

Rupam Creator Studio for AI Assisted Content Production and Multilingual Captioning

DR. T. CYNTHIA ANBUSELVI, KOTA SANTHOSH, A.N.TEJA, KEERTHI A.S, KRISHTIPATI PRAHARSHITHA, ANUPRIYA . M

Department of Computer Science and Engineering (Artificial Intelligence and Machine Learning)
East Point College of Engineering and Technology, Bengaluru, India

Abstract

Short video production requires creators to move between topic research, writing, visual generation and caption editing. Rupam Creator Studio connects these activities through a web application with reusable creator context and editable intermediate outputs. The system combines public-source research, structured script generation, thumbnail and avatar service integrations, and a multilingual caption editor. Its caption pipeline separates speech recognition, acoustic alignment, text revision and rendering so that design changes do not require a new transcription request. A persistent local worker processes caption jobs, while Supabase supports authenticated asset storage and library records. This paper describes the implemented architecture, data flow and failure handling, and reports a local regression run in which 29 software checks passed. The result supports specific functional properties, including source-link validation, revision handling and job recovery; it does not establish transcription accuracy or creative effectiveness. The contribution is an integrated, human-reviewed production workflow rather than a new generative model.

Index Terms—generative AI, creator tools, script generation, speech recognition, multilingual captions, media workflows.

I. INTRODUCTION

A creator may research a topic in a browser, draft a script in a text generator, prepare visual assets in separate services and finish captions in a video editor. Each transfer can lose context: a revised hook may not reach the thumbnail prompt, while changes to a transcript may become disconnected from timing and style. A connected workspace can retain these intermediate decisions and make them available at the next stage.

Rupam addresses this coordination problem through a shared creator profile, an active script and a media library. Users can supply previous writing samples, inspect retrieved sources, edit draft fields and send relevant content into asset-generation screens. Video captioning remains a separate editable process because wording, language representation, timing and appearance require different forms of review.

The project has three practical objectives: retain creator context across writing tasks, expose evidence and failures during service-assisted generation, and keep caption edits

reproducible during preview and export. It does not automatically publish content or infer that a retrieved topic will perform well. Users choose the final wording and media they wish to export.

A Problem definition

The central problem is continuity across a sequence of creative decisions. A creator profile describes the intended audience and tone; a research collection supplies evidence; a script organizes the narrative; visual assets communicate the chosen idea; and captions expose the spoken content during playback. When these representations are handled independently, the creator must repeatedly copy information and reconcile changes. Rupam maintains explicit intermediate objects so that a user can inspect the result of each stage before starting another.

The system is intended for an individual creator or a small production workflow using a persistent application host. It supports interactive writing and media preparation without requiring the user to operate every provider through its own interface. The design does not assume that all stages must run for every project. A creator may write a script without producing an avatar, or upload an existing video directly to Caption Studio. This independence is necessary because creative work often starts from material that already exists.

B Scope and contributions

The first contribution is the connection between creator context, retrieved evidence and editable script fields. The second is the separation of caption text, word identity, timing and visual design. The third is explicit lifecycle management for asynchronous media jobs, including ownership checks and recoverable storage failures. These are application-level engineering contributions. The project uses existing language, image, video and speech models; it does not claim a newly trained foundation model or a measured advantage over commercial creator platforms.

II. RELATED WORK

A Language models and creator context

Vaswani et al. introduced the Transformer, an attention-based architecture for sequence processing [1]. Brown et al. subsequently showed that a large language model can perform varied tasks from textual instructions and examples without

task-specific parameter updates [2]. Together, these studies provide background for supplying a creator profile and previous writing examples as context. Rupam uses provider-hosted models and reusable writing rules; it neither trains a Transformer nor fine-tunes a model on the uploaded scripts. This distinction matters because a saved writing pattern changes the request context, not the underlying model weights.

Ouyang et al. investigated model training with human demonstrations and preference feedback to improve instruction following [3]. Their work highlights that useful output depends on alignment with the requested task rather than model size alone. Wei et al. studied prompting with intermediate reasoning examples on reasoning benchmarks [4]. These studies motivate explicit instructions and task decomposition, but their findings are not evidence that a creator application will automatically produce accurate scripts. Rupam therefore validates response structure and leaves the final narrative editable instead of treating a generated response as an approved script.

B Retrieval and tool-assisted generation

Lewis et al. combined retrieval with language generation for knowledge-intensive tasks [5]. Retrieved passages offer a route to external evidence that is not limited to a model's stored parameters. Rupam follows the application-level principle of providing source excerpts to a writer. Its implementation uses public web search, identifiers and retained research collections rather than the learned dense retriever described in that paper. Search coverage and source reliability must still be assessed for each topic; attaching a link does not establish that every generated claim follows from it.

Yao et al. presented ReAct, which interleaves reasoning and actions that interact with external information or environments [6]. Schick et al. proposed Toolformer, in which a language model learns how to use external APIs [7]. Both address the relationship between language generation and tool execution. Rupam uses explicit application routes to choose services, validate requests and retain job state. It does not implement either paper's complete agent-training or prompting method. The relevant design lesson is to preserve the boundary between a language response, a tool call and a verified result.

C Revision and factual support

Madaan et al. proposed Self-Refine, where a model produces feedback and revises an initial answer iteratively [8]. This supports examining generation as a revisable process rather than a single final response. Rupam's refinement workflow is user-directed: it consumes the current draft, its research and a bounded conversation history. Manual corrections remain part of the input. Unlike autonomous self-feedback, this process lets the creator specify what should change and retain fields that are already satisfactory.

Maynez et al. examined unfaithful content in abstractive summarization [9]. Kryściński et al. developed a factual-consistency evaluation approach that also identifies supporting or inconsistent spans [10]. These studies show why fluent output and factual support need separate assessment. Their experiments concern summarization rather than this studio, so their reported results cannot be transferred directly. They nevertheless motivate the project's retention of evidence and its distinction between source-address validation and claim verification. A valid URL is a provenance constraint, not a factuality score.

D Speech recognition and script conversion

Baevski et al. introduced wav2vec 2.0, a method for learning speech representations from unlabeled audio before supervised adaptation [11]. Radford et al. explored large-scale weak supervision for multilingual speech recognition in Whisper [12]. These approaches illustrate different ways of learning from speech data. Rupam includes a local Whisper path and an external speech-recognition service; wav2vec 2.0 is discussed as research context, not an implemented dependency. Recognition quality in either route still depends on the recording and language, and must be evaluated using reference transcripts.

Hermjakob et al. presented uroman for converting text from multiple writing systems to a Latin-script representation [13]. Its purpose differs from speech recognition and semantic translation. That distinction informs the caption document: native text, Romanized display text and timing should remain separate. Rupam retains word identifiers and intervals when changing display representation, while contextual Romanization is checked for complete and valid identifier mappings. The project does not claim that a mechanically or contextually produced spelling is always the preferred form for a particular audience.

E Visual generation and speech-driven avatars

Rombach et al. introduced latent diffusion models for high-resolution image synthesis, performing diffusion in a learned latent space [14]. This is relevant background for conditional image creation, but it does not identify the undisclosed internals of the studio's thumbnail provider. Prajwal et al. addressed speech-to-lip generation with a lip-sync expert in Wav2Lip [15]. Their work establishes lip synchronization as an explicit technical problem requiring its own evaluation. Rupam's avatar path uses a ComfyUI workflow and an InfiniteTalk-based backend; it does not claim to implement Wav2Lip or inherit its reported performance.

F Relationship to the proposed system

The reviewed studies address language modeling, retrieval, refinement, factuality, speech and visual synthesis at different levels. Rupam connects these capabilities through editable

application objects and durable job handling. Its contribution is the continuity between research, script, generated media and captions, together with explicit review points. Evaluation must therefore cover both software correctness and the quality of individual outputs. The local tests reported in this paper address the former; they do not establish a comparative advantage over the reviewed models or systems.

III. METHODOLOGY

A System architecture

Figure 1 summarizes the implementation as a browser workspace, a server coordination layer and three processing paths. Storage is distributed deliberately: writing state is cached in the browser and synchronized for signed-in users, caption projects and jobs use persistent local disk, and authenticated generated assets are archived in Supabase. These stores have different lifetimes and should not be treated as one fully synchronized database.

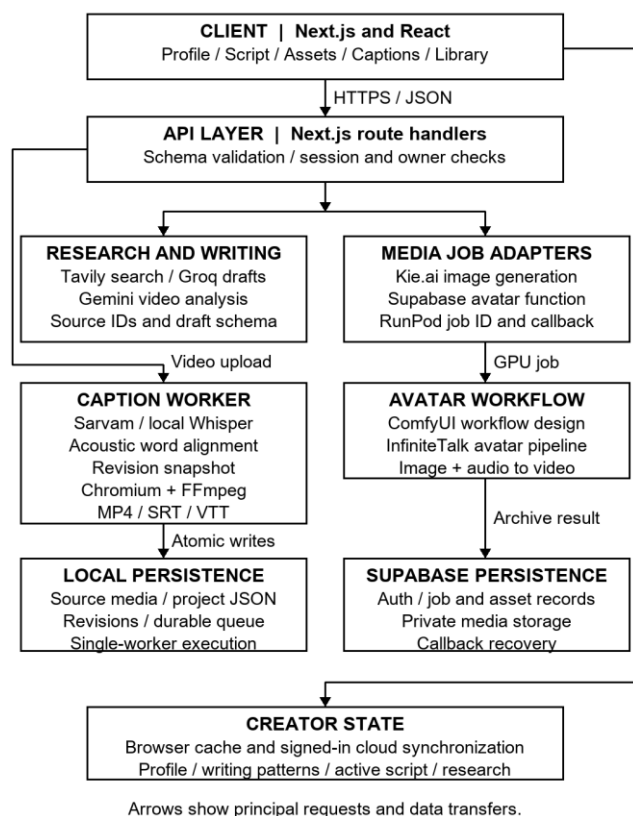


Fig. 1. Technical architecture of Rupam Creator Studio with API, processing and persistence components.

Workspace and server routes

The application uses Next.js 15 and React 18. Its studio screens expose Profile, Script, Thumbnail, Avatar, Frames, Caption Studio and Library. Server routes validate requests and call specialized services. The active script can be reused by Thumbnail and Avatar, while exported media is represented by

library records. The newer Caption Studio provides a richer editing workflow alongside the earlier Frames caption route.

Research and media services

Tavily supplies public search results; Groq supplies structured script responses; Gemini supports optional inspection of public creator videos. Thumbnail requests use a Kie.ai image-generation job. Avatar requests pass through an authenticated Supabase function to a RunPod endpoint. These are service integrations, so credentials, provider availability and job completion remain external dependencies.

Caption execution and storage

Caption Studio uses a dedicated worker with file-backed projects, media and queued jobs. Sarvam is the configured cloud transcription path and local Whisper is selectable. Chromium renders the caption composition and FFmpeg produces the video output. Supabase private storage requires authorized access or time-limited links [16]; the application also checks project ownership. The local queue currently requires the worker and web server to share the same persistent host storage.

Authentication and persistence are related but separate concerns. A signed-in user must own a project or job before receiving its status or media. A completed provider job must also be copied into durable storage before the application can rely on it for later playback. The Supabase avatar handler therefore separates provider completion from archival completion. If the provider succeeds but storage fails, recovery can repeat the archival operation without submitting a second generation job.

B Module design and processing

Creator profile and writing memory

The creator records a niche, audience and communication preferences in Profile. At least two previous scripts can be supplied as text or supported text-based files. The learning action produces a reusable writing pattern with rules, a proposed structure and evidence from the samples. This is prompt context rather than model fine-tuning. Long samples are represented by opening and closing excerpts, so the learned pattern may omit characteristics found only in the middle of a script.

Patterns are cached in the browser, synchronized with Supabase while signed in, and can be exported or imported as JSON. Selecting an alternative style creates a separate reusable pattern. This keeps an experimental style from silently replacing the creator's learned preferences.

Research and script generation

A research request uses the topic and creator profile to query four lanes: the general web, Reddit, X and YouTube. Each lane

requests up to four results within the previous month. Results are deduplicated and retained with source identifiers, URLs and excerpts. A failed lane is reported separately, allowing accessible evidence to remain available. The resulting sample is not an exhaustive trend analysis.

The writer receives the profile, selected writing pattern and compact research excerpts. It returns structured fields for the title, angle, hook, sections, call to action and claim-source pairs. Zod validation checks the response shape, and a URL membership check rejects citations outside the retrieved set. This restricts fabricated source addresses but cannot prove that a source supports each sentence. Users still need to inspect the claim and cited passage.

Direct edits remain part of the active draft. The refinement action uses that draft, its original research and a bounded conversation history. Browsing a different topic does not replace the evidence associated with an existing script. Model timeouts, invalid JSON and rate limits are surfaced or handled through bounded retries rather than an unlimited generation loop.

Creator analysis and visual assets

Optional creator-video analysis asks Gemini to inspect the opening 90 seconds of a public YouTube video with one-frame-per-second sampling. It returns timestamped observations and a paraphrased summary, not a copied full transcript. Rapid edits can be missed. With a configured YouTube API key, the application can compare a video's views against a recent channel median, requiring at least five baseline videos and a nonzero median. A ratio of at least three marks an outlier candidate; upload age, promotion and format are not normalized.

Thumbnail generation submits a prompt and optional reference URLs to an asynchronous image service, then polls for completion. The avatar path creates a remote video job and supports callback-based completion. Completed authenticated assets are archived before the corresponding job is marked complete. The library provides a place to retain generated outputs; it does not itself demonstrate that a thumbnail or avatar increases engagement.

ComfyUI avatar generation

ComfyUI is used to design the avatar-generation workflow. Its node graph represents connected processing operations and their inputs and outputs [17]. In this project, the avatar workflow forms part of an InfiniteTalk-based image-and-audio-to-video process: a reference image defines the visible character, an audio input supplies the speech, and the generated video is returned as an asynchronous result. ComfyUI describes the generation workflow, while RunPod supplies the remote job execution interface.

The web application sends the prompt, resolution and optional image and audio payloads through an authenticated Supabase function. The function records the job identifier and handles completion callbacks or status recovery. The completed video is copied to private storage before its Library record is finalized. Workflow-node configuration and model checkpoints belong to the generation backend; they are separate from the web application and are not evaluated by the local JavaScript regression tests.

Speech recognition and alignment

Caption processing begins with a video upload. Audio is prepared for recognition, and the cloud path splits it into short sections with offsets derived from measured PCM duration. Phrase-level recognition timestamps are not assumed to be individual word boundaries. Sarvam documents chunk timestamps at sentence or phrase granularity [18]. The project therefore aligns the retained transcript against audio with stable-ts and a local multilingual faster-whisper model.

Each word has a stable identifier and timing interval. Native text, Romanized text and source information are retained separately. Contextual Romanization validates identifier mappings and preserves existing English text and timings. Words that cannot be resolved individually may share an acoustic interval and receive a review flag. Such flags identify uncertainty rather than hiding it behind artificially precise boundaries. Manual text edits retain intervals and do not automatically realign the audio.

Editing and reproducible export

The editor supports transcript changes, phrase split and merge, undo and redo, timing adjustments, overlays and selected-line or whole-video styling. Twenty-four templates are available. Template and language-display changes reuse the transcript rather than issuing another recognition request. Figure 2 illustrates native text editing, and Fig. 3 shows the signature-template interface.

Caption visibility uses start-inclusive and end-exclusive intervals: a word is visible when its start time is less than or equal to playback time and playback time is less than its end. Both preview and export evaluate the shared HTML composition from playback time, which avoids a separate animation clock. Export snapshots a document revision and renders at 30 frames per second with a maximum long dimension of 1280 pixels. SRT and VTT export speech cues without the visual styles or independent overlays.

Optimistic revision checks detect stale writes. Atomic file replacement and process-aware locks protect local project updates. A single worker processes durable jobs, with bounded recovery of interrupted work. Stable export identifiers support idempotent library registration. These mechanisms address consistency and recovery on a dedicated host; distributed

Feature	Rupam	Descript	VEED
ComfyUI workflow	I	NV	NV
Cloud / local speech	I	NV	NV
Export snapshots and recovery	I	NV	NV

I = implemented; D = documented; NV = not verified in the reviewed sources, not absent. Availability may depend on plan or region. Sources reviewed on 10 October 2026.

	Rupam	Descript	VEED
AI script drafting	I	D	D
AI image generation	I	D	D
AI avatar generation	I	D	D
Automatic captions	I	D	D
Caption styles	I	D	D
Text overlays	I	D	D
Writing patterns from samples	I	NV	NV
Four-source research and URL checks	I	NV	NV
Native / Romanized word IDs	I	NV	NV
ComfyUI workflow	I	NV	NV
Cloud / local speech	I	NV	NV
Export snapshots and recovery	I	NV	NV

I: implemented D: documented NV: not verified

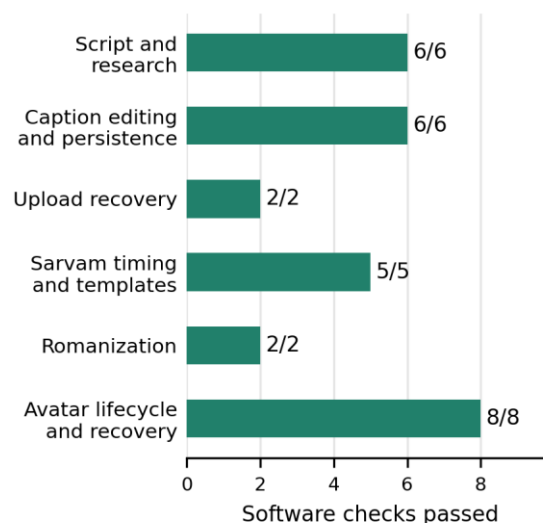
Fig. 4. Selected feature comparison. Not verified indicates an evidence gap, not feature absence.

B Software regression results

Figure 5 summarizes the recorded regression run of 26 September 2026: 29 checks passed across six test files, with no failures or skipped checks. The 100% pass rate applies to these controlled software checks, not transcription accuracy, avatar quality or competitor performance.

29/29 passed | 0 failed | 0 skipped

100% pass rate in the recorded test suite



Run: 26 September 2026. Not model accuracy.

Fig. 5. Software checks passed by test area. No comparative output-quality benchmark was performed.

V. CONCLUSION

Rupam Creator Studio integrates research-assisted script writing, thumbnail generation, ComfyUI-based avatar preparation and multilingual caption editing within one creator workspace. Shared context and editable intermediate outputs connect the stages, while revision snapshots, ownership checks and recoverable job handling support dependable processing. All 29 local regression checks passed, providing evidence for the exercised software contracts rather than for model accuracy or audience impact. The contribution is the coordinated application workflow built around existing AI services. Future work should evaluate transcription and timing against human references, assess avatar synchronization and creator usability, and extend job coordination across multiple hosts.

VI. REFERENCES

- [1] A. Vaswani et al., "Attention Is All You Need," arXiv:1706.03762, 2017. doi: 10.48550/arXiv.1706.03762.
- [2] T. B. Brown et al., "Language Models are Few-Shot Learners," arXiv:2005.14165, 2020. doi: 10.48550/arXiv.2005.14165.
- [3] L. Ouyang et al., "Training language models to follow instructions with human feedback," arXiv:2203.02155, 2022. doi: 10.48550/arXiv.2203.02155.
- [4] J. Wei et al., "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," arXiv:2201.11903, 2022. doi: 10.48550/arXiv.2201.11903.
- [5] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," arXiv:2005.11401, 2020. doi: 10.48550/arXiv.2005.11401.

- [6] S. Yao et al., “ReAct: Synergizing Reasoning and Acting in Language Models,” International Conference on Learning Representations, 2023. <https://arxiv.org/abs/2210.03629>
- [7] T. Schick et al., “Toolformer: Language Models Can Teach Themselves to Use Tools,” arXiv:2302.04761, 2023. doi: 10.48550/arXiv.2302.04761.
- [8] A. Madaan et al., “Self-Refine: Iterative Refinement with Self-Feedback,” arXiv:2303.17651, 2023. doi: 10.48550/arXiv.2303.17651.
- [9] J. Maynez, S. Narayan, B. Bohnet, and R. McDonald, “On Faithfulness and Factuality in Abstractive Summarization,” arXiv:2005.00661, 2020. doi: 10.48550/arXiv.2005.00661.
- [10] W. Kryściński, B. McCann, C. Xiong, and R. Socher, “Evaluating the Factual Consistency of Abstractive Text Summarization,” arXiv:1910.12840, 2019. doi: 10.48550/arXiv.1910.12840.
- [11] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, “wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations,” arXiv:2006.11477, 2020. doi: 10.48550/arXiv.2006.11477.
- [12] A. Radford et al., “Robust Speech Recognition via Large-Scale Weak Supervision,” arXiv:2212.04356, 2022. doi: 10.48550/arXiv.2212.04356.
- [13] U. Hermjakob, J. May, and K. Knight, “Out-of-the-box Universal Romanization Tool uroman,” Proceedings of ACL 2018, System Demonstrations, pp. 13–18, 2018. doi: 10.18653/v1/P18-4003.
- [14] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-Resolution Image Synthesis with Latent Diffusion Models,” arXiv:2112.10752, 2021. doi: 10.48550/arXiv.2112.10752.
- [15] K. R. Prajwal, R. Mukhopadhyay, V. P. Nambodiri, and C. V. Jawahar, “A Lip Sync Expert Is All You Need for Speech to Lip Generation In The Wild,” arXiv:2008.10010, 2020. doi: 10.48550/arXiv.2008.10010.
- [16] Supabase, “Storage Buckets,” documentation, accessed Sep. 26, 2026. <https://supabase.com/docs/guides/storage/buckets/fundamentals>
- [17] Comfy, “The ComfyUI interface,” documentation, accessed Sep. 26, 2026. <https://support.comfy.org/articles/7675456845-the-comfyui-interface>
- [18] Sarvam AI, “Speech to Text REST API,” documentation, accessed Sep. 26, 2026. <https://docs.sarvam.ai/api-reference/speech-to-text/transcribe>
- [19] Descript, “Features,” product documentation, accessed Oct. 10, 2026. <https://www.descript.com/features>
- [20] Descript, “Video Caption Generator,” product documentation, accessed Oct. 10, 2026. <https://www.descript.com/tools/video-caption-generator>
- [21] Descript, “Add Text to Video,” product documentation, accessed Oct. 10, 2026. <https://www.descript.com/tools/add-text-to-video>
- [22] VEED, “Video Editing Tools,” product documentation, accessed Oct. 10, 2026. <https://www.veed.io/tools>
- [23] VEED, “How to Create a New Project in VEED,” Help Center, accessed Oct. 10, 2026. <https://support.veed.io/en/articles/16413135-how-to-create-a-new-project-in-veed>
- [24] VEED, “How VEED's dashboard works,” Help Center, accessed Oct. 10, 2026. <https://support.veed.io/en/articles/11641509-how-veed-s-dashboard-works>