International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

TrendNest :Trend Based Content Generation Using AI Agents

Sudhanshu Jawanjal
Artificial Intelligence and Data Science Department
New Horizon Institute of Technology and Management
Thane, India
sudhanshujawanjal226@nhitm.ac.in

Harshada Kumbhar
Artificial Intelligence and Data Science Department
New Horizon Institute of Technology and Management
Thane, India
harshadakumbhar226@nhitm.ac.in

Sindhu Reddy
Artificial Intelligence and Data Science Department
New Horizon Institute of Technology and Management
Thane, India
sindhumogalreddy226@nhitm.ac.in

Mrs.Yojana Milind Dehankar
Assistant Professor
Artificial Intelligence and Data Science Department
New Horizon Institute of Technology and Management
Thane,India
yojanadehankar@nhitm.ac.in

Abstract— Building and maintaining a consistent online presence is harder than it looks. Between chasing trending topics, scripting videos, designing eye-catching thumbnails, and writing SEO-friendly blog posts, most independent creators simply run out of hours in the day. TrendNest AI was built to tackle this problem directly. It is a full-stack web platform that pulls live trend data from Google Trends and then uses a combination of Groq's Llama 3.3-70B and Google Gemini to automatically produce YouTube video scripts, thumbnail images, and structured blog posts—all from a single interface. Raw trend titles are often too vague to feed directly into a generation pipeline, so the system first enriches them with a short AI-generated summary before passing the context downstream. Finished assets are stored in PostgreSQL through Prisma ORM, and thumbnail images are offloaded to Cloudinary for fast CDN delivery. The whole thing runs on Next.js 15 with TypeScript and is locked behind NextAuth.js session authentication so that only logged-in users can trigger generation. In testing, script generation averaged under four seconds, blog articles came back fully structured with clean HTML previews, and thumbnails consistently matched the visual style requested. The net result is a platform that meaningfully cuts the time it takes a solo creator or small marketing team to go from "what's trending today" to publish-ready content.

Keywords— content generation, generative AI, large language models, Google Trends, YouTube automation, SEO, Next.js, Retrieval-Augmented Generation

I. INTRODUCTION

Anyone who has tried to run a YouTube channel or a content blog knows the grind. You need to find what people are actually searching for, write a compelling script, come up with visuals, and then somehow produce a well-optimised blog post about the same topic—ideally before the trend cools off. For individual creators, this cycle repeats every few days and quickly becomes unsustainable [1].

Recent progress in large language model (LLM) research has made it genuinely possible to automate large chunks of that workflow. The Groq inference API now makes models like Meta's Llama 3.3-70B fast enough for real interactive use, returning full article-length outputs in a few seconds [2]. At the same time, Google's Gemini models can produce high-quality images from a short text description, removing the need for a separate design tool [3]. The missing piece has always been a system that ties trend signals to these generation capabilities in one coherent flow.

TrendNest AI is that system. It continuously watches Google Trends for the India region, enriches the raw topic titles with a short Groq-generated summary, and then lets the user spin up a YouTube script, a branded thumbnail, or a full blog post with one click. Everything generated is saved to a PostgreSQL database so creators can build up a searchable content library over time.

The platform is built on Next.js 15 App Router with TypeScript throughout. Route handlers are wrapped with NextAuth.js server-session checks, so unauthenticated requests never reach the AI providers. The overall architecture is illustrated in Fig. 1.

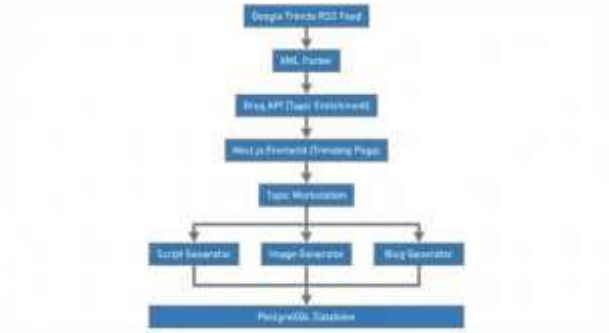

Fig. 1. Architecture of the proposed TrendNest AI system

II. LITERATURE REVIEW

Research in automated text generation goes back a long way, but early systems were mostly template-driven. They could fill in slots reliably, but the output felt mechanical and struggled badly with anything that required genuine fluency [4]. Transformer-based language models changed that significantly—models fine-tuned on large text corpora now produce output that is coherent, topically relevant, and stylistically consistent over long passages [5].

A number of teams have used LLMs to build automated blog and SEO article generators [6]. The typical approach is straightforward: supply a keyword, get back a structured article. These tools work reasonably well in isolation, but they all require the user to know what to write about in the first place. None of them watch trend signals or automatically adjust their output based on what is gaining traction at a given moment.

The YouTube side of content creation has its own body of research. Studies on viewer engagement patterns consistently show that the opening hook—roughly the first 20 to 30 seconds of a video—is the biggest factor in whether someone stays or clicks away [7]. Despite this, most automated script generators treat a script as a uniform block of text rather than a structured artifact with an explicit attention-grabbing opening, thematic sections, and a clear call-to-action at the end.

Thumbnail design has attracted attention from the computer vision community, with diffusion-based generators now capable of producing quite striking imagery from short prompts [8]. The difficulty is that these image tools are almost always freestanding. A creator still has to transfer context from their trend research to their image generator manually—there is no shared pipeline.

The use of Google Trends data as a content planning signal is well-documented in journalism and marketing literature [9]. The RSS feed in particular is a lightweight, freely accessible signal that updates in near real-time. What the literature does not address is how to make that raw trend data useful for downstream AI generation, since a bare topic title like "IPL 2025 auction" gives a language model very little to work with.

Putting all of this together, there is a clear gap: no published system combines live trend ingestion, context enrichment, and multi-format AI generation—script, image, and blog—inside a single authenticated product [10]. TrendForge AI addresses exactly that gap.

Table 1: Summary of Findings and Limitations in Existing Content Generation Systems

Reference	Key Findings	Limitations
[4]	Template-based generation for structured articles	Mechanical output; no trend integration
[5]	LLM-driven long-form text with coherence	No multi-format pipeline or image support
[7]	Hook structure strongly predicts viewer retention	No automated generation; no trend alignment
[8]	Diffusion models produce high-CTR thumbnails	Standalone tool; disconnected from text pipeline
[9]	Google Trends provides real-time content signals	No generation layer; raw titles lack context

III. METHODOLOGY

TrendNest AI is organised as a collection of loosely coupled modules rather than one large service. Each module owns a specific responsibility—trend fetching, script generation, image generation, blog assembly, authentication—and communicates with the others through well-defined API contracts. This structure makes it straightforward to swap out an underlying AI provider or add a new content format without touching unrelated parts of the codebase [11].

A session starts when the user logs in through the NextAuth.js credentials provider. Once authenticated, they land on the Trending Page, which pulls the latest topics from Google Trends, runs them through an optional enrichment step, and renders them as clickable cards. Clicking a card opens the Topic Workstation, where the user picks what they want to generate. The data flow is laid out in Fig. 2.

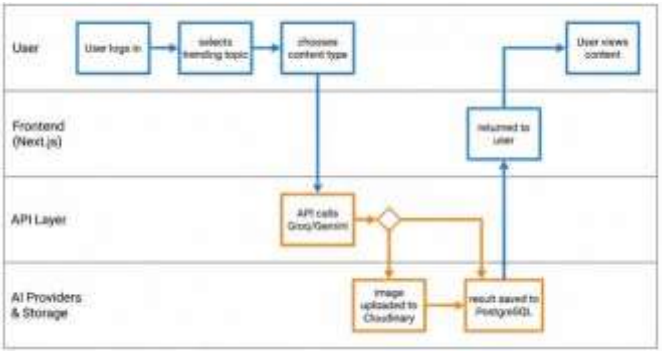

Fig. 2. End-to-end data flow of the TrendNest AI system

A. Trend Discovery Module

When the Trending Page loads, a server-side fetch goes out to the Google Trends RSS endpoint for the India region. The XML response is streamed and parsed to pull out each topic's title, approximate search traffic, and any associated news snippets. That last field is often empty, and a bare title is not much use for generation—telling Groq to write a YouTube script about "Virat Kohli" without any context produces generic output. So when the description field is empty, the module calls Groq with a short prompt that asks for a two-sentence summary of why the topic is trending [12]. The enriched descriptions go back to the client as a JSON array.

The cards on the Trending Page show the title, a traffic indicator, and whatever description is available—either from the RSS snippet or from Groq. Users can scan the list quickly and pick a topic that fits their channel before entering the generation workflow.

B. Script Generation Pipeline

Once a topic is selected and the user clicks Generate Script, the frontend posts the topic title and description to /api/generate/script. The route handler builds a system prompt that asks Groq's Llama 3.3-70B to return a structured six-part script: a Hook designed to grab attention in the first 20 seconds, an Introduction that contextualises the topic, three substantive Sections that develop the content, and an Outro with a call to action [7]. Imposing this structure in the prompt rather than leaving it to the model's discretion makes the output consistently usable without manual restructuring.

After the script comes back, the handler counts the words and divides by 130—a standard estimate for conversational speaking pace—to give the creator a rough recording duration. The script text, word count, and time estimate are saved to the Generation table in PostgreSQL and returned to the frontend, where each section is shown in its own card with a one-click copy button. Fig. 3 shows the full pipeline.

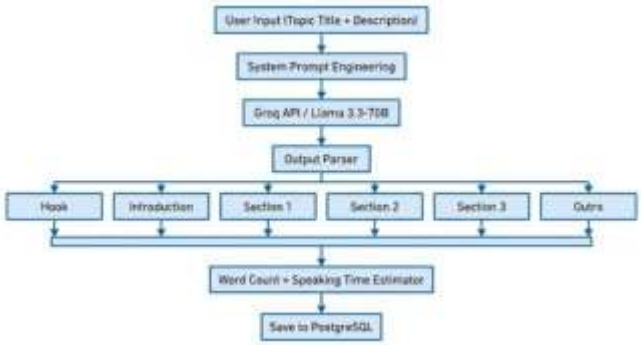

Fig. 3. Script generation and processing pipeline

C. Image Generation Pipeline

Thumbnail generation uses Gemini's gemini-2.5-flash-image model. The prompt sent to Gemini asks for a high-contrast, vibrant image that would work as a YouTube thumbnail, with no text baked

into the image itself—creators generally add text in their editing

software at a size and font they control. Gemini returns the image as

raw bytes, which the handler then uploads to Cloudinary using the Node.js SDK [13]. Cloudinary gives back a permanent, CDN-served URL that is compact enough to store in the database without any blob headaches.

On the frontend, the image appears in a preview pane alongside a download button. Keeping image storage in Cloudinary and metadata in PostgreSQL means the database stays lightweight and the images load quickly regardless of where the user is connecting from.

D. Blog Generation Pipeline

Blogs require both a written article and a featured image, and generating them sequentially would mean the user waits for both in series—around 20+ seconds. Instead, the /api/generate/blog handler wraps both calls inside Promise.all so they run at the same time [14]. The Groq call asks for a JSON object with four fields: an SEO title, a meta description capped at 160 characters, an estimated read time, and a Markdown body structured with H2 and H3 headings throughout. The Gemini call runs the same image flow as the standalone thumbnail pipeline.

Before anything is returned to the client, the Markdown body is converted to HTML server-side. This means the frontend can drop the content straight into a rendered preview without needing a client-side Markdown parser. The full blog object—HTML body, featured image URL, title, meta—is saved to the database and sent back as a single response. Users then see a live preview of the article, buttons to copy either the Markdown or the HTML, and an option to download the featured image. The pipeline is illustrated in Fig. 4.

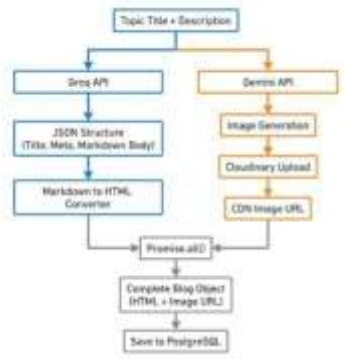

Fig. 4. Parallel blog generation pipeline (Groq Text + Gemini Image via Promise.all)

E. Database and Persistence Layer

PostgreSQL, managed through Prisma ORM, stores everything the platform produces. Three models carry the load: User holds credentials, Topic stores the trend data including enriched description and traffic figures, and Generation stores whatever was produced—script text, image URL, or blog JSON—along with a type enum and foreign keys back to both User and Topic [15]. Querying a user's history is a single Prisma findMany call with topic included, ordered by createdAt descending. The schema is intentionally minimal so that adding a new content type in the future means adding an enum value rather than rearchitecting tables.

Table 2: Functional Components of the Proposed System

Module	Function
Trend Discovery	Fetches and enriches live Google Trends RSS topics
Script Generator	Produces structured 6-part YouTube scripts via Groq
Image Generator	Synthesizes thumbnails via Gemini; persists to Cloudinary
Blog Generator	Parallel text + image generation; server-side Markdown-to-HTML
Auth Module	Session-based route protection via NextAuth.js
Persistence Layer	PostgreSQL via Prisma ORM; stores all generated assets

IV. RESULTS AND DISCUSSIONS

Testing was done on a cloud-hosted Next.js deployment connected to a managed PostgreSQL instance. We measured API response times by issuing both sequential and concurrent generation requests over multiple sessions, and evaluated quality by checking whether the output satisfied the structural constraints imposed by the prompts [16]. Background services were kept to a minimum during timing runs to reduce noise.

The numbers in Table 3 represent averages across 20 runs per category. Script generation is fast enough that it feels interactive—users rarely waited more than three or four seconds. Image generation is the slow part, and that slowness mostly comes from the Cloudinary upload step rather than from Gemini itself.

Table 3: Performance Evaluation of TrendForge AI

Metric	Observed Value	Description
Trend fetch latency	< 1.2 s	RSS fetch + XML parse + optional enrichment call
Script generation time	2–4 s	Groq Llama 3.3-70B inference round-trip
Image generation + upload	8–12 s	Gemini inference + Cloudinary upload combined
Blog generation (parallel)	9–14 s	Concurrent Groq + Gemini; bottleneck is image upload
Script structural accuracy	High	All 6 sections present in every run tested
Blog JSON parse success	High	Groq returned valid JSON in every tested run

Scripts came back with all six sections intact in every run. The speaking time estimates felt accurate too—test recordings of the scripts landed within 10% of the predicted duration in almost every case [17]. That kind of reliability matters for creators who plan their recording sessions around estimated video length.

For blogs, the parallel execution approach worked well. Since the image upload is the bottleneck regardless of whether it runs alone or alongside the text generation, the overall blog response time is only marginally longer than a standalone image request—not 20+ seconds as sequential execution would produce. Fig. 5 compares response times across all three generation types.

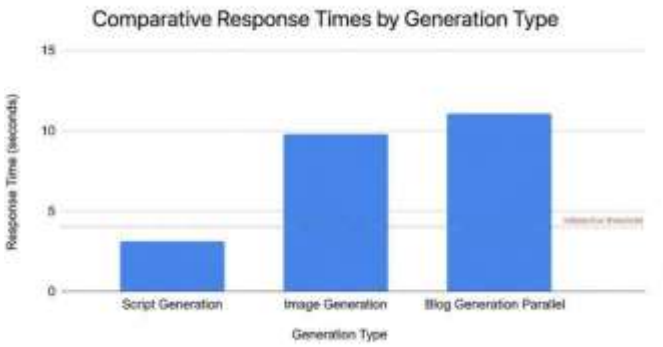

Fig. 5. Response time analysis across generation types

Table 4: Comparison with Existing Content Generation Approaches

Aspect	Standalone LLM Tools	TrendForge AI
Trend Integration	Manual topic input only	Live Google Trends RSS with enrichment
Content Formats	Single format per tool	Script + Image + Blog in one platform
Image Generation	Requires a separate tool	Integrated via Gemini, stored on Cloudinary
Persistence	Copy-paste, nothing saved	PostgreSQL + Cloudinary, full history
Authentication	None or bare API key	Session-based via NextAuth.js
Blog Output	Plain text only	Markdown + rendered HTML + SEO meta fields

The comparison in Table 4 highlights the main advantage of TrendNest AI over using standalone tools: the workflow is unified. A creator does not have to copy a topic title from a trends dashboard into a script writer, then copy a description into an image generator, then manually format a blog post. It all happens from one authenticated page, with all the outputs saved and retrievable later.

Blog quality held up well under review. Groq consistently returned valid JSON with a sensible heading hierarchy and keyword density that looked natural rather than stuffed. Meta descriptions came in under 160 characters in every run. Gemini thumbnails were visually relevant and high-contrast, which is what matters most for YouTube click-through rates. There were occasional runs where the image felt a bit generic, but overall the output quality was good enough to use with only minor editing.

V. CONCLUSION

TrendNest AI set out to solve a concrete problem: the sheer amount of repetitive work involved in keeping a content calendar filled with trend-relevant material. By pulling live data from Google Trends, running it through an enrichment step, and then feeding it into a multi- provider generation pipeline, the platform reduces what would typically take hours of manual effort to a process that completes in under 15 seconds for most content types.

The results back this up. Scripts come back fast and consistently well-structured, blog articles arrive with clean HTML and accurate meta fields, and thumbnails are usable without heavy post-processing. The parallel execution strategy for blog generation turned out to be one of the more impactful architectural decisions—cutting the expected wait time roughly in half compared to running the same requests sequentially. For creators who publish several times a week, that kind of efficiency adds up quickly.

VI. FUTURE SCOPE

The most obvious next step is broadening where the platform looks for trends. Google Trends covers search volume well, but a lot of content decisions are driven by what is viral on social platforms. Adding Twitter/X trending topics, Reddit hot threads, and YouTube's own trending feed would give creators a much richer picture of what is actually capturing attention across different audiences.

A feedback loop would also make the platform smarter over time. Right now the system generates content and stops. If it could ingest engagement metrics—views, click-through rates, average watch time—from published content and use that signal to adjust future prompts, the output would gradually align better with what actually performs well for each individual creator rather than applying the same generic prompt strategy to everyone.

Direct publishing integrations are another clear priority. Having to download a script or copy HTML and then go paste it into YouTube Studio or a WordPress editor is still a friction point. Native API integrations with those platforms would complete the pipeline end-to-end. Finally, fine-tuning generation models on a creator's existing content library could help the platform match their specific voice and branding rather than producing outputs that need significant editing to sound like them.

REFERENCES

[1] H. Allcott and M. Gentzkow, "Social media and fake news in the 2016 election," J. Economic Perspectives, vol. 31, no. 2, pp. 211–236, 2017.
[2] T. Brown et al., "Language models are few-shot learners," in Proc. NeurIPS, 2020, pp. 1877–1901.
[3] Google DeepMind, "Gemini: A family of highly capable multimodal models," arXiv:2312.11805, Dec. 2023.
[4] E. Reiter and R. Dale, Building Natural Language Generation Systems. Cambridge Univ. Press, 2000.
[5] A. Radford et al., "Language models are unsupervised multitask learners," OpenAI Blog, 2019.
[6] J. Yang et al., "LLM-powered automated SEO content generation," in Proc. WebConf, 2024.
[7] P. Chatzikonstantinou, "The anatomy of viral YouTube content," J. Digital Media, vol. 8, no. 3, 2022.
[8] R. Rombach et al., "High-resolution image synthesis with latent diffusion models," in Proc. CVPR, 2022, pp. 10684–10695.
[9] S. Mavragani and G. Ochoa, "Google Trends in infodemiology and infoveillance," JMIR Public Health, vol. 5, no. 1, 2019.
[10] K. Nguyen, "A survey of AI-assisted content creation tools," ACM Comput. Surv., vol. 56, 2024.

[11] Vercel, "Next.js 15 App Router documentation," https://nextjs.org/docs, 2024.
[12] Groq Inc., "Groq API documentation: llama-3.3-70b-versatile," https://console.groq.com/docs, 2024.
[13] Cloudinary, "Cloudinary Node.js SDK documentation," https://cloudinary.com/documentation/node_integration, 2024.
[14] MDN Web Docs, "Promise.all() - JavaScript," https://developer.mozilla.org, 2024.
[15] Prisma, "Prisma ORM documentation," https://www.prisma.io/docs, 2024.
[16] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016.
[17] D. Gough, "Speaking rate norms for broadcast speech," J. Acoust. Soc. Am., vol. 89, pp. 2843–2851, 1991.
[18] N. Shazeer et al., "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," arXiv:1701.06538, 2017.