU`

The part worth noticing: nothing was clipped from the source

The output does not contain a single frame, second of audio or still image from the original talk. The agents took the *idea* and the pipeline generated entirely new footage around it. That distinction matters more for your project than it might look at first:

- **It answers your dynamic video requirement directly.** The visuals are generated clips with motion, not stock images panning across the screen, because there was no source footage to lean on in the first place.
- **It removes a risk you did not list but will hit.** A system that publishes automatically to three platforms, unattended and on a schedule, is a system that can collect copyright strikes at machine speed. Clip based automation fails here eventually. Generating from the idea does not.

The same mechanism works from any starting point: a topic, a keyword, a competitor video, a document. The source shapes the argument; the footage is always generated.

3. How the video is actually dynamic, and not a slideshow

This is the part of your brief I would answer first, because it is where most proposals fail. Movement in my pipeline comes from four independent sources, composed in one graph rather than stitched afterwards:

- **A generated presenter that speaks.** The avatar is rendered from the script, so the base layer is real motion with lip sync, not a still image.
- **Generated video clips as B roll**, cut to the beats of the voiceover.
- **Compositing with alpha**, not cuts: each clip is masked, faded in and out on its own alpha channel, and positioned at its beat.
- **Continuous camera motion** where the footage is static, plus cross dissolves between segments.

The core of the overlay stage, as it runs today:

```
[1:v]format=gray,split=N[m0][m1]...            # rounded alpha mask, one per clip
[2:v]format=rgba[f0];[f0][m0]alphamerge[am0];   # clip carries its own alpha
[am0]fade=t=in:st=0:d=0.2:alpha=1,
     fade=t=out:st=DUR-0.2:d=0.2:alpha=1,
     setpts=PTS+START/TB[ov0]                   # placed on its beat
[base][ov0]overlay=X:Y:enable='between(t,START,END)'
[bvid]subtitles=captions.ass:fontsdir=...[v]    # burned in the same pass
```

Segment transitions use `xfade` with a matching `acrossfade` on audio, and the final assembly is a filter based `concat` so every segment is normalised to the same fps, scale and pixel format before joining. Where footage would otherwise sit still, a slow `zoompan` push keeps the frame alive.

4. Local versus paid, and what that costs

Your brief asks for local models to control cost, and also for genuinely dynamic video. Those two goals pull in opposite directions, so I would split them rather than promise everything local.

STAGE	RECOMMENDATION	WHY
Script and metadata	**Local, viable today**	A 7B to 14B instruct model on Ollama handles scripts, titles, descriptions and hashtags well. This is the largest share of LLM calls and the easiest win.

Text to speech	**Local first, paid fallback**	Local TTS is good enough for many voices and languages, and it is where recurring cost accumulates fastest. I would put it behind a provider interface and let quality decide per language.
Composition, subtitles, audio mix	**Local, no API at all**	FFmpeg. This is already zero marginal cost in my current system.
Generated video clips	**Paid API, by default**	Local video generation needs serious GPU and still trails hosted models on reliability. This is the one place I would not force local.
Presenter avatar	**Paid API**	Only if you want a talking presenter. Skipping it removes a recurring cost and a dependency.

On cost figures. I would rather give you a measured number than a confident guess. Per video cost depends on clip seconds generated, voice length and whether an avatar is used, and provider pricing moves. In milestone 1 I will instrument the pipeline to report actual cost per render, so you get real numbers from your own runs instead of an estimate from mine.

5. Architecture

```
topic  →  script (LLM, swappable)  →  language profile
       →  voiceover (TTS, swappable)  →  automated voice QA  ⟶ regenerate bad segment
       →  visual plan  →  clips (video API) + optional avatar
       →  FFmpeg composition (overlays, captions, music, transitions)
       →  automated QA (duration, aspect, identity)  →  human approval
       →  per platform metadata  →  publish (YouTube / Instagram / TikTok)
```

Four decisions I would carry over from the running system, because each one came from a real failure:

- **Every job is a directory with a state file.** A run can be resumed, inspected or replayed. No hidden state in a queue.
- **QA is a stage, not an afterthought.** Voice defects are caught per segment and only that segment is regenerated, which is far cheaper than rerolling a full take.
- **Publishing is idempotent.** The video id is persisted before thumbnail, playlist and comment steps, so a failure in any of those never causes a duplicate upload.
- **Providers sit behind interfaces**, with keys, models and languages in configuration and secrets encrypted at rest.

6. Hardware

SETUP	SPECIFICATION	FITS
Minimum	8 cores, 16 GB RAM, no GPU	Composition and publishing with cloud LLM, cloud TTS and cloud clips. FFmpeg encoding is CPU bound and this is enough for short vertical output.
Recommended	12 cores, 32 GB RAM, GPU with 12 to 16 GB VRAM	Local LLM for scripts and local TTS, with clips still from a paid API. This is the configuration that actually reduces your bill.
Full local ambition	24 GB VRAM or more	Local video generation. I would treat this as a later experiment measured against the hosted option, not as the launch configuration.

7. Scope and price

Your budget buys a system that runs, not a fraction of one. That is only possible because I am adapting something that already works rather than building it from zero.

Phase 1 · Working system · USD 1,000 · 2 weeks

DELIVERED	DETAIL
End to end pipeline	Topic in, vertical 1080x1920 video out, with script, voiceover, burned captions, background music and transitions.
Genuinely dynamic visuals	Generated clips cut to the voice beats, alpha compositing, continuous camera motion. Not a slideshow, as described in section 3.
Portable install	Docker setup, configuration file for providers, models and languages, encrypted credential storage. Runs on your machine or your server.
Provider switching	LLM, TTS and video generation behind interfaces. Changing provider is a configuration change, not a code change.
English output	Full pipeline in English, with the procedure to add a language documented.
YouTube publishing	Automated upload with title, description and hashtags, and a review step before anything goes public.
Cost instrumentation	Every render reports its actual cost, so the local versus paid decision in phase 2 is made on your numbers, not on my estimate.

Why two weeks and not two months. I am not building this from zero. I run a system in production that already does the composition, the voice QA, the state machine and the publishing, and phase 1 is adapting it to your requirements and packaging it to install cleanly elsewhere. That reuse is the entire reason this price works.

Documentation, install instructions for a second machine, the provider replacement guide and a two week bug fixing window are included, not sold separately.

What phase 1 does not include, and why

I would rather be explicit about the boundary than discover it mid project. These four items are in your brief and are not in phase 1. Each is genuinely independent work, and each is better decided once you have the core running and can see what you actually need:

NOT INCLUDED	WHY IT IS SEPARATE
Local LLM and local TTS	The interfaces ship in phase 1, so the switch is already possible. What takes real time is validating that local quality is acceptable for your content and language, and that is a measurement, not a feature. Better done with cost data from your own runs.
Korean	Adding a language is not a translation setting. It needs its own voice selection and its own QA profile, because what counts as a bad take differs per language.
TikTok publishing	Their Content Posting API requires an app audit that TikTok runs on their own schedule. I will not put a date on something I do not control, and I will not bundle it into a two week commitment.
Instagram Reels and scheduling	Instagram will not accept a binary upload and requires the file served from a public URL, which means hosting decisions that belong to you. Scheduling is straightforward once one platform is proven end to end.

I have built all four before, so none of this is unknown territory. Quoting them separately when you want them is simply more honest than folding them into a number today and discovering the timeline later.

8. Two things I would push back on

Fully local is the wrong target on day one. Local LLM and local TTS pay for themselves quickly. Local video generation does not yet, and forcing it is how you end up with the slideshow you explicitly do not want. I would ship with hosted clips, measure real cost per video, and revisit with data.

Publishing to three platforms is not one integration. Instagram requires the file to be served from a public URL and will not take a binary upload. TikTok requires an application audit before an app may post publicly. Treating these as one line item is the most common way this kind of project slips, so I have separated them and put the audit early.

Examples of the automated avatar led output are on my LinkedIn profile, produced by the pipeline described above. Happy to walk through the composition stage or the state machine on a call, or to run a sample topic of yours through the current system so you can judge the output before committing to anything.