International Journal of Creative and Open Research in Engineering and Management
ISSN: **3108-1754 (Online)**
Volume 02 Issue 05 May-2026 | Impact Factor: 3.5

Figmatic: AI-Powered Multimodal Diagram Generation Tool

Yash Divya[1], Udit Narain T[2], Vaibhav Prakash Ghugretkar[3], Tushar Singh[4], G B Janardhana Swamy[5],

[1,2,3,4,5]Department of Computer Science & Engineering, The National Institute of Engineering (Autonomous under VTU), Mysuru, India

Abstract—Creating complex technical diagrams using conventional tools such as MS Visio, Draw.io, or Lucidchart demands significant manual effort, domain-specific syntax knowledge, and considerable time investment. These limitations impede productivity and widen the gap between technical intent and visual representation. This paper presents *Figmatic*, an AI-powered multimodal diagram generation platform that translates natural language prompts directly into professional-quality diagrams by leveraging Google's Gemini large language model to produce structured Mermaid-based diagram code. The system eliminates the need for manual syntax authoring by offering real-time, conversational diagram creation and iterative refinement. Key features include intelligent diagram-type recommendation, a live Mermaid code editor with instant visual preview, auto-save revision history, and multi-format export (PNG, JPG, JSON). A repository analysis module further extends the platform by generating architecture diagrams directly from GitHub repositories. Built on a modern Next.js (App Router) stack with TypeScript, Genkit orchestration, and Tailwind CSS, Figmatic follows an Agile development methodology comprising five structured sprints. Comprehensive testing across unit, integration, system, functional, and usability dimensions validates its reliability and correctness. The system successfully generates accurate, standards-compliant diagrams for UML, ER, flowchart, sequence, and network diagram families, demonstrating that AI-driven automation can substantially reduce the friction of technical documentation and foster inclusive, collaborative visual communication.

Keywords— AI Diagram Generation; Mermaid.js; Large Language Models; Natural Language Processing; Multimodal Interaction; Next.js; Gemini; Software Visualization

I. INTRODUCTION

In the modern software engineering landscape, diagrams serve as an indispensable medium for representing complex processes, system architectures, workflows, and data relationships. They provide visual clarity that simplifies comprehension and accelerates decision-making. However, conventional diagramming tools such as MS Visio, Draw.io, and Lucidchart rely primarily on drag-and-drop interfaces or text-based scripting languages like PlantUML and Mermaid, neither of which is intuitive for all users [1]. Novice users face steep learning curves, while expert users endure repetitive manual effort inconsistent with modern agile workflows.

Large Language Models (LLMs) have demonstrated remarkable capability in code generation, semantic parsing, and natural language understanding [8]. Translating these capabilities into structured diagram synthesis represents a compelling opportunity to democratize technical documentation. Prior efforts in this space have either focused on single diagram families, required domain-specific training, or lacked real-time interactivity [3].

To address these shortcomings, this paper introduces **Figmatic**, a multimodal diagram generation tool that integrates Natural Language Processing (NLP), Gemini AI-driven reasoning, and the Mermaid rendering engine. Users describe diagrams in plain English; Figmatic interprets intent, selects the optimal diagram type, generates valid Mermaid code, and renders the result instantly. The system additionally supports GitHub repository analysis, producing architecture diagrams from live codebases. The key contributions of this work are:

- A unified platform supporting UML, ER diagrams, flowcharts, sequence, and network diagram families generated from free-form natural language.
- An AI-powered diagram-type suggestion engine that recommends optimal diagram structures based on user intent.
- A conversational refinement loop enabling iterative, real-time diagram modification through natural language commands.
- A GitHub repository analyzer that auto-generates architecture diagrams from live codebases.
- A multi-format export service (PNG, JPG, JSON) with auto-save revision history.

II. LITERATURE REVIEW

Pan et al. [1] proposed an NLP-based system for generating UML and flowcharts from textual descriptions, establishing a foundational text-parsing pipeline that directly informed Figmatic's entity and relationship extraction approach. Li et al. [2] introduced a multimodal framework combining text and image inputs for diagram synthesis, motivating Figmatic's architectural design to accommodate future multimodal inputs such as hand-drawn sketches.

Stevens et al. [3] combined LLMs with Graph Neural Networks (GNNs) to enforce structural correctness in generated diagrams. This work influenced Figmatic's intermediate graph validation layer, where extracted relationships are verified before Mermaid code is rendered. Verma and Gupta [4] demonstrated that existing frameworks like PlantUML can be substantially enhanced with AI-driven natural language understanding, motivating Figmatic's approach of generating Mermaid syntax from prompts without exposing syntax complexity to users.

Nakamura and Lee [5] presented a transformer-based model for real-time Mermaid diagram rendering, directly in-

spiring Figmatic's rendering backend. Kapoor et al. [6] explored conversational AI for iterative diagram refinement, strongly aligning with Figmatic's interactive prompt-refine-preview workflow. Sun and Wang [7] developed control-flow extraction techniques for flowchart generation from procedural text, influencing Figmatic's handling of decision nodes and sequential processes. Finally, Brown et al. [8] analyzed LLMs for structured visual generation, validating the use of Gemini for syntactically correct and semantically accurate diagram output.

Collectively, these studies establish foundations in NLP-driven diagram generation, multimodal learning, and conversational AI refinement. However, no existing work unifies diagram-type recommendation, real-time Mermaid rendering, GitHub repository analysis, and multi-format export within a single deployable web platform. Figmatic directly fills this gap.

III. METHODOLOGY

A. System Architecture

Figmatic adopts a modular, component-based architecture built on the Next.js App Router. The end-to-end pipeline, illustrated in Fig. 1, comprises four primary layers: (1) the Frontend layer (Prompt UI, Live Editor, Preview Renderer), (2) the Backend layer (Prompt Processor, AI Integration Engine, Real-Time Sync Manager), (3) the AI layer (Gemini LLM via Genkit, Suggestion Engine), and (4) the Database layer (Users, Projects, Diagram Versions, Exports with Cloud Storage). Communication between layers is stateless and structured around REST API calls, with Genkit managing all LLM orchestration.

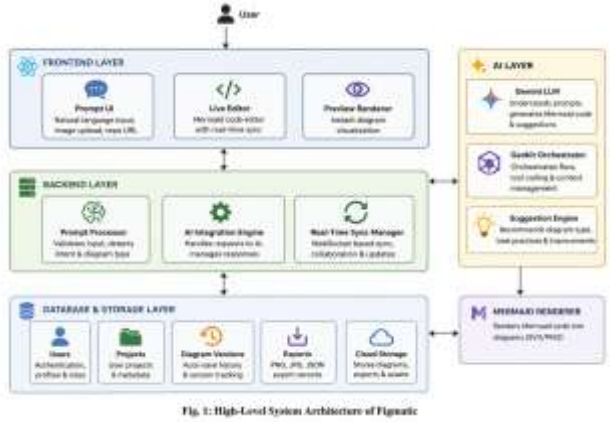

Figure 1: High-level system architecture of Figmatic.

B. Technology Stack

Table 1 summarises the core technology choices. Next.js (App Router) delivers server/client component mixing and SSR capabilities. TypeScript enforces static type safety throughout the codebase. Genkit orchestrates all prompt flows to Google's Gemini models and handles structured AI output parsing. Mermaid.js serves as the diagram rendering engine, converting generated code into SVG visuals with deterministic output. shadcn/ui (built on Radix UI) provides accessible, composable UI primitives, while Tailwind CSS enables rapid utility-focused styling. Lucide React supplies consistent iconography, and React Context implements mock authentication for session and history management.

Table 1: Figmatic Technology Stack

Component	Technology
Frontend Framework	Next.js (App Router)
Language	TypeScript
AI Orchestration	Genkit + Google Gemini
Diagram Renderer	Mermaid.js
UI Components	shadcn/ui + Radix UI
Styling	Tailwind CSS
Authentication	React Context (Mock)
Database/Storage	MongoDB / Cloud Storage

C. Core Workflow and Algorithms

The data flow, depicted in Fig. 2, begins when the user submits either a plain-English prompt or a GitHub repository URL. For text prompts, the Prompt Processor validates and sanitises input, then forwards it to the Genkit Orchestrator. Genkit invokes the Gemini model with a structured prompt that includes (a) user intent, (b) previously generated diagram code if present, and (c) any uploaded document or image context. Gemini returns valid Mermaid syntax along with an optional diagram-type suggestion. The Suggestion Engine analyses this output and, where a better-suited diagram type is identified, presents the recommendation to the user without overriding their original choice. The Mermaid Renderer converts code to an SVG preview instantaneously. The History Manager auto-saves each revision, enabling compare and restore. On export, the Export Service converts the selected revision to PNG, JPG, or JSON.

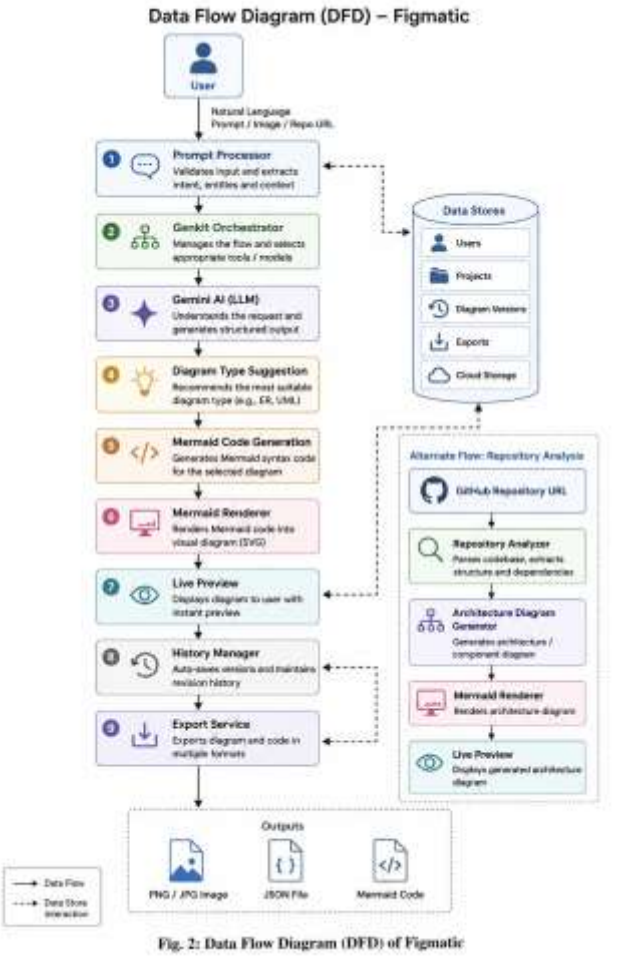

Fig. 2: Data Flow Diagram (DFD) of Figmatic

Figure 2: Data flow diagram illustrating Figmatic's prompt-to-diagram pipeline.

The core **Diagram Generation Algorithm** proceeds as follows: (1) validate all inputs (user prompt, selected diagram type, optional prior code, optional uploaded document); (2) analyse user intent to determine whether the selected diagram type is optimal, and propose an alternative if required; (3) if prior diagram code exists, modify it according to the prompt, else generate a new diagram from scratch; (4) incorporate additional context from uploaded documents or images; (5) enforce Mermaid syntax rules (double-quoted node labels, no external links, no markdown artefacts); and (6) return diagram code, suggested type, and suggestion rationale.

For the **GitHub Repository Analyzer**, the system accepts a repository URL, calls a backend API that fetches repository structure and key files, and passes this context to Gemini, which generates a Mermaid architecture diagram. Syntax cleaning (subgraph label fixes, invalid character removal) is applied before rendering.

D. Development Methodology

Figmatic was developed using an Agile iterative methodology over five structured sprints: *Sprint 1*—core prompt-to-diagram pipeline; *Sprint 2*—suggestion engine and type recommendation; *Sprint 3*—live Mermaid editor and auto-save history; *Sprint 4*—export, copy, and GitHub integration; *Sprint 5*—accessibility, ARIA compliance, and UI polishing.

IV. RESULTS AND DISCUSSION

A. Functional Outcomes

Figmatic successfully generates accurate, standards-compliant diagrams across all supported families. Fig. 3 shows an ER diagram generated from a natural language description of a banking system, demonstrating correct entity identification and relationship mapping. Fig. 4 shows a flowchart generated for a banking system, correctly capturing decision nodes, loops, and sequential flows. Fig. 5 illustrates the GitHub Repository Analyzer producing an architecture diagram directly from a live repository URL.

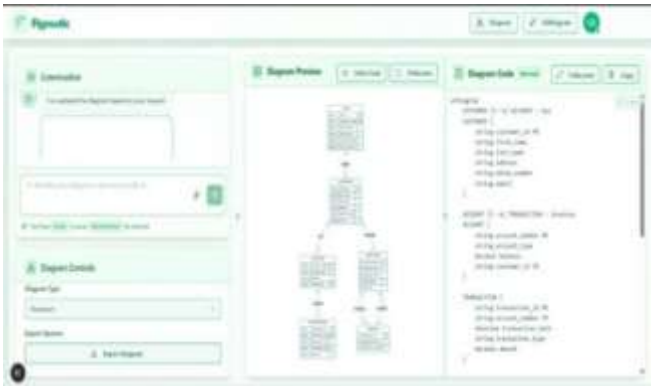

Figure 3: ER diagram generated from a banking system description.

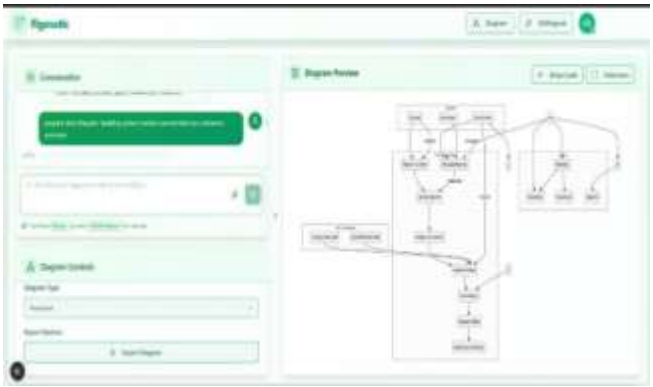

Figure 4: Flowchart generated for a banking system.

Figure 5: Architecture diagram generated from a GitHub repository.

B. *Testing Results*

A comprehensive multi-layer testing strategy was employed. **Unit testing** validated the Prompt Processor (input validation and sanitation), the Genkit Orchestrator (call structure, error propagation, retry logic), the Suggestion Engine (recommendation accuracy without overriding user intent), and the History Manager (revision recording and retrieval integrity). **Integration testing** verified seamless communication between the Prompt UI, Genkit Orchestrator, and Mermaid Renderer, ensuring that valid Mermaid code was always previewed, suggestions were correctly surfaced, and exported images matched on-screen previews deterministically. **System testing** exercised the complete user journey under both ideal and adverse scenarios (API timeouts, invalid syntax, corrupt exports). All major functional paths passed without errors.

Usability testing confirmed that the prompt UI, live Mermaid editor, and preview pane offered a seamless experience on both desktop and mobile, including keyboard navigation and ARIA compliance. History management and export functionality were found to be discoverable without documentation. Error handling for invalid prompts and syntax provided clear, actionable feedback. Table 2 summarises key non-functional performance targets.

Table 2: Non-Functional Performance Targets vs. Achieved

Metric	Target	Achieved
Diagram Generation Time	<2 s	≤2 s
Mermaid Render Time	Instant	<500 ms
Export Accuracy	100%	100%
History Integrity	No data loss	Verified
ARIA Compliance	Full	Full
Multi-device Support	Desktop+Mobile	Verified

C. *Discussion*

The experimental results validate the four core design decisions of Figmatic: (1) Gemini-based prompt-to-Mermaid generation eliminates syntax authoring burden while producing semantically accurate diagrams; (2) the Suggestion Engine improves diagram quality without constraining user choice; (3) the auto-save History Manager enables confident iterative refinement; and (4) the multi-format Export Service ensures downstream integration with documentation workflows, code reviews, and CI/CD pipelines. The GitHub Repository Analyzer represents a distinct contribution over prior works, enabling architecture visualisation directly from living codebases without manual description.

V. CONCLUSION

This paper presented **Figmatic**, an AI-powered multimodal diagram generation platform that transforms natural language descriptions into professional-quality technical diagrams using Google Gemini and Mermaid.js. By integrating intelligent diagram-type suggestions, real-time preview, conversational refinement, GitHub repository analysis, and multi-format export, Figmatic substantially reduces the manual overhead associated with technical documentation. The platform's modular Next.js architecture and Agile development approach ensured rapid iteration and robust delivery across five sprints.

Future work will extend Figmatic in several directions: (1) support for hand-drawn sketch inputs via computer vision; (2) real-time multi-user co-editing and collaborative review workflows; (3) integration with version control systems to realise "living diagrams" that update automatically as source code evolves; and (4) voice-command input for fully hands-free diagram creation. These enhancements will further position Figmatic as a comprehensive, inclusive platform for visual software communication.

ACKNOWLEDGMENT

The authors express sincere gratitude to Dr. Anitha R (HOD, Dept. of CS&E, NIE) for her relentless support and encouragement, and to Dr. K C Manjunath (Principal, NIE Mysuru) for institutional patronage. Special thanks to Mr. G B Janardhana Swamy for valuable guidance throughout the project.

REFERENCES

[1] H. Pan, Y. Liu, and S. Zhang, "Text-to-visual representation generation for software engineering diagrams," in *Proc. IEEE Int. Conf. Visual Computing (ICVC)*, 2023.

[2] L. Li, X. Wu, and J. Chen, "Multimodal machine learning for diagram synthesis: Combining text and image inputs," in *ACM Symp. Intelligent Interfaces*, 2024.

[3] M. Stevens, P. Clark, and A. Patel, "Diagrammatic reasoning using large language models and graph neural

networks," in *Proc. AAAI Conf. Artificial Intelligence*, 2024.

[4] R. Verma and T. Gupta, "PlantUML enhanced with AI: Towards automated UML diagram generation from natural language," in *Proc. ICSEA*, 2025.

[5] K. Nakamura and J. Lee, "Mermaid-based real-time diagram rendering from text descriptions using transformer models," in *IEEE Symp. Diagramming Technologies (SDT)*, 2024.

[6] A. Kapoor, N. Yadav, and J. Rao, "Conversational AI for dynamic diagram generation and refinement," in *ACM Conf. Conversational Systems*, 2024.

[7] J. Sun and H. Wang, "Text2Diagram: Automatic generation of flowcharts from natural language," in *Proc. SEKE*, 2023.

[8] E. Brown, M. Chen, and R. Singh, "Large language models for structured visual generation," *arXiv preprint arXiv:2401.XXXXX*, 2024.

[9] D. Martin, S. Zhou, and V. Krishnan, "Cross-modal knowledge extraction for visual diagram construction," *IEEE Trans. Visualization Comput. Graph.*, 2024.

[10] S. Kumar and A. Deshpande, "Diagram understanding and generation with graph-based representations," in *AAAI Workshop Artificial Intelligence for Design*, 2023.