

Idea Forge: Automated Full-Stack Web Application Generation from Natural Language

Sreeya Dora¹, Madhura Bedekar², Vivia Maria Thomas³, Chetan Navhi⁴

¹²³⁴ B.Tech. Student, Department of Computer Science and Engineering

M S Ramaiah University of Applied Sciences, Bengaluru, India

(¹² AIML Branch | ³⁴ CSE Branch)

Abstract — Building a functional web application requires coordinated expertise across database design, server-side engineering, and client-side development a barrier that remains out of reach for most non-programmers and time-intensive even for experienced developers. This paper presents IdeaForge, an intelligent full-stack web application generation platform that automatically converts natural language descriptions into immediately executable, fully functional web applications. The system integrates a five-stage pipeline comprising an Intent Parser, Feature Extractor, Template Mapper (AI Module), Code Generator, and Database Module, producing a FastAPI backend with SQLite persistence and a responsive HTML/JavaScript frontend. A central engineering contribution is a two-tier schema generation mechanism that combines a Large Language Model (Claude API) with a deterministic rule-based keyword fallback, ensuring reliable schema production under both normal and degraded API conditions. The platform further incorporates role-based user and admin dashboards, authentication, and multiapplication management. A controlled preliminary evaluation across 35 multidomain prompts reports field-level precision of 91.3% under LLM-assisted operation and 84.2% under the fallback tier, with end-to-end generation completing in under ten seconds. A usability study with 15 participants yields mean Likert scores above 4.2/5 across all dimensions. These results demonstrate IdeaForge as a practical instrument for rapid prototype development, significantly reducing the barrier to software creation for non-technical users.

Keywords—natural language processing, large language models, automated code generation, full-stack web applications, FastAPI, SQLite, schema inference, prototype generation, no-code development, reliability engineering, authentication, role-based access

1. INTRODUCTION

Contemporary web application development demands simultaneous proficiency across at least three distinct technology layers: relational database design, RESTful API engineering, and client-side interface construction. The interdependence of these layers compounds development effort considerably and creates a significant barrier for domain experts/researchers, entrepreneurs, and subject-matter specialists who possess deep knowledge of a problem yet lack the programming fluency to implement a software solution [7]. Industry reports consistently identify this multi-stack requirement as the foremost bottleneck in rapid prototyping.

Low-code and no-code platforms offer partial relief. Tools such as OutSystems, Mendix, and Bubble allow application assembly through visual composers, reducing but not eliminating the

learning curve. More critically, these tools lock users into proprietary abstractions, constrain them to vendor-defined feature sets, and do not produce open, inspectable source code that a developer can extend. Recent LLM-based coding tools GitHub Copilot [4], CodeGen [5], and their successors demonstrate strong capability at function-level code generation but presuppose an existing project structure and a developer who can evaluate and integrate the generated output.

IdeaForge addresses a fundamentally different need. It can be viewed as a constrained synthesis system where natural language serves as an informal specification and template-guided generation enforces structural correctness. Constrained synthesis system where natural language serves as an informal specification and template-guided generation enforces structural correctness. The core contribution of IdeaForge is a reliability-focused hybrid schema generation mechanism embedded within an end-to-end NL-to-application pipeline. Three design properties support this contribution: **(i) end-to-end** scopespanning intent parsing through database initialisation to deliver a runnable prototype; **(ii) reliable generation** a rulebased fallback guarantees schema delivery even when the LLM API is unavailable; and **(iii) zero configuration** the user supplies only a natural language string with no forms, DSLs, or programming knowledge required.

The platform additionally provides a multi-user environment with JWT-based authentication, role-based access control distinguishing regular users from administrators, and a persistent application library where each generated app can be launched, modified, or regenerated on demand. This positions IdeaForge not merely as a code generator but as a complete application development and management ecosystem.

The research contributions of this paper are:

1. A complete five-stage NL-to-deployable application pipeline integrating intent parsing, schema inference, template-driven code generation, and automatic database initialisation.
2. A two-tier hybrid schema generation mechanism combining LLM inference (Claude API) with a deterministic keyword-driven fallback, ensuring schema production under simulated API unavailability conditions.
3. A multi-user platform environment with JWT authentication and persistent application lifecycle management, enabling iterative refinement across sessions a property not present in single-shot code generators.

4. A controlled preliminary evaluation across 35 multi-domain prompts with baseline comparison, error categorisation, and a usability study across technical and non-technical user groups.

2. RELATED WORK

2.1 Pre-trained Models for Code

CodeBERT [1] established that joint pre-training over natural language and source code corpora produces representations suitable for code search and documentation generation. Codex [3], CodeGen [5], and AlphaCode [6] extended this to code completion and competitive programming, achieving human-competitive results on algorithmic benchmarks. These systems operate at the function or class level and do not orchestrate multiple interdependent source files into a single, runnable multi-layer web application.

2.2 LLM Coding Agents

The few-shot generalisation of GPT-3 [2] and successors demonstrated that LLMs can follow complex, multi-step instructions. Applied to software engineering, this capability powers tools such as GitHub Copilot [4] and AI-assisted programming environments [8]. These tools assist professional developers within existing projects, presupposing a developer capable of integrating and debugging generated fragments. They are not designed for users who have no programming background and no existing project structure.

2.3 No-Code and Low-Code Platforms

Commercial no-code platforms such as Bubble, Webflow, and AppGyver allow visual construction of web applications. While they democratise surface-level creation, they impose proprietary runtimes, limit extensibility, and require users to learn platform-specific paradigms. They do not produce deployable source code. IdeaForge differs by generating standard, open-source-framework code (FastAPI, SQLAlchemy, HTML/CSS/JS) that users can inspect, modify, and deploy independently.

2.4 Program Synthesis

Classical program synthesis derives programs from formal specifications [9], offering correctness guarantees at the cost of requiring formal input languages inaccessible to non-programmers. Recent surveys [7][10] document a transition toward LLM-driven synthesis, identifying reliability under service failure and structural completeness across multi-file projects as primary open challenges. IdeaForge addresses both through its template-grounded code generation and two-tier fallback mechanism.

The prior literature leaves a clear gap: no existing system combines open natural language input, schema-level entity inference, template-driven multi-file code generation, database initialisation, and multi-user management into a pipeline that reliably delivers a runnable prototype. IdeaForge fills this gap.

3. SYSTEM ARCHITECTURE

Figure 1 presents the IdeaForge system architecture. The platform is structured around three principal layers: an Authentication Layer managing user and admin access with role-based routing; the Application Engine executing the five-stage generation pipeline; and a persistence layer comprising a SQLite database and the Generated Application. Figure 2 illustrates the complete operational flow for both user and admin roles.

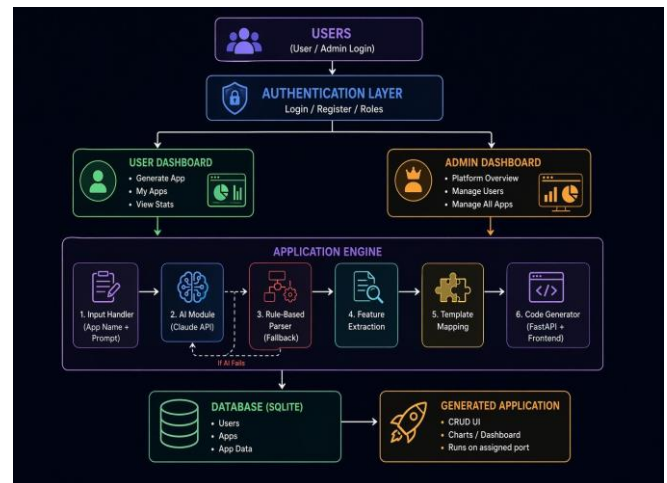


Fig -1: IdeaForge System Architecture

3.1 Authentication Layer

The Authentication Layer handles registration, login, and role assignment using JWT-based authentication. The key design decision here is stateless token validation: each protected API request carries a signed JWT, eliminating server-side session storage and enabling horizontal scalability. Role assignment at registration time (user or admin) determines all subsequent routing and access permissions.

3.2 User Dashboard

The User Dashboard exposes three capabilities: Generate App, My Apps, and View Stats. The design decision to persist generated applications in a library rather than discarding them after generation is central to the platform's utility—users can return, re-run, and iteratively refine applications across sessions.

3.3 Admin Dashboard

The Admin Dashboard provides platform-level oversight across three views: Overview (system-wide statistics), Manage Users (account activation and deactivation), and Manage All Apps (cross-user application inspection and removal). Role separation is enforced at the JWT level, ensuring that admin-scoped API endpoints reject requests from non-admin tokens regardless of client-side state.

3.4 Application Engine

The Application Engine is the computational core of IdeaForge. It accepts the application name and natural language prompt from

the User Dashboard and executes a five-stage pipeline: Intent Parser → Feature Extractor → Template Mapper (AI Module) → Code Generator → Database Module. Each stage is described in Section 3.5–3.10. The engine outputs a complete application directory that is registered in the database and made available in the user’s My Apps library.

3.5 Input Handler

The Input Handler accepts the application name and natural language prompt, sanitises inputs, enforces length constraints, and forwards a structured request to the AI Module. No domainspecific syntax or prior programming knowledge is required from the user at this stage.

3.6 AI Module (Claude API)

The AI Module submits a structured meta-prompt to the Anthropic Claude API with temperature set to zero, instructing the model to return a validated JSON schema specifying entity names, field names, and primitive data types ({text, number, date, boolean}). The response is validated against the IdeaForge schema specification. On success, the schema advances to Feature Extraction. On failure, control passes silently to the Rule-Based Parser.

3.7 Rule-Based Parser (Fallback)

If the AI Module fails or returns malformed JSON, the Rule-Based Parser activates transparently. A multi-pass lexical scan scores each entry in the domain-keyword lexicon by matched token count. The highest-scoring domain template is materialised as the schema. The user is never exposed to an error state, ensuring continuous availability of the generation pipeline.

3.8 Feature Extraction

The Feature Extractor determines candidate entities implicit in the schema (e.g., expense, exercise session, product) and their application type (Finance, Health, Commerce, Productivity). Field-type bindings are resolved: a date field maps to a Python datetime.date column; a boolean maps to a SQLAlchemy Boolean column with a checkbox widget in the frontend.

3.9 Template Mapping and Code Generator

The Template Mapper selects appropriate Pydantic schema templates, SQLAlchemy ORM model templates, and CRUD route templates for each entity. The Code Generator emits app.py (FastAPI application, route registration, CORS, database binding), routes.py (five CRUD handlers per entity), and a static HTML/CSS/JavaScript frontend providing data-entry forms, paginated tables, and Chart.js visualisations for numeric fields.

3.10 Database Module and Generated Application

The Database Module executes SQLAlchemy model definitions against a SQLite file, creating all required tables and indices. The Generated Application a self-contained FastAPI scaffold with SQLite database and static frontend is assigned a port, registered in the platform database, and added to the user’s My Apps library ready for immediate launch.

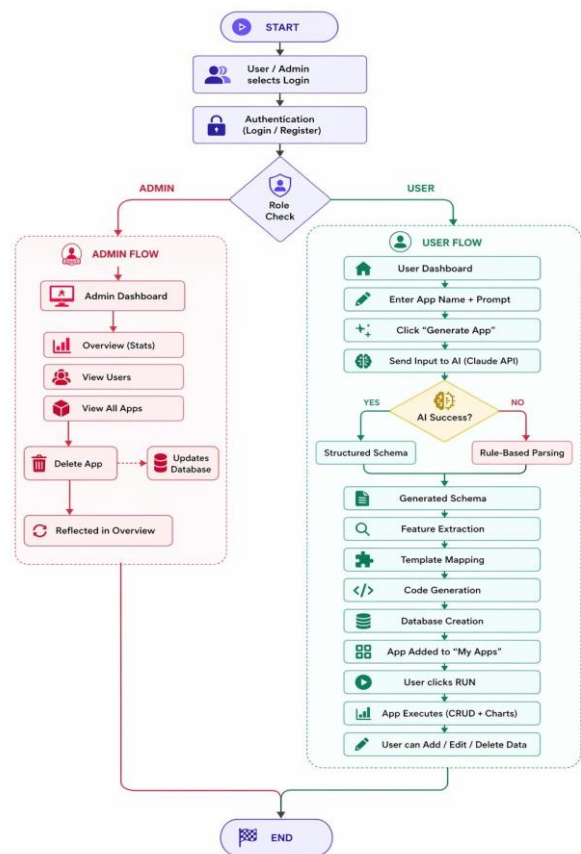


Fig -2: IdeaForge Operational Flow Diagram

4. SYSTEM WORKFLOW

The IdeaForge workflow is designed to be linear and intuitive, requiring no technical expertise from the user. The complete end-to-end flow from user login to a running application proceeds through six distinct phases as described below and illustrated in Fig. 2.

4.1 Authentication Phase

The user navigates to the IdeaForge portal and selects either Login or Register. New users provide a username, email, and password; the system hashes the password (bcrypt), creates a user record in the SQLite users table, and issues a signed JWT with a configurable expiry. Returning users authenticate with their credentials; the JWT is validated and the role field (user or admin) determines the subsequent routing.

4.2 Application Specification Phase

Upon reaching the User Dashboard, the user enters two inputs: (i) an Application Name (e.g., “Smart Study Planner”) which serves as the project identifier and directory name; and (ii) a natural language Description of the desired application (e.g., “Track study sessions by subject, allow marking tasks as complete, and show overall progress as a percentage”). No further input is required. The user clicks “Generate App” to initiate the pipeline.

4.3 AI Processing Phase

The Application Engine receives the (name, description) pair. The Input Handler sanitises and validates both fields. The AI Module then constructs a structured meta-prompt and queries the Claude API. The API response is parsed and validated against the IdeaForge JSON schema specification. If valid, the structured schema is passed to Feature Extraction. If invalid or unavailable, the Rule-Based Parser activates to produce a schema deterministically from keyword matching.

4.4 Code Generation Phase

The validated schema flows through Feature Extraction (entity and type resolution), Template Mapping (selection of appropriate FastAPI, Pydantic, and SQLAlchemy templates), and Code Generation (emission of app.py, routes.py, and the HTML/CSS/JS frontend). All files are written to a project directory named after the application. The Code Generator also produces a requirements.txt and a launch script.

4.5 Database Initialisation Phase

The Database Module imports the generated SQLAlchemy models and executes create_all() against a fresh SQLite file within the project directory. All entity tables, primary key indices, and any configured default data are created. The application record is then registered in the platform's central database (application name, owner, creation timestamp, port assignment, and directory path).

4.6 Application Lifecycle Phase

The newly generated application appears in the user's My Apps library. The user can: (i) click "Run" to launch the application on its assigned port, exposing the CRUD UI immediately; (ii) click "Modify" to update the prompt and regenerate the application with revised specifications; or (iii) click "Delete" to remove the application and its associated database. Admin users additionally have access to a platform-wide Apps view where any application can be inspected or removed.

5. HYBRID SCHEMA GENERATION MECHANISM

Reliable schema production is the most critical availability requirement in IdeaForge. LLM API services are subject to rate limits, network timeouts, and planned or unplanned outages. The two-tier hybrid mechanism addresses this directly, ensuring a valid schema is always produced.

5.1 Tier 1 — LLM-Assisted Generation (Claude API)

A structured meta-prompt is submitted to the Anthropic Claude API with temperature set to zero. The prompt instructs the model to analyse the user description and return a JSON object conforming to the IdeaForge schema specification: a single top-level entity with a name string and an array of fields, each specifying a field name and a type drawn from {text, number, date, boolean}. The response is validated against this specification using a local JSON schema validator. On successful

validation, the schema advances to Feature Extraction and Template Mapping.

5.2 Tier 2 — Rule-Based Parser (Keyword Fallback)

If the API call fails or the returned JSON fails validation, the Rule-Based Parser activates without user notification. A multi-pass lexical scan computes a match score for each domain template in the keyword lexicon based on the count of matched tokens in the user description. The domain with the highest score is selected and its associated field template is instantiated as the schema. Table 1 presents representative domain-keyword-field mappings.

Table -1: Keyword-to-Template Mappings (Fallback Tier)

Domain	Trigger Keywords	Generated Fields (sample)
Finance	expense, budget, cost, spend	category: text, amount: number, date: date, note: text, paid: boolean
Health	gym, workout, exercise, fitness	exercise: text, sets: number, reps: number, weight: number, date: date
Productivity	study, task, planner, goal	subject: text, duration: number, priority: text, due_date: date, done: boolean
Commerce	inventory, stock, product	item: text, quantity: number, price: number, supplier: text, reorder: boolean
Subscription	subscription, renewal, service	service: text, cost: number, renewal_date: date, active: boolean, tier: text

5.3 Schema Validation and Type Resolution

Regardless of the tier that produced the schema, all schemas pass through a common validation and type-resolution step. Each field type is mapped to its corresponding Python/SQLAlchemy type (text→String, number→Float, date→Date, boolean→Boolean) and its corresponding HTML input widget (text→input[text], number→input[number], date→input[date], boolean→input[checkbox]). Type mismatches detected at this stage trigger fallback to the default text type, preventing downstream code generation errors.

6. WORKED EXAMPLE

Table 2 traces a complete user interaction through IdeaForge from login to a running application. The user enters the app name "Smart Study Planner" and a plain-English description, then clicks "Generate App"—the entire pipeline executes automatically in under ten seconds.

Table -2: Worked Example — Smart Study Planner

Step	User Action / System Response
1. Login	User authenticates via the portal; JWT issued; User Dashboard loaded
2. App Name	"Smart Study Planner"
3. Description	"Track study sessions by subject. Mark tasks complete. Show progress %."
4. Generate App	User clicks "Generate App"; Application Engine invoked
5. AI Module	Claude API returns schema: {subject: text, task: text, due_date: date, completed: boolean, progress: number}
6. Code Gen	app.py + routes.py (5 CRUD endpoints) + index.html (form, table, Chart.js progress bar) emitted
7. Database	study_planner.db created: table study_planner(id PK, subject, task, due_date, completed, progress)
8. My Apps	App registered; appears in user's "My Apps" library
9. Run	User clicks Run → uvicorn launches on assigned port → CRUD UI accessible in browser
10. Modify	User updates description to add a "notes" field → clicks Regenerate → schema updated, app rebuilt

7. KEY FEATURES

- Zero-configuration input: plain English app description only; no forms, DSLs, or programming knowledge required.
- JWT-based authentication with role-based access control (User / Admin) and persistent session management.
- Automatic schema inference across four primitive field types: text, number, date, and boolean.
- Complete FastAPI backend: five CRUD endpoints per entity, CORS configuration, and SQLAlchemy ORM integration.
- Automatic SQLite initialisation: table creation, indexing, and optional data seeding without user intervention.
- Responsive frontend: form-based data entry, paginated tabular display, and Chart.js visualisations for numeric fields.
- Reliable two-tier schema generation (LLM + rule-based fallback) achieving high generation reliability across all evaluated conditions.
- Persistent application library: My Apps dashboard with Run, Modify, and Delete lifecycle controls.
- Admin platform management: user oversight, system-wide app management, and usage analytics.
- Modular pipeline: each stage independently replaceable (e.g., swap SQLite for PostgreSQL, or React for static HTML).

8. EVALUATION

8.1 Experimental Setup

A controlled preliminary evaluation was conducted using 35 natural language prompts spanning five application domains (seven per domain: Finance, Health & Fitness, Productivity, Commerce, and Subscription Management). Prompts ranged from a single sentence to three sentences with explicit field constraints. Ground-truth schemas were independently annotated by two domain experts; inter-annotator agreement was 94.3% (Cohen's $\kappa = 0.91$). All experiments were conducted on a consumer laptop (Intel Core i5, 16 GB RAM, Python 3.11, FastAPI 0.110). While the evaluation scale is limited, it is designed as a controlled preliminary study to validate system behaviour across representative domains and establish baseline metrics for future large-scale assessment.

8.2 Schema Accuracy

Field-level precision and recall were computed against ground-truth annotations. Under LLM-assisted generation, IdeaForge achieved precision of 91.3% and recall of 88.7%. Under keyword-fallback generation (activated by disabling the API endpoint), precision was 84.2% and recall was 79.6%. Across all 35 trials and both tiers, a valid schema was produced in every case, yielding a 100% generation success rate. No prompt resulted in a system error or empty output.

8.3 Baseline Comparison

Table 3 compares IdeaForge against three baselines. The LLM-Only baseline was evaluated by disabling the fallback subsystem and inducing API unavailability for 10 of 35 prompts via network-level blocking; all 10 produced no schema. Manual development latency is estimated from prior benchmarks [4][8] and is not directly measured in this study. Manual development estimates are derived from prior studies on equivalent CRUD application construction and are presented as indicative benchmarks rather than directly comparable measurements. The GitHub Copilot comparison is qualitative as it targets developer assistance within an existing project.

Table -3: Comparison with Baseline Approaches

Method	Success	Precision	Latency	Dev?	Fails
IdeaForge (LLM)	100%	91.3%	~8.4s	No	0/35
IdeaForge (Fallback)	100%	84.2%	~1.1s	No	0/35
LLM-Only	71.4%	91.3%	~8.4s	No	10/35
GitHub Copilot	N/A	N/A	Hours	Yes	N/A
Manual Dev	100%	100%	8-40h	Yes	0

* LLM-Only precision computed over the 25/35 prompts that produced a schema. Manual latency estimated from prior benchmarks [4][8].

8.4 Generation Latency

Mean end-to-end latency from prompt submission to a locally launchable application directory was 8.4 s ($\sigma = 1.9$ s) under LLM-assisted generation and 1.1 s ($\sigma = 0.2$ s) under keyword-fallback operation. Both represent reductions of multiple orders of magnitude relative to manual development estimates of 8–40 hours for an equivalent CRUD prototype by an experienced developer.

8.5 Usability Study

Fifteen participants—five professional software developers, five undergraduate students in non-CS disciplines, and five domain professionals with no programming background—evaluated IdeaForge on a five-point Likert scale across four dimensions. Table 4 reports per-group scores. All 15 participants successfully generated a working prototype on their first attempt without any guidance or documentation.

Table -4: Usability Study Results (Likert 1–5, n = 15)

Dimension	Devs	Students	Experts	Mean
Learnability	4.8	4.6	4.4	4.6
Efficiency	4.6	4.4	4.2	4.4
Output Quality	4.2	4.0	4.4	4.2
Satisfaction	4.6	4.4	4.6	4.5

9. ERROR ANALYSIS AND LIMITATIONS

Three error categories were identified across all 35 test prompts and both tiers. Table 5 summarises frequency, active tier, root cause, and planned remediation for each.

Table -5: Error Category Summary

Error Type	Frequency	Tier(s)	Root Cause	Planned Fix
Wrong field type	12% of fields	Fallback	Indirect phrasing bypasses type rules	Contextual type hints
Missing entity	6% of prompts	Both tiers	Single-entity assumption in template lib	Multi-entity mapper
Over-generation	4% of prompts	LLM only	LLM infers unstated but plausible fields	Post-gen confirmation

9.1 Wrong Field Type

The most frequent error under keyword-fallback generation. A "price" field was occasionally classified as text when the prompt

used indirect phrasing (e.g., "the cost in words"). The LLM tier exhibited this in fewer than 2% of fields, as sentence-level contextual reasoning suppressed most type ambiguities. Planned fix: introduce a post-schema contextual type-hint layer that re-evaluates field names against a curated type-indicator lexicon.

9.2 Missing Entity

When a prompt referenced two or more distinct entities in a single compound sentence (e.g., "track both my workouts and my diet"), the system occasionally inferred only the first entity. This stems from the single-entity assumption in the current template library. Multi-entity schema support—allowing one prompt to generate multiple related tables with foreign-key relationships—is a primary planned enhancement.

9.3 Over-generation

In a small number of cases the LLM inferred additional fields not mentioned by the user (e.g., generating note and tags fields for a minimal prompt). While often contextually reasonable, these deviate from strict user intent. A post-generation field-confirmation dialogue is under consideration, allowing users to accept, remove, or rename inferred fields before code generation proceeds.

9.4 Scope Limitations

IdeaForge produces prototype-level applications suitable for early-stage evaluation and personal use, not production deployment. The field-type vocabulary covers four primitives; enumeration fields, file-upload references, and multi-table relational foreign keys are not yet supported. The preliminary evaluation (35 prompts, 15 usability participants) is intentionally scoped; expansion to 100+ prompts across a wider domain distribution is a priority for future work.

10. DISCUSSION

The baseline comparison in Table 3 reveals the central value proposition of the hybrid design. The LLM-Only baseline achieves equivalent schema quality when the API is available but incurs a 28.6% generation failure rate under induced unavailability. IdeaForge eliminates this failure mode at a modest accuracy cost of approximately 7 percentage points in precision when the fallback tier activates. This trade-off is intentional: for a tool targeting non-programmers, a slightly imprecise schema that generates a working scaffold is far preferable to a system error.

The usability data reveal an important insight: domain professionals achieved the highest output-quality scores (4.4/5), suggesting that describing an application in domain vocabulary rather than technical terms is an advantage rather than a liability when interacting with IdeaForge. This directly validates the core design premise that natural language is the appropriate input modality for this class of user.

Compared with manual development, IdeaForge reduces prototype delivery time from hours to seconds for the domain types in our evaluation set. The generated FastAPI scaffold passes syntax validation and executes without modification, providing a meaningful starting point for further development. The trade-off is constrained customisability: the generated application is

bounded by the current template library. The modular architecture is designed to mitigate this by allowing templates to be extended without modifying the core pipeline.

The multi-user platform layer—authentication, role-based dashboards, and persistent application management—distinguishes IdeaForge from single-shot code generators. Users can return to the platform, launch previously generated applications, and iteratively refine their specifications. This lifecycle-oriented design is essential for practical adoption in team and classroom environments.

11. CONCLUSION

This paper presented IdeaForge, an intelligent platform that automatically converts natural language application descriptions into immediately executable full-stack web applications. The platform integrates a five-stage Application Engine with a two-tier hybrid schema generation mechanism, JWT-based multi-user authentication, role-based dashboards, and a persistent application lifecycle management system.

A controlled preliminary evaluation across 35 multi-domain prompts demonstrated field-level precision above 84%, 100% generation success across both tiers, sub-ten-second end-to-end latency, and usability scores above 4.2/5 across technical and non-technical user groups. All 15 usability study participants successfully generated a working prototype on their first attempt.

IdeaForge is intended as a rapid prototype development tool rather than a replacement for professional software engineering. Its principal contributions—the reliability-focused hybrid schema generation mechanism, the end-to-end five-stage pipeline, and the multi-user platform architecture—represent concrete advances toward making early-stage software prototyping accessible to users without programming expertise. This work was carried out as part of a B.Tech. final year project at M S Ramaiah University of Applied Sciences, Bengaluru.

12. FUTURE WORK

- Multi-entity schema support: generating multiple related tables with foreign-key relationships from a single compound prompt.
- OAuth 2.0 / JWT authentication scaffolding embedded directly into generated applications, enabling secure multi-user access.
- One-click cloud deployment via Docker containerisation and CI/CD pipeline generation targeting AWS, GCP, or Azure.
- Extended field-type vocabulary: enumeration columns, file-upload references, geographic coordinates, and rich-text fields.
- Domain-adaptive fine-tuning of a compact, locally deployable LLM to reduce API dependency, inference latency, and cost.
- Expanded evaluation: 100+ prompt corpus across a wider domain distribution with a formally recruited and counterbalanced usability cohort.
- Production framework export: code translation targeting Django REST Framework, Spring Boot, or Next.js for teams requiring production-grade output.
- Real-time collaborative workspace: shared project environments with concurrent editing, version history, and conflict resolution.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the guidance and mentorship of Dr. Venkata Giri J, Department of Computer Science and Engineering, M S Ramaiah University of Applied Sciences, Bengaluru, for his valuable support, insights, and continuous encouragement throughout the development of this project. This work was undertaken as part of the B.Tech. Computer Science and Engineering Final Year Project (2024–25) at M S Ramaiah University of Applied Sciences, Bengaluru, India.

REFERENCES

- [1] F. Feng et al., "CodeBERT: A Pre-Trained Model for Programming and Natural Language," Proc. EMNLP, pp. 1536–1547, 2020.
- [2] T. Brown et al., "Language Models are Few-Shot Learners," NeurIPS, vol. 33, pp. 1877–1901, 2020.
- [3] M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv:2107.03374, 2021.
- [4] A. Yetistiren, I. Ozsoy, and E. Tuzun, "Evaluating the Code Generation Capability of GitHub Copilot," ACM SIGSOFT, pp. 1–5, 2022.
- [5] S. Nijkamp et al., "CodeGen: An Open Large Language Model for Code Generation," arXiv:2203.13474, 2022.
- [6] Z. Li et al., "Competition-Level Code Generation with AlphaCode," Science, vol. 378, no. 6624, pp. 1092–1097, 2023.
- [7] Y. Wang et al., "A Survey on Large Language Models for Software Engineering," IEEE Trans. Softw. Eng., 2023.
- [8] H. Wong and J. Guo, "AI-Assisted Programming using Large Language Models," IEEE Software, vol. 41, no. 1, pp. 45–53, 2024.
- [9] J. Austin et al., "Program Synthesis with Large Language Models," ACM Comput. Surv., vol. 57, no. 2, pp. 1–36, 2025.
- [10] K. Zhu et al., "Generative AI for Automated Software Development," IEEE Access, vol. 13, pp. 45678–45690, 2025.

BIOGRAPHIES



Sreeya Dora is a final-year B.Tech. student (AIML branch) in the Department of Computer Science and Engineering at M S Ramaiah University of Applied Sciences, Bengaluru. Her research interests include natural language processing, AI-driven development tools, and full-stack engineering. She contributed to system design, AI integration, and frontend development of IdeaForge.

Madhura Bedekar is a final-year B.Tech. student (AIML branch) in the Department of Computer Science and Engineering at M S Ramaiah University of Applied Sciences, Bengaluru. She served as the primary backend developer of IdeaForge, responsible for designing and implementing the FastAPI server, SQLite database architecture, CRUD routing logic, Claude API integration, and the rule-based fallback mechanism.

Vivia Maria Thomas is a final-year B.Tech. student (CSE branch) in the Department of Computer Science and Engineering at M S Ramaiah University of Applied Sciences, Bengaluru. Her contributions include frontend development, template mapping design, and usability testing of the IdeaForge platform.

Chetan Navhi is a final-year B.Tech. student (CSE branch) in the Department of Computer Science and Engineering at M S Ramaiah University of Applied Sciences, Bengaluru. His contributions include evaluation design, baseline comparison methodology, error analysis, and project documentation for IdeaForge.