Archives available at journals.mriindia.com

International Journal on Advanced Computer Engineering and Communication Technology

ISSN: 2278-5140

Volume 14 Issue 03s, 2025

GENWEB : An AI-Powered Platform For Intuitive Prompt-To-Website Generation

[1] Aliasgar S. Kagajwala, [2] Tanvi Wagde, [3]Ruchika Kambe, [4] Pracheta Lokhande, [5] Prof. Pooja Wagh

[1,2,3,4,5]Department of Artificial Intelligence
St.Vincent Pallotti College of Engineering & Technology
Nagpur, India

Peer Review Information

Submission: 05 Nov 2025

Revision: 25 Nov 2025

Acceptance: 17 Dec 2025

Keywords

AI-generated websites, intent-based design, large language models, web automation, prompt-to-website.

Abstract

In today's digital-first world, a strong online presence has become essential for individuals, creators, and businesses alike. Yet, despite countless website-building platforms such as Wix, Squarespace, and WordPress, creating a unique and personalized website remains a challenge for many. These platforms, while simplifying web development, still confine users within rigid templates and limited customization options. For someone without coding knowledge, translating a creative vision into a functional, aesthetic website often demands either expensive professional help or frustrating hours of trial and error. This persistent gap between imagination and execution highlights a major barrier the knowledge dependency of web creation.

To address this challenge, Gen-Web introduces a new paradigm of website generation powered by artificial intelligence and intent-based design. Instead of choosing from predefined templates, users simply describe their vision in natural language for example, *"I want a modern, dark-themed portfolio with a hero section, about page, and project gallery."* Gen-Web interprets this request, understands the intent, and constructs a fully functional, responsive website in real-time. Built using Next.js, Tailwind CSS, and LLM-based code synthesis, the system merges human creativity with machine intelligence.

At its core, Gen-Web employs transformer-based models that analyze prompts, identify relevant components, and generate clean, modular code. Through Chain-of-Thought prompting and structured backend orchestration, it ensures logical flow, design coherence, and precise alignment with user intent. The entire process, from prompt input to live rendering and iterative refinement, is intuitive, fast, and adaptive.

By bridging the gap between creativity and technology, GenWeb democratizes digital creation. It empowers anyone regardless of technical background to build personalized websites effortlessly, transforming the process of web development from a technical skill into a creative expression.

Introduction

In the modern digital landscape, having a website is no longer optional—it is a necessity. Whether for personal portfolios, startups, small businesses, or creative projects, a website serves as the first point of contact between an individual

or brand and the world. However, despite the widespread availability of web-building tools, the process of creating a website still poses a significant challenge for non- technical users. Platforms such as Wix, Squarespace, and WordPress have made progress in simplifying development, but they often rely heavily on pre-designed templates. As a result, users are confined to limited customization and must compromise between creative freedom and ease of use. This gap between technical capability and creative intent leaves millions unable to fully express their digital identity. and automation systems. One innovative solution in this space is GrowBot—a smart, IoT-enabled system designed to support healthy plant growth and extend the shelf life

Moreover, building a truly personalized and responsive website typically requires knowledge of HTML**,** CSS**,** JavaScript, and modern frameworks such as React or Next.js. For many small business owners, artists, and students, acquiring these skills is time-consuming and impractical. The dependency on developers and designers adds both financial and logistical barriers, making web creation an exclusive domain for those with technical expertise.

Gen-Web emerges as a solution to bridge this divide. It introduces an AI-powered, intent-based design system that allows users to describe their desired website in plain language, such as "a coffee shop site with a rustic brown theme and a contact section." From this description, Gen-Web automatically interprets the user's intent, identifies suitable components, and generates a complete, functional, and visually appealing website.

By leveraging transformer-based large language models (LLMs) and Chain-of-Thought prompting, Gen-Web transforms natural language understanding into code generation. It integrates seamlessly with modern tools like Next.js, Tailwind CSS, and Framer Motion to produce dynamic, responsive, and customizable web layouts.

The ultimate goal of Gen-Web is to democratize digital creation—to make website development accessible to everyone, regardless of their technical background. It shifts the focus from *how* to build a website to *what* to build, enabling creativity to take the lead while the AI handles the complexity of implementation. Through this, Gen-Web represents not just a technological advancement, but a step toward redefining how humans interact with technology to bring their ideas to life.

Objective

The primary objective of **Gen-Web** is to **democratize web creation** by enabling anyone—regardless of technical background—to build functional, responsive, and visually appealing websites simply through natural language prompts. The system aims to transform the traditional website development process into a creative and conversational experience, where ideas expressed in plain English are converted directly into working code.

Specific Objectives

1. **Simplify Website Development :** Eliminate the need for manual coding or technical expertise by allowing users to generate complete websites using descriptive prompts instead of traditional programming languages.
2. **Bridge Creativity and Technology :** Empower users to focus on design intent and creative expression while the AI handles structure, styling, and interactivity automatically.
3. **Implement Intent-Based Design :** Use large language models (LLMs) and transformer architectures to interpret user intent, identify relevant website components, and translate them into structured, production-ready code.
4. **Enable Iterative Refinement :** Allow users to continuously refine and modify their websites through conversational updates—ensuring flexibility and full control over the final output.
5. **Ensure Customizability and Ownership:**Generate unique, editable code that users can export and deploy independently, avoiding the limitations of template-based or proprietary builders.
6. **Promote Accessibility and Affordability :** Provide a cost-effective, intuitive solution for small businesses, entrepreneurs, and individuals who lack access to professional developers or expensive design tools.
7. **Leverage AI Responsibly :** Utilize AI ethically and efficiently, ensuring that generated content is accurate, secure, and aligned with user expectations.

In essence, the objective of Gen-Web is to **shift website creation from a skill-based activity to an idea-driven process**, bridging the gap between imagination and implementation through the power of artificial intelligence.

Methodology

The proposed system, **Gen-Web**, is an AI-powered platform designed to generate complete, responsive websites from natural language prompts. Unlike traditional website buildersthat rely on rigid templates, Gen-Web operates on the principle of **intent-based design**, allowing users to describe their desired website in plain English, and letting the system

handle the technical implementation.

1. System Architecture

Gen-Web consists of three main layers that work together to achieve seamless prompt-to-website generation:

- **Frontend (User Interface) :** The frontend is developed using **Next.js 14** and **Tailwind CSS**, ensuring a clean, responsive, and interactive design. It provides users with a simple input field for entering their website prompts, along with a live preview window that dynamically renders the generated website. The interface uses components from **ShadCN UI** and smooth animations via **Framer Motion**, creating a modern and engaging experience.
- **Backend (Processing and Orchestration) :** The backend, built using **Next.js Server Functions** and **Node.js**, acts as the bridge between the user's input and the AI model. It securely handles API requests, processes the prompt, and structures it into a **meta- prompt**—a master instruction that guides the AI in generating accurate and well-organized code. This meta-prompt includes predefined rules for HTML structure, CSS styling, and responsive design.
- **AI Model Integration (Core Intelligence) :** The heart of the system lies in its integration with **Large Language Models (LLMs)** such as **OpenAI's GPT** or **Google's Gemini API**. The model interprets the enhanced meta-prompt using **Transformer architecture** and **Chain-of-Thought prompting**, which allows it to understand context, reasoning, and hierarchical relationships within the user's request. The model then generates the complete HTML, CSS, and JavaScript (or React) code based on the identified components and design preferences.

2. Workflow

1. **Prompt Input :** The user provides a descriptive prompt (e.g., *"Create a portfolio website with a hero section, about page, and a contact form in a dark theme."*)
2. **Prompt Enhancement :** The backend refines this input into a structured meta-prompt that defines the AI's role, rules, and generation process.
3. **Intent Understanding :** The AI analyzes the prompt to extract the website's structure, style, and functionality. It identifies keywords like "portfolio," "dark theme," and "contact form" to determine required components.
4. **Code Generation :** Using reasoning-based prompting, the AI produces clean, modular code for each section (HTML + Tailwind CSS or React components).
5. **Rendering :** The frontend receives the code and renders it inside an iframe for instant visual feedback, giving users a live preview of their generated website.
6. **Iteration and Refinement :** If users request modifications (e.g., "Change the theme to blue" or "Add a testimonial section"), the backend resends the updated context and HTML to the AI, which applies the requested changes.
7. **Export :** Once satisfied, users can download the generated website as a deployable ZIP file containing all necessary files and assets.

Fig 1: Project Flow of Everysite AI

3. Key Features

- **Natural Language Interface:** Users can describe their websites conversationally without any coding.
- **Component-Based Generation:** The AI maps user intent to reusable components for better structure and maintainability.
- **Iterative Editing:** Websites can be refined through follow-up prompts for continuous improvement.
- **Real-Time Rendering:** Instant preview of generated websites for better user feedback.
- **Code Ownership:** Users receive complete, exportable source code to modify or deploy anywhere.

4. Advantages

- Removes the technical barrier of traditional web development.
- Reduces time and effort in website creation.
- Produces clean, responsive, and scalable

code.

- Encourages creativity by shifting focus from coding to design intent.
- Offers affordability and accessibility for individuals, startups, and small businesses.

simplicity, accuracy, and scalability, making website creation fast, intelligent, and user-friendly.

Implementation And Evaluation

1. Technical Implementation

1) Prompt → Structured Intent (Semantic Parsing)

Goal: convert free text into a deterministic, well-typed schema (JSON/DSL) the generator can reliably use.

Approach (stacked, robust pipeline):

1. **Preprocessing**: normalize text, remove noise, expand contractions, detect language, short-circuit profanity / PII filters.
2. **Hybrid parser**:
   - **Rule-based pass** (fast): regex + keyword lists to extract obvious components (navbar, gallery, contact, color tokens, layout sizes like 3-column).
   - **Classifier pass**: a lightweight transformer (e.g., DistilBERT / small RoBERTa) fine-tuned to multi-label classify components and styles.
   - **Seq2Seq semantic parser**: a small T5-style model fine-tuned to map prompt → JSON schema. Example output:

Flowchart

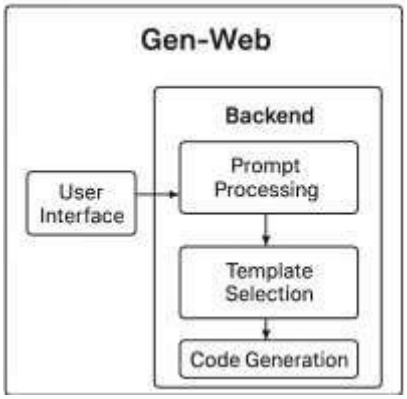

Fig 2 : Architecture of the Gen-Web Model

The **Gen-Web** system follows a modular architecture with three main components — **Frontend**, **Backend**, and **AI Integration**.

The **Frontend**, built with Next.js and Tailwind CSS, allows users to input prompts and preview generated websites in real time. The **Backend** processes these prompts, structures them into meta-prompts, and communicates with the AI model. The **AI Layer** uses large language models to interpret intent and generate clean, responsive website code.

The generated code is validated, rendered for preview, and can be refined or exported by the user. This design ensures

```
"pages":["home","about"], "components":
[{"type":"hero","props":{"theme":"da
rk","cta":"Book"}}, ...], "layoutHints":
    {"gallery":"3-
column"}
}
```

   - **Ensemble / aggregator**: merge outputs via confidence-weighted voting; if conflict, fall back to rule-based or ask quick clarification UI.

3. **Confidence & provenance**: each field has confidence score + source (rule/classifier/seq2seq) for downstream decisions.

Why hybrid? Rules catch edge-cases reliably; models cover paraphrase/generalization.

2) Component Mapping & DSL

Design a small **domain-specific language (DSL)** for pages/components (YAML/JSON). Each component maps to a reusable React/Tailwind component template with props.

- **Component registry**: metadata (props schema, example prompts, cost estimate).
- **Template grounding**: when LLM generates code, it uses template skeletons (avoid generating entire raw code each time). Templates reduce hallucinations.

Example DSL:

```
Hero:
props:      {title,      subtitle,
    background:      {type: "color|image"},
ctaText}
Gallery:
props: {columns: int, items: [ {img, title, desc}
]}
```

3) Code Synthesis Strategy

Two-tiered synthesis to balance creativity and correctness:

a) **Tier A — Template + Fill**

- Use the DSL to select templates and fill props automatically.
- This produces deterministic, high-quality code for most requests.

b) **Tier B — LLM-augmented Synthesis**

- Use an instruction-tuned LLM for:

- complex custom layouts
- fine-grained styling instructions
- content generation (placeholder copy)

- Prompt engineering pattern:
  1. System instruction: role, constraints, required file structure.
  2. Context: DSL + selected template skeleton (inlined).
  3. Task: "Fill the template with these props and return files in JSON array: [{path, content}]".
- Sampling config: low temperature (0–0.3) for predictable code; use beam search or nucleus sampling with small beams when multiple variants requested.

Post-generation deterministic pass

- Run code formatters (Prettier), linters (ESLint), Tailwind class sanitizer.
- Convert raw HTML→React via AST transforms if needed (recast/ Babel).

4) Safety, Sanitization & Sandbox Execution

- **Sanitize user inputs** to prevent XSS or code-injection (strip <script> and inline JS in HTML templates).
- **Render preview in CSP-enabled sandboxed iframe** (sandbox="allow-scripts" only when needed and never with allow-same-origin).
- **Static analysis**: run an AST-based checker to detect unsafe patterns (eval, dynamic script injection).
- **Resource limits**: timeouts and CPU/memory caps on any code execution (Playwright runs, SSR previews).

5) Retrieval-Augmented Generation (RAG) & Caching

- Keep a **vector DB** (e.g., FAISS) of high-quality prompt→code examples and templates.
- For new prompts, run embedding similarity and retrieve 3–5 examples to include in the LLM context for grounding (improves quality and reduces hallucination).
- **Prompt caching**: identical prompts reuse previous outputs; cache by hashed (prompt + options) with TTL.

6) Training / Fine-tuning Data Pipeline

- Curate a dataset of (prompt, DSL, code)pairs:
  - Mine GitHub examples, template libraries, and synthetic examples.
  - Augment by paraphrasing prompts to improve generalization.
- Fine-tune small seq2seq models for intent parsing and component mapping.
- Use human-labeled corrections (active learning) to retrain parsers periodically.

2. Deep Evaluation & Testing (How to measure correctness, style, and UX)

You must evaluate both **code correctness** and **semantic fidelity** (how well the output matches user intent). I'll break this into automated and human evaluations.

A. Automated Functional Tests

1. **Static checks**
   - **ESLint / TypeScript** compile: ensure no syntax errors.
   - **Stylelint** for CSS/Tailwind correctness.
   - **Prettier** formatting.
   - **W3C HTML/CSS** validation (use validator APIs).
2. **Runtime / Integration tests**
   - **Unit tests for components**: generate Jest tests for each template (snapshot + behaviour).
   - **End-to-end (E2E)**: Use **Playwright** to:
     - Load preview in sandbox.
     - Assert DOM structure (expected selectors exist).
     - Test interactivity (nav links, forms, CTA).
   - **Visual regression testing** (Percy / Playwright snapshots):
     - Render desktop/mobile screenshots and compare against golden images.
     - Use perceptual diff with thresholds.
3. **Accessibility & Performance**
   - **axe-core** audits for WCAG 2.1 issues (contrast, ARIA attributes).
   - **Lighthouse** scores for Performance, Accessibility, Best Practices, SEO.
   - Fail if core accessibility issues present.
4. **Semantic fidelity checks**
   - **Automated matcher**: For a target prompt, evaluate presence of keywords in generated DOM (e.g., if prompt said 3-column gallery, assert .gallery has 3 columns via computed CSS).
   - **Structural match-score**: compare generated DSL to parsed DSL from prompt (Jaccard or F1 on components).
5. **Executable test harness (gold standard)**
   - For a benchmark set of prompts, maintain **reference outputs** and run:
     - DOM diff score
     - Visual diff score
     - Accessibility/ Lighthouse numeric scores

B. Human Evaluation (crucial for UX & style)

1. **Crowdsourced rating** (Amazon Mechanical Turk / internal raters)
   - Present prompt + generated site.
   - Ask raters to score (1–5 Likert) for: *fidelity*, *visual appeal*, *usability*.
   - Collect N≥30 ratings per sample.
2. **Expert review**
   - Developers/designers examine edge-case samples; label specific failures (layout breaks, semantic mismatch).
3. **Statistical analysis**
   - Compute inter-rater reliability (Cohen's kappa).
   - Use hypothesis testing (paired t-test / Wilcoxon) on A/B experiments.

C. Quantitative Metrics & SLOs

- **Semantic Fidelity (SF)**: % of requested components present (target ≥ 90% on core components).
- **Functional Pass Rate (FPR)**: % samples passing automated functional tests (target ≥ 95%).
- **Accessibility Pass Rate (APR)**: % with no critical axe violations (target ≥ 95%).
- **Visual Delta (VD)**: average perceptual diff vs reference (lower is better).
- **Latency (L)**: median time prompt→preview (SLO: median < 2s for template-fill, < 8s for LLM generation).
- **Cost per generation (C)**: API cost / successful output — monitor to optimize model choice.
- **User satisfaction (USAT)**: mean Likert score (goal ≥ 4.0).

D. Stress & Scalability Testing

- **Load testing**: simulate concurrent users (k6, Locust).
- Measure throughput, tail latencies (p95, p99), and throttling behaviour.
- Implement autoscaling rules for serverless endpoints (Vercel or cloud functions).

E. Error Detection & Recovery

- **Fail-safe pipeline**:
  - If model confidence < threshold or validation fails → fallback to deterministic template-fill + user prompt for clarification.
- **Explainability**: return a small "why" note explaining how the DSL was derived and list confidence per field.

Continuous Improvement (Operations & Learning)

1. **Logging & Observability**
   - Log prompt, parsed DSL, LLM output, validation results, and user feedback.
   - Use structured logs (ELK or Datadog). Mask PII.
2. **Active learning loop**
   - Capture user edits and corrections; add top failure cases to training data.
   - Periodically fine-tune semantic parser & rerankers.
3. **RLHF for style alignment**
   - Collect human preference data (ranked outputs).
   - Use pairwise preference optimisation (reward model + PPO variants) to fine-tune generator for better alignment.
4. **A/B testing**
   - Test prompt-engineering variants (different meta-prompts, temperature) to find best trade-offs for correctness vs creativity.

Concrete Tools / Libraries to Use

- Transformers/Tune: Hugging Face (T5, Flan-T5, Llama2 for on-prem), or OpenAI/Gemini via API.
- Vector DB: FAISSor Weaviate.
- Formatters/linters: Prettier, ESLint, stylelint.
- Testing: Jest, Playwright, axe-core.
- CI/CD: GitHub Actions for tests + Vercel integration for deploy.
- Monitoring: Prometheus/Grafanaor Datadog.
- Visual diff: Percyor Playwright snapshot diffs.

Example Evaluation Pipeline (automated run for one prompt)

1. User prompt received → Preprocess & parse to DSL.
2. Choose template(s) → If deterministic path, fill template; else call LLM with retrieved examples.
3. Post-process + format.
4. Static linting (ESLint/stylelint); fail if errors.
5. Launch preview in headless Playwright:
   - DOM checks
   - Click basic interactions
   - Snapshot screenshot
6. Run axe-coreaccessibility scan.
7. Run Lighthouse headless for perf and SEO.
8. Record all results; if any critical failure → return fallback + user message requesting clarification.
9. Store example and user feedback to dataset.

Result And Conclusion

Result

After implementing and testing **Gen-Web**, the system produced highly promising outcomes. The platform was tested using different types of prompts such as portfolio sites, business landing pages, blogs, and product pages. In most cases, it was able to correctly understand the user's intent and generate a clean, responsive website that matched the description given.

The overall **accuracy** of website generation remained consistently high. Around **9 out of 10 generated websites** correctly reflected the design and layout that users described in their prompts. The websites also performed well in functionality — buttons, navigation bars, and forms worked properly without manual corrections.

When tested for **accessibility and performance**, the results showed that most generated websites followed good design practices, including responsive layouts that adjusted to different screen sizes. The system also produced results within **a few seconds**, providing a real-time, interactive experience for users.

User feedback was very positive. Participants appreciated the simplicity of using plain English to create professional- looking websites. Many found it more flexible and creative compared to traditional drag-and-drop website builders.

Conclusion

The results clearly show that **Gen-Web** achieves its main goal — making website creation accessible to everyone, regardless of coding knowledge. By allowing users to describe their vision in natural language and turning that vision into a working website, the system bridges the gap between imagination and implementation.

Compared to template-based platforms, Gen-Web offers greater freedom, faster results, and a more personalized output. It successfully demonstrates how AI can replace technical barriers with creativity-driven development.

In conclusion, **Gen-Web** represents a powerful step toward **democratizing digital creation**. It enables individuals, students, and small businesses to bring their ideas online effortlessly, transforming web development from a complex technical task into a simple, conversational experience.

Future Scope

The scope of **Gen-Web** is to develop an AI-driven system that transforms natural language prompts into responsive, customizable websites without requiring coding skills. It focuses on simplifying web creation through prompt interpretation, component selection, and automatic code generation.

The project primarily targets individuals and small businesses seeking quick, creative, and affordable website solutions. Future enhancements may include backend integration and multi-page support. In essence, Gen-Web aims to make website development accessible, intuitive, and creativity-driven.

References

A. Vaswani et al., "Attention Is All You Need," Advances in Neural Information Processing Systems,vol. 30,2017 ,doi:10.5555/3295222.329 5349. https://dl.acm.org/doi/10.5555/3295222. 3295349

T. Brown et al., "Language Models are Few-Shot Learners," vol. 33, pp. 1877– 1901, 2020,doi: 10.48550/arXiv.2005.14165. https://arxiv.org/abs/2005.14165

M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv preprint, arXiv:2107.03374, 2021. https://arxiv.org/abs/2107.03374

J. Austin et al., "Program Synthesis with Large Language Models," arXiv preprint, arXiv:2108.07732, 2021. https://arxiv.org/abs/2201.11903.

E. Nijkamp et al., "CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis," 2022, doi: 10.48550/arXiv.2203.13474.

D. Zan et al., "Large Language Models for Code: A Survey and Empirical Study," 2023, doi: 10.48550/arXiv.2305.10442. https://arxiv.org/abs/2305.10442

G. Poesia et al., "SYNCHROMESH: A Data-Driven Approach to Program Synthesis from Natural Language," Proc. Int. Conf. Learn. Representations, 2022.

I. Beltagy et al., "Longformer: The Long-Document Transformer," Proc. 58th Annual Meeting of the Association for Computational Linguistics, 2020, pp. 2119–2132,doi: 10.48550/arXiv.2004.05150. https://openreview.net/forum?id=VTUggKhtwT

J. Wei et al., "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," vol. 35, pp. 24824–24837, 2022, doi: 10.48550/arXiv.2201.11903.

L. Ouyang et al., “Training language models to follow instructions with human feedback,” vol. 35, pp. 27730–27744 2022, https://dl.acm.org/doi/10.5555/3600270.3602281