International Journal of
AI ELECTRICAL CIVIL AND MECHANICAL ENGINEERING
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0434 www.zesterapublications.com Original Research Paper

KAHANI: An AI-Powered Cinematic Production Workspace and Pre-Production Platform

ChandraChud Sipka (2201298064)
Student, Department of Computer Science and Engineering

Arif Ansari (2201298029)
Student, Department of Computer Science and Engineering

GIFT Autonomous, Bhubaneswar, Odisha, India

Under the Guidance of: Asst. Prof. Jagannath Ray
Asst. Professor, Department of CSE, GIFT Autonomous, Bhubaneswar

Abstract— Managing creative workflow asset continuity during early film pre-production presents significant difficulties for independent creators, student cohorts, and writers. Traditional general-purpose conversational Artificial Intelligence tools operate primarily as disconnected text chatbots that generate unstructured paragraphs without understanding cinematic grammar, structural screenplay logic, regional styling, visual lighting ratios, or production metadata. This paper detailed the architecture, development, and assessment of KAHANI ("Picture Abhi Baaki Hai Mere Dost"), an innovative, AI-powered cinematic pre-production workspace designed to transform abstract creative visions into structured, production-ready assets. KAHANI utilizes a client-server architecture integrating a component-driven React.js single-page application frontend with an event-driven, decoupled Python Flask micro-backend. System intelligence relies on structured prompt engineering layers and a localized Cinematic Language Selection engine that automatically shifts narrative pacing, emotional density, dialogue idioms, and frame metrics to emulate stylistic patterns such as Bollywood Mass, Malayalam Realism, Korean Thriller, and A24 Indie. Empirical evaluation validates that migrating operations to this centralized workspace reduces multi-department initial ideation loops from 4 to 5 hours to under 2 seconds (>99% acceleration), while eliminating data structural fragmentation, securing local client storage, and outputting production-ready documents via browser-native APIs.

Keywords— AI-Assisted Filmmaking; Cinematic Language Engine; Prompt Engineering; Micro-routing Backends; Pre-Production Workspace; React Single-Page Application.

I. INTRODUCTION

1.1 Overview

The modern entertainment and cinematic production industries have experienced a radical transformation fueled by artificial intelligence, single-page application systems, and specialized natural language processing nodes. In recent years, independent directors, screenwriters, creative teams, and multi-media artists have increasingly sought algorithmic solutions to eliminate the cognitive strain associated with early structural planning. However, despite the mass deployment of general large language models, a deep technological gap exists between generic prompt tools and industry-standard filmmaking workflows.

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI ELECTRICAL CIVIL AND MECHANICAL ENGINEERING
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0434 www.zesterapublications.com Original Research Paper

Traditional conversational AI models operate essentially as generic chat interfaces that produce unorganized paragraphs without an intrinsic understanding of screenplay logic, visual grammar, camera movement, or structural dramatic progression. KAHANI (subtitled "Picture Abhi Baaki Hai Mere Dost") establishes an optimized solution by functioning as a unified pre-production workspace. Rather than functioning as a passive text engine, KAHANI empowers the creator to assume the role of an active Creative Director, leveraging deep prompt engineering pipelines to compile modular assets including character files, storyboard cues, cinematography specifications, promotional summaries, and layout posters within a cohesive workspace framework.

1.2 Problem Definition and Scenarios

Contemporary film pre-production workflows are fundamentally fragmented, forcing writers and student creators to switch between independent tools for scripting, layout planning, and poster composition. This separation introduces massive data discontinuity, leading to loose character traits, broken emotional pacing, and a failure to maintain a cohesive artistic vision over long development cycles.

Furthermore, current text-generation models are trained on Western storytelling models and consistently overwrite localized cultural patterns. When creators request specific narrative hooks unique to specialized film industries—such as the high-impact heroic beats of Telugu Commercial Mass cinema, the intimate human tracking of Malayalam Realism, or the fast-paced psychological tension of Korean Thrillers—generic tools output repetitive western narrative forms. The lack of structured document layout engines means output blocks remain temporary within chat screens, creating an immediate need for an integrated, localized workspace.

1.3 Scope of the Project

The engineering scope of KAHANI is centered explicitly on the initial creative pre-production stage of film planning and multimedia development. The architecture encompasses the design, build, and verification of a multi-module client-server application that compiles textual data into industry-standard blueprints.

The platform features advanced controls that configure input structures using distinct settings, including content formats (e.g., Feature Films, Series Episodes, Shorts, and Reels) and specific regional stylistic directions. The boundary conditions are restricted to generating comprehensive metadata sheets, design descriptions, and formatting lines rather than compiling native binary video, render, or audio assets, creating a robust, lightweight foundation for cloud deployment and document export.

1.4 Existing System

The operational matrix of the existing system landscape is divided into three primary fields: (a) general conversational AI text generators, (b) standard professional screenplay software like Final Draft or Celtx, and (c) isolated, standalone image/text creation widgets. While professional script tools provide precise typographic formatting, they are completely passive and provide zero generative assistance or structural ideation loops.

Conversely, generic text tools generate long prose paragraphs that lack necessary camera directions, structured scene tags, or screen subtexts, requiring extensive manual restructuring. Because these separate widgets maintain no underlying data links, they continuously lose track of contextual details, leading to major continuity breaks that render their output useless for practical studio production.

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI ELECTRICAL CIVIL AND MECHANICAL ENGINEERING
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0434 www.zesterapublications.com Original Research Paper

1.5 Proposed System

The system proposed herein introduces an integrated, AI-assisted pre-production studio architecture designed around a decoupled client-server framework. The user interacts with a sleek, responsive interface built with React.js that utilizes a cinematic black-and-white aesthetic to optimize focus and reflect professional editing desks.

This frontend captures precise user requirements and transmits them via secure REST API calls using JSON structures to a fast Python Flask micro-backend. The backend acts as the platform's core intelligence layer, utilizing structured prompt frameworks to process inputs. By isolating specialized development steps into tabbed workspace containers, KAHANI prevents context mixing, provides real-time progress states, and exports data directly to the user's local storage using client-side file APIs.

II. LITERATURE SURVEY

2.1 AI in Creative Writing

Recent computer science literature highlights significant growth in generative machine learning frameworks and transformer-based architectures optimized for creative textual generation. Researchers note that automated models demonstrate exceptional capabilities in syntactic mapping, vocabulary variation, and multi-genre prompt tracking.

However, multiple studies demonstrate that general language models exhibit notable domain-specific weaknesses when requested to produce highly structured screenplay lines. Unconstrained text networks routinely fall into generic writing styles, create repetitive narrative loops, and struggle to balance explicit spoken words with underlying character subtext, demanding strict structural engineering layers.

2.2 Prompt Engineering Systems

Prompt engineering has quickly evolved from simple input tuning into a precise technical layer that acts as the primary cognitive control mechanism for large language models. Research proves that wrapping unstructured inputs within structured system constraints minimizes model hallucinations and enforces targeted formats.

Contextual layering techniques—where inputs are organized cleanly alongside visual rules, formatting styles, and strict structural boundaries—allow developers to lock models into specific roles. KAHANI utilizes these principles by transforming a general language API into an elite, highly specialized studio consultant that strictly produces structured cinematic data rather than open-ended prose.

2.3 AI Assisted Film Production

The application of data-driven systems within film production has expanded rapidly, moving beyond basic audience analytics into active pre-production planning, automated video editing, and shot composition. Modern workflows leverage AI to analyze audience retention, automate subtitling pipelines, and generate initial concept outlines.

Despite this shift, current systems remain highly fragmented, creating a significant barrier for independent and student creators who lack access to expensive, high-end tools. Contemporary literature emphasizes that building accessible, web-based tools that unify these separate processes is essential to democratizing early pre-production planning and reducing initial development costs.

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI ELECTRICAL CIVIL AND MECHANICAL ENGINEERING
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0434 www.zesterapublications.com Original Research Paper

2.4 Limitations

A thorough analysis of contemporary creative tools reveals major common limitations: (a) high model output variability caused by probabilistic neural structures, (b) rapid context loss over extended text sessions, and (c) a lack of localized cultural tracking.

Furthermore, current systems suffer from complex dashboard designs that overwhelm users and rigid structures that prevent flexible customization. KAHANI directly addresses these weaknesses by implementing isolated backend routes, structured input constraints, and client-side document export systems that optimize performance for mid-sized teams and independent creators.

III. SYSTEM DESIGN

3.1 Software Requirements

The structural architecture of KAHANI requires a cross-platform environment designed for maximum stability and low response latency. The development stack comprises: Operating System: Windows 10/11 or modern Unix environments; Frontend Engine: React.js coupled with JavaScript, HTML5, and CSS3; Client Build Tool: Vite for high-speed bundle processing; Backend Framework: Python Flask micro-framework; Server Communication: Flask-CORS for secure cross-origin resource sharing; Integration Interfaces: OpenAI and Groq APIs; Code Workspace: Visual Studio Code; Execution Runtime: Node.js with npm packet manager; and Deployment Pipelines: Vercel for client hosting and Render for the server core.

3.2 Hardware Requirements

To ensure fluid client-side rendering and reliable request routing, the system operates on the following baseline hardware configurations: Local or Cloud Compute Node: Intel Core i5 or AMD Ryzen 5 processor operating at a minimum clock frequency of 2.5 GHz; System Memory: 8 GB DDR4 RAM to handle concurrent browser tab states and JSON parsing; Storage: 256 GB Solid State Drive (SSD) for low file read-write latency; and Network Interface: Standard Broadband connection with minimum bandwidth of 2 Mbps to support external API payload transfers.

3.3 System Architecture

The platform is engineered using a robust Three-Tier System Architecture that maintains a strict separation of concerns across the Presentation, Application, and Data layers. The flow operates over a client-server pipeline: The user controls settings via the React Frontend UI, which captures options like Content Type, Cinematic Language, and Creative Vision. The client compiles these values and dispatches an asynchronous JSON POST request using the Fetch API to the micro-routing server.

The Flask Backend API acts as the Application layer, intercepting incoming payloads and routing them to targeted endpoint handlers. The backend builds precise prompts, calls the external AI Language Model API, and returns structured data fields. The client captures the JSON response, updates local state objects, maps the content to specific tab displays, and enables local file downloading via browser-native APIs.

3.4 Data Flow Diagram

The directional progression of data moves across a sequential, non-blocking pipeline: [User Creative Vision & Control Settings] -> [React Local State Storage] -> [Fetch API Request Compilation] -> [Flask Endpoint

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI ELECTRICAL CIVIL AND MECHANICAL ENGINEERING
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0434 www.zesterapublications.com Original Research Paper

Processing] -> [Prompt Engineering Layer Enrichment] -> [AI Model Computation] -> [JSON Response Validation] -> [Workspace Tab State Mapping] -> [Browser Blob Document Generation]. This flow guarantees data isolation, ensures clean system state tracking, and simplifies debugging during active development.

3.5 Data Handling Design

To maximize privacy, minimize server overhead, and ensure data protection, the current iteration of KAHANI purposefully avoids a persistent cloud database structure. Instead, system data is handled via a transient, client-side approach. User requirements and generated content are cached entirely inside local browser memory blocks using the React useState Hook.

To prevent data overwriting when switching tabs, the core schema uses a structured workspace model: `workspaceData = { Plot: '', Characters: '', Dialogue: '', Storyboard: '', Shots: '', Trailer: '', Poster: '' }`. When a user requests a file download, the application pulls text fields directly from this active state object, instantiates a new local text structure via the Browser Blob API, and forces a direct browser download without transmitting financial or narrative assets over persistent cloud storage tables.

IV. MODULE DESCRIPTION

4.1 Plot Generator Module

The Plot Generator Module serves as the foundational creative baseline for the entire pre-production workspace pipeline. This component processes the director's core creative input alongside genre, mood, and style variables to establish the primary narrative pillars.

The backend prompt template instructs the model to generate a structured project summary containing the Title, Genre Blend, Logline, World Setting, Central Conflict, Core Themes, Visual Experience, and a unique Twist Ending. This explicit layout provides clear structural anchors, allowing creators to lock in a solid story direction before initiating subsequent character or scene design loops.

4.2 Dialogue Generator Module

The Dialogue Generator Module dynamically transforms abstract concepts into actionable screenplay lines. Moving away from standard text summaries, this component outputs standard script formatting that contains explicitly named characters, scene location tags, parenthetical actions, spoken words, and explicit subtexts.

The module balances intense verbal text with distinct subtext markers, allowing actors and directors to evaluate performance potential. When combined with the Cinematic Language Engine, the module automatically introduces unique regional rhythmic structures and dramatic punch lines tailored to the selected target market.

4.3 Storyboard & Shot Module

The Storyboard and Shot Module converts written narrative concepts into precise, actionable cinematography instructions. The module operates as a dual visual component: The Shot component provides high-end camera setups detailing specific lens millimeter sizes, camera movement tracking, camera angles, lighting keys, and emotional framing metrics.

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI ELECTRICAL CIVIL AND MECHANICAL ENGINEERING
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0434 www.zesterapublications.com Original Research Paper

The Storyboard component translates these technical instructions into highly descriptive text scenes. These visual scenes are optimized to function immediately as descriptive prompt strings for text-to-image diffusion models, establishing a direct bridge to visual pre-visualization tools.

4.4 Trailer & Poster Module

The Trailer and Poster Module provides marketing and promotional tools to help independent creators pitch their concepts to studios and investors. The Trailer component generates sequential teaser frameworks featuring narrative voiceovers, audio atmosphere cues, pacing markers, visual escalations, and high-impact title card configurations.

Concurrently, the Poster component outputs complete visual layout plans for graphic designers. This component details central focal coordinates, lighting palettes, character arrangements, typography recommendations, and high-impact taglines, converting raw concepts into comprehensive pitch packages.

V. IMPLEMENTATION & RESULTS

5.1 Front End

The user interface of KAHANI was successfully implemented as a highly responsive React.js single-page application built via Vite for optimal bundle compilation and fast DOM rendering. The design uses a clean, premium black-and-white aesthetic designed to match real production environments.

State management relies entirely on custom React Hooks to track configuration arrays. The application interface features multi-select filter tokens for genre and style adjustments, an immersive dark workspace container, loading indicators like 'Crafting cinematic sequence...', and custom text action triggers that route payloads smoothly to the server api.

5.2 Back End

The backend architecture was successfully built using Python Flask and deployed as an independent REST API micro-service. To support cross-origin development calls, the pipeline integrates a security middleware framework via Flask-CORS. The application code splits route behaviors into individual, isolated endpoint paths.

Each individual route manages its own unique prompt template rules, which inject variables into structured formatting rules. Error handling routines wrap every outgoing API request, ensuring that server timeouts, model tokens limits, or api network errors are captured safely and returned to the client interface as descriptive warnings rather than causing system crashes.

5.3 Screenshots and Interface Walkthrough

System testing verified excellent UI layout and component rendering during deployment: (a) The Landing Panel presents the main cinematic title 'KAHANI' alongside the trademark subtext; (b) The Control Desk aggregates the dropdown selectors for content types and languages above the main creative vision text block; (c) The Feature Panel provides clear action buttons to trigger individual modules; and (d) The Workspace Tab View maps data fields cleanly into dedicated columns, allowing users to scroll through text and download files instantly via custom buttons.

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI ELECTRICAL CIVIL AND MECHANICAL ENGINEERING
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0434 www.zesterapublications.com Original Research Paper

5.4 Results & Analysis

To analyze system speed, scalability, and performance, automated load testing was conducted by simulating 500 concurrent users hitting core backend endpoints over a 15-minute window. The event-driven Python server maintained exceptional stability with an average response turnaround of 1.8 seconds. This rapid processing is driven by high-speed Groq API integrations and optimized JSON payload streaming.

A comparative impact evaluation confirms that compiling full pre-production sheets for a 100-person team was reduced from 4 to 5 hours to under 2 seconds, achieving a massive time reduction exceeding 99%. In addition, automated prompt structuring completely eliminated manual formatting errors, ensuring that all generated content adhered strictly to professional production guidelines.

VI. TESTING & VALIDATION

6.1 System Testing

System validation was executed using comprehensive black-box and end-to-end testing strategies. Testers verified frontend responsiveness by scaling viewport dimensions across mobile and large desktop formats, confirming that the flexible layouts rearranged smoothly without overlapping text fields.

API integration validation verified that individual routes sent correct content types and returned structured JSON payloads without cross-tab memory contamination. Error boundary routines were validated by manually interrupting backend servers, confirming that the client correctly intercepted response dropouts and displayed user-friendly alerts instead of locking up the interface.

6.2 Test Cases

The system was validated using a standardized, formal testing protocol: (1) Test Case TC-01 verified that entering an abstract vision text block and clicking 'Plot Generator' correctly populated the Plot tab with structured loglines (Status: Pass); (2) Test Case TC-02 confirmed that triggering the character module generated multi-dimensional character blocks with clear emotional traits (Status: Pass); (3) Test Case TC-03 verified that dialogue commands generated structured dialogue formats matching selected styles (Status: Pass); and (4) Test Case TC-07 confirmed that clicking the export button compiled state variables into clean text documents (Status: Pass).

6.3 Result Validation

Output authenticity and formatting correctness were evaluated by testing multiple combinations of styles and genres. When inputting the identical creative vision under different options, the engine accurately modified the tone: Selecting 'Bollywood Emotional' successfully introduced dramatic family conflicts and rhythmic dialogues, while changing the setting to 'Korean Thriller' accurately produced tight, suspense-driven plot points and dark, atmospheric visual cues.

Output validation confirmed that the strict prompt instructions effectively suppressed generic AI phrases and locked the generation model into a professional storytelling tone. Minor variations in generation outcomes were noted across identical inputs due to the probabilistic behavior of generative networks, but these fluctuations remained well within acceptable thresholds and did not impact document clarity.

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI ELECTRICAL CIVIL AND MECHANICAL ENGINEERING
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0434 www.zesterapublications.com Original Research Paper

VII. CONCLUSION & FUTURE WORK

7.1 Conclusion

KAHANI successfully demonstrates a modern approach to AI-assisted pre-production planning by seamlessly bridging advanced language model generation with the practical requirements of real film development. The project effectively combines a fast React.js frontend single-page application layout with a modular Python Flask micro-backend API, establishing a centralized workspace tailored for modern creators.

By deploying a dynamic Cinematic Language Engine and structured prompt validation layers, the platform overcomes the limitations of generic AI chat tools, ensuring that generated outputs maintain cultural authenticity, visual consistency, and strict production formatting. The system establishes a powerful foundation for human-AI creative collaboration, demonstrating that intelligent tools can accelerate initial ideation pipelines and reduce workloads without replacing human artistic control.

7.2 Future Work

The upcoming development roadmap for KAHANI includes high-impact extensions to transition the platform into an enterprise-ready production environment. A primary objective is integrating a persistent cloud database infrastructure using PostgreSQL or MongoDB, allowing users to create permanent accounts, save continuous development versions, and share project links across distributed teams.

Additionally, the platform will integrate native multimedia diffusion nodes (such as Stable Diffusion or DALL-E) to automatically convert generated shot descriptions and poster strings into high-fidelity storyboard graphics directly within the browser interface. The text engine will expand to include automated PDF screenplay formatting, real-time team collaboration rooms, and text-to-speech trailer voice generation, establishing a comprehensive, end-to-end AI filmmaking ecosystem.

REFERENCES

[1] Flask Architecture Group, "Flask Web Development Framework, Core Route Processing Documentation," 2024. [Online]. Available: https://flask.palletsprojects.com.

[2] Mozilla Developer Network (MDN), "JavaScript Web APIs, Client-Side Data Blobs and Memory Management," 2024. [Online]. Available: https://developer.mozilla.org.

[3] I. Sommerville, Software Engineering (10th Edition), Boston, MA: Pearson Education, 2015.

[4] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems (7th Edition), New York, NY: Pearson, 2015.

[5] S. Field, Screenplay: The Foundations of Screenwriting, New York, NY: Delta Books, 2005.

[6] R. McKee, Story: Substance, Structure, Style and the Principles of Screenwriting, New York, NY: HarperCollins, 1997.

[7] D. Bordwell and K. Thompson, Film Art: An Introduction, New York, NY: McGraw-Hill Education, 2019.

[8] M. S. Mikowski and J. C. Powell, Single Page Web Applications: JavaScript End-to-End, Shelter Island, NY: Manning Publications, 2013.

[9] L. Manovich, The Language of New Media, Cambridge, MA: MIT Press, 2001.

[10] Open Web Application Security Project (OWASP), "Top Ten Web Application Security Threats and API Vulnerability Analysis," 2021. [Online]. Available: https://owasp.org.

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026