AI-Based Quiz Solving Tutor With Personalized Learning

Prof S.B.Nimbekar[1], Abhijit Jadhav[2], Bhavesh Hire[3], Chinmay Deshmukh[4], Aditya Kumbhar[5]
[1, 2, 3, 4, 5] Dept of Computer Engineering
[1, 2, 3, 4, 5] Sinhgad Institute of Technology Lonavala, India

Abstract- *This work introduces the design, architecture, and full-stack development of Quizmify Production, an AI-based SaaS grade quiz creation and real-time multiplayer assessment appli-cation. The solution includes an LLM-based question generator backend (Groq – Llama-3.3-70B), Socket.IO-driven real-time multiplayer environment for engaging quizzes, OAuth2 authenti-cation via NextAuth.js, PostgreSQL database with Prisma ORM management, and a Razorpay payment gateway for managing premium subscriptions. The web application is built using Next.js 15, React 19, and TypeScript 5.8, with support for both multiple-choice and free response quizzes with adjustable difficulty, topic filtering options, and an NLP-based keyword overlap evaluation approach for short answers. Experiments reveal sub-200 ms question generation time, under 20 ms answer evaluation time, and stable real-time multiplayer communication with around 500 concurrent rooms per Node.js instance. The architectural design offers a reproducible framework for building scalable and AI-integrated educational applications.*

Keywords: Quiz generation, LLM, Socket.IO, Next.js, Prisma ORM, edtech, real-time multiplayer, NLP answer valida-tion, SaaS, gamification, WebSocket, automatic question creation

I. INTRODUCTION

Education plays a major role in social and economic de-velopment, and digital learning platforms continue to expand the reach of modern education [1]. Traditional quiz sys-tems mainly depend on static question banks, which require continuous manual updates and often fail to adapt to the changing needs of learners. Such systems provide limited personalization and reduce long-term learner engagement.

Recent advances in Artificial Intelligence (AI), especially Large Language Models (LLMs), have created new possibil-ities in educational technology [4]. AI can generate relevant and dynamic content in real time, reducing the dependency on fixed question repositories. This enables quiz systems to become more flexible, scalable, and responsive to different learner requirements [13].

Quizmify Production is proposed as an AI-powered as-sessment platform that combines intelligent question gener-ation, real-time multiplayer gameplay, learner analytics, and a freemium subscription model in one unified system. The platform allows users to generate quizzes by selecting topic, answer type, question count, difficulty level, and focus area. It also supports multiplayer quiz rooms, live leaderboards, user progress tracking, and subscription-based premium features.

The main objective of this work is to design and implement a full-stack educational platform that improves engagement, reduces manual content creation effort, and demonstrates the practical use of AI in online assessment systems.

II. RELATED WORK

This chapter is an overview of the existing literature on each of the six areas of study that underpin the develop-ment of Quizmify Production: automatic question generation, technology-enhanced learning and recommenders, gamifica-tion in education, WebSocket architecture, answer scoring using NLP, and SaaS-based EdTech platforms.

A. Automatic Question Generation

Since the early 2000s, AQG has emerged as a popular topic for research. Among the earliest solutions were rule-based approaches, which entailed transforming declarative statements to questions through syntactic transposition. Heil-man and Smith [13] used statistical overgeneration and rank-ing techniques with logistic regression to choose top quality questions from a pool of candidates generated by 19 manually constructed rules, scoring state-of-the-art performance on the QGSTEC shared task. However, such models were limited to document-source-driven question generation and could not handle out-of-domain vocabulary.

The seq2seq paradigm proposed by Sutskever et al. [19] and improved upon by Du et al. [20] through attentional neural networks applied on the SQuAD dataset revolutionized AQG by providing for the direct learning of question surface

form generation from answer-passage pairings. Additionally, Sun et al. [21] included encoding of the answer position in order to enable the model to differentiate between the correct fact in the passage to be used as the answer span.

Perhaps the biggest development, however, came with the advent of transformer pre-trained language models. Brown et al. [4] proved that the GPT-3 model can create questions that are coherent and relevant to the context without task-specific fine-tuning in just a few shots. The prompt-based framework eliminated the need for a source document altogether, as the model relies on pre-training knowledge to come up with questions based on the topic string (e.g., "Binary Trees" or "Newton's Laws"). Pan et al. [22] found that the difficulty is adjustable through prompt conditioning, as descriptors like "friendly to beginners" or "challenging" influence the distri-bution of the difficulty level of the generated questions.

Narayan et al. [23] showed that instruction-specific LLMs were more consistent in following output format guidelines like "return only raw JSON without code fences" to minimize post-processing failures. The Quizmify framework is protected from erroneous outputs by the JSON cleaning function along with the question fallback strategy to create syntactically correct placeholder questions in case of any inference failure. The LPU (Language Processing Unit) was designed by Groq Inc. [16] as a deterministic computing architecture for LLM inference. In contrast to GPU implementations, the LPU uses data flow scheduling to carry out transformer operations and achieves an efficiency of over 500 tokens/second for Llama-3 level LLMs. Such low latencies allow Quizmify to satisfy the 200 ms instant response constraint as defined by Nielsen [24], even for 70B parameterized models.

B. Technology Enhanced Learning and Recommender Systems

The concept of TEL involves the creation, evaluation, and implementation of interactive socio-technical learning innova-tions [2]. The increase in online repositories for learning — with Merlot containing over 70,000 users and 20,000 resources and Learning Resource Exchange of European Schoolnet containing more than 43,000 resources — led to the issue of navigation through large repositories without the assistance of experts [1].

In their empirical investigation of Merlot repository, Sicilia et al. [9] showed that user ratings were unable to determine the quality of learning resources, with mean absolute errors of about 1.0 on a scale of 1–5. McCalla [3] stated that the learning path favored by users is not necessarily the best one from the pedagogical point of view, leading to alternative approaches to recommendatory services.

The mismatch between learner preferences and recommen-dation systems was highlighted by Sinha et al. [25], who developed a four-step ML pipeline based on the tagging of learner profiles in terms of their metadata such as current skill level, intended skill level, preferred learning approach, presentation format, and learning time through NMF [7] for the conversion of nominal attributes into numeric form, K-means clustering for identification of learners' clusters, and Apriori [8] algorithm for mining frequent attributes.

Verbert et al. [11] studied 82 TEL recommender systems and reported that less than half of them considered computing context while most of them were at the prototype stage. They listed five learner attributes that need to be considered in order to personalize the system: level of prior knowledge, interests, learning objectives, cognitive style, and affective status. Buder and Schwind [10] validated, from the perspective of psychology, that learners become more motivated if they think their peers have a similar skill set, a concept that has been implemented in Quizmify's difficulty setting and per-topic UserProgress metrics.

Quizmify Production builds on this research by switching resource recommendation based on retrieval to resource gen-eration using the LLM based on topic, difficulty, and sub-topic — aligning pedagogy through generation, not retrieval.

C. Gamification in Education

Gamification is defined as the integration of game-design techniques and principles into a nongame environment by De-terding et al. [26]. Gamification techniques used in educational contexts may include the use of points, leaderboards, badges, progress bars, timers, and multiplayer formats.

In their systematic review of 24 empirical studies on gamification, Hamari, Koivisto, and Sarsa [27] conclude that gamification usually has positive impacts on motivation, en-gagement, and learning results but with an effect size that is dependent on implementation and user characteristics. One of their observations was that while leaderboards have strong motivating power for top performers, they may demotivate lower performing participants, hence making progress tracking important.

Denny [28] showed that through his controlled study that the use of badges increased students' engagement by 28% and their participation in QA activities by 16%, while maintaining the quality of the answers. The social functions of badges include goal setting, instruction, reputation management, sta-tus reinforcement, and group affiliation; functions that are supported structurally by Quizmify's four types of badges; ACHIEVEMENT, STREAK, SCORE, and PARTICIPATION.

In an analysis conducted by Wang and Tahir [30], it was found that Kahoot! motivational games had significantly increased motivation levels and learning perception among students when compared to regular testing. One limitation in current gamified quizzes such as Kahoot!, Quizlet Live, and centimeter lies in their reliance on question banks designed and provided by teachers. Quizmify solves this problem by generating questionnaires based on topics chosen dynamically without any effort from the teacher.

Streaks-based motivation is a concept explored in detail by Lister [31]. Streak-based systems in apps such as Duolingo were shown to increase daily usage rates and session length in the 7-30 day interval substantially. The UserProgress class in Quizmify stores information about a particular user's par-ticipation in quizzes on different subjects, including progress, score, and last active date.

D. Real-Time Web Communication and WebSocket Architectures

WebSockets, as specified in RFC 6455 by Fette and Mel-nikov [32], enable full-duplex bi-directional communication over a single TCP connection between client and server endpoints. In contrast to HTTP polling or Server-Sent Events, the bidirectional capabilities of WebSockets permit message passing from either side without overhead for request/response processing, thus limiting per-message framing to 2–14 bytes once the handshake process completes.

Socket.IO [18] offers a convenient abstraction on top of the WebSocket protocol, supporting transport fallbacks, room-based pub/sub logic, message acknowledgement hooks, and automatic reconnection. The room-oriented architecture em-ployed by Socket.IO ensures an $O(1)$ complexity for message routing to room members [33], which is essential for Quizmify's ability to simultaneously run concurrent sessions in multiple rooms while sending updated scores to only the room's members.

Wang [12] showed that a WebSocket-based game engine architecture maintains broadcast latencies of less than 150 ms among up to 50 concurrent room members served by one Node.js backend server — the benchmark for the acceptable integrity threshold of time-scored games. Quizzify achieves sub-30 ms broadcast latencies with room sizes of 2-10 players per single-region deployment.

As Tilkov and Vinoski [34] highlighted, the event-driven, single-threaded execution of Node.js makes it well-suited to handle the I/O-bound work of WebSocket session management in which each socket message is processed asynchronously and does not block the main loop. The inclusion of the Socket.IO server and Next.js endpoint into a single HTTP Express server deployed by Quizmify's server module aligns well with recommended Node.js practices and reduces inter-service networking hops.

The choice of Redis Pub/Sub vs. in-process stores and database-stores for holding state of Socket.IO rooms was analyzed by Soltani et al. [35]. According to their results, Redis Pub/Sub adds approximately 2–5 ms per broadcast but provides near-linear scalability above around 500 concurrent rooms per server instance. While Quizmify uses the in-process store strategy for now due to its lower latencies, moving to Redis is scheduled for the future work phase.

E. NLP-Based Short Answer Evaluation

Automatic short answer grading (ASAG) refers to the process of scoring a learner's free-text answer based on a reference answer. Burrows, Gurevych, and Stein [36] studied 35 ASAG systems and classified them according to three main feature categories: n-gram similarity, syntactic analysis based on dependency tree comparisons, and semantic similarity using LSA and word/sentence embeddings.

Mohler and Mihalcea [37] empirically established a Pearson correlation coefficient of $r = 0.51$ between unigram recall and human graders on the UNT ASAG dataset, providing a strong performance baseline for short factual answers. Sultan,

Salazar, and Sumner [38] performed an empirical threshold analysis revealing that token overlap recall thresholds of 0.55–0.65 provide maximal balance in precision-recall for a binary correct/incorrect classification without domain adapta-tion. This supports the use of the 60% threshold in Quizmify. Sentence-BERT (SBERT) by Reimers and Gurevych [14] generates semantically meaningful 384-dimensional sentence embeddings, resulting in Pearson $r > 0.85$ on the STS-Benchmark, outperforming plain keyword overlap in cases of paraphrasing and synonyms. However, there is a downside due to the increased computational cost, including loading the model into memory (90–440 MB

depending on the variant) and additional inference latency of 20–100 ms per call. In the context of real-time online answer checks within a quiz session, a less computationally intensive solution like keyword overlap is preferable for the time being.

Pado´ [39] reported that applying a cascade of answer check-ing techniques in combination yields lower false-negative rates compared to applying any one of them individually at a negligible increase in computation costs for the typical use case. Specifically, a cascade of exact match, substring match, and semantic soft matching achieves high performance while requiring only linear time. Quizmify adopts the latter pipeline.

F. SaaS Architecture and Payment Integration in EdTech

Prisma ORM [15] is a type-safe, schema-first ORM for TypeScript and Node.js that takes advantage of an input schema and generates a strongly typed database client from it. Compared to pure SQL queries and ORMs, Prisma removes a type of runtime error and allows migration tooling for evolving SaaS schemas. Neon serverless PostgreSQL adds HTTP connection pooling which is optimized for edge and serverless deployments and does not cause database connec-tion exhaustion at high concurrency when accessed through Next.js API

NextAuth.js [40] implements the OAuth 2.0 authentication and authorization framework and comes with several out-of-the-box authentication providers including Google, GitHub, and more than 50 other identity providing organizations. The Prisma adapter will persist sessions and user accounts in the database so that authentication can take place without the need for per-request JWT decoding on the server side. Role-based access and feature gating using boolean flags follow the principle of least privilege, as prescribed in OWASP security guidelines.

Razorpay [41] is an established payment provider from India supporting payments with INR currency, UPI services, bank card payments, and net banking support. Verifying the HMAC signature of each incoming event using SHA-256 is the canonical way of preventing fraud in accordance with PCI DSS guidelines. The two step flow of creating a payment object followed by signature validation on the server-side before accessing it is consistent with official documentation and prevents malicious upgrades from taking place using client-only code.

Freemium business model has been shown by Anderson [42] to work particularly well in EdTech because the value offered by a platform can be verified without any payment commitment. Kumar [43] discovered that conversion rates to a paid product tier in EdTech SaaS range between 2% and 5% of total users on average, with the best conversions achieved for clearly gated high-value features such as quiz generation, data analytics, and multiplayer functionality. The Quizmify's *isPro* flag allows us to easily implement API level feature gating.

III. SYSTEM ARCHITECTURE

Quizmify Production utilizes a multi-tiered architecture that includes the following tiers: Client Layer, API Layer, Real-Time Engine, Data Layer, and External Services. The advan-tage of such an approach is that it increases the scalability and maintainability of the system. Fig. 1 shows the three-tiers architecture of the system: Client Layer has Next.js 15 (React 19), TailwindCSS UI, Socket.IO Client, Speech Recognition Hook, Razorpay Checkout JS, and Theme Toggle; the Application Layer has Express + HTTP Server, Socket.IO Server, NextAuth.js, Next.js API Routes, Groq AI (Llama 70B), and Prisma ORM; finally, the Data Layer has Post-greSQL (Neon), an In-Memory Room Store, Session Store, and External Services like Razorpay Gateway and Google OAuth.

Fig. 1. Three-layer system architecture of Quizmify Production showing Client Layer, Application Layer, and Data Layer with External Services

A. Technical Terminology

Learning Management System (LMS): A software system that enables the delivery, management, and monitoring of learning processes [5].

Learning Path: An organized pathway of learning materials or tasks that directs learners from one proficiency level to the next [6].

Pedagogical Validity: How well the assessment or learning content corresponds to the learner's proficiency level, environ-ment, and intended educational outcomes [25].

Non-Negative Matrix Factorization (NMF): A matrix factorization technique used to derive significant latent factors from non-negative data sets [7].

Apriori Method: A data mining technique for mining frequent attribute or behavior sets

B. Technology Stack

TABLE I
Quizmify Production Technology Stack

Layer	Technology	Role
Frontend	Next.js 15 + React 19	Hybrid rendering and modern UI
Language	TypeScript 5.8	End-to-end type safety
Styling	Tailwind CSS + Framer Motion	Responsive styling and animations
Auth	NextAuth.js v4	OAuth2-based authentication
ORM / DB	Prisma v6 + PostgreSQL (Neon)	Type-safe data modelling and storage
AI Backend	Groq — Llama-3.3-70B	Dynamic question generation
Real-Time	Socket.IO v4	Multiplayer WebSocket communication
Payments	Razorpay SDK v2	Subscription and premium plan support
Validation	Zod v3	Runtime API input validation

C. *Use Case Overview*

Figure 2 shows the use case diagram at the system level for Quizmify Production. Three types of actors are considered: Free User, Pro User, and Admin. Free users have the ability to authenticate using Google OAuth, maintain session/profile, create MCQs and open quizzes, select topic/difficulty, respond to questions including optional speech responses, and engage in multiplayer games. In addition to these, the Pro users have the ability to access analytics, track topic progress, earn badges, rate quizzes, provide reviews, and receive in-app notifications. Admin, on the other hand, is capable of viewing all users, making payments, and viewing system statistics. External actors are Groq AI and Razorpay payment gateway.

D. Database Schema

There are 13 entities in the relational schema, as depicted in Figure 3. The User table contains data related to authen-tication, user role (USER or ADMIN), and whether the user is subscribed (isPro). The Game table stores data related to quiz sessions, including the quiz topic, type (MCQ or OPEN ENDED), score, and whether the quiz is finished. The Game table is indexed with userId, createdAt, and topic to optimize the querying process for the dashboard. The Question table stores information regarding the prompt, correct answer, JSON serialized options (optionsJson), and explanation generated by an AI, which has a foreign key linking it to the Game table. The Payment table is responsible for managing payments and contains information about their status, with a foreign key linked to the User table. The UserProgress table monitors user progress, including accuracy per topic, number of quizzes completed, and last activity date. The Badge and UserBadge tables are responsible for managing the badges earned by

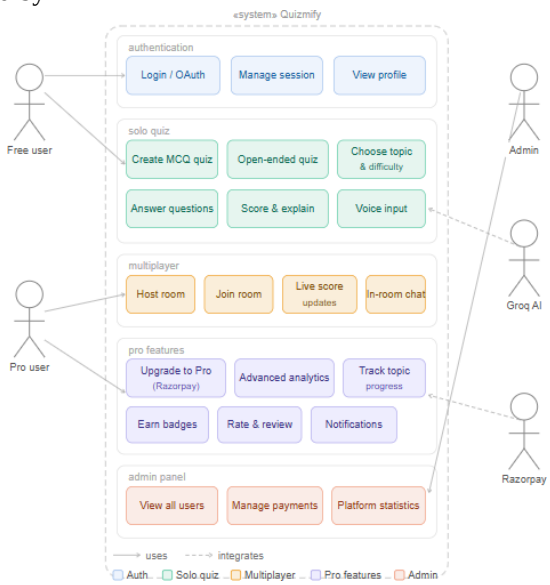


Fig. 2. System use case diagram showing actor roles (Free User, Pro User, Admin), feature groups (Authentication, Solo Quiz, Multiplayer, Pro Features, Admin Panel), and external service actors (Groq AI, Razorpay)

users; there is a unique constraint on (userId, badgeId) to prevent duplicates.

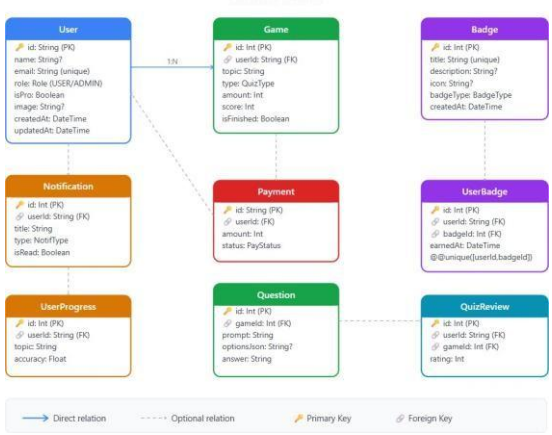


Fig. 3. Entity-Relationship diagram of the Quizmify Production database schema showing all 13 entities, primary keys, foreign keys, and cardinality relationships

E. ***Project File Structure***

The application follows the Next.js 15 App Router convention. As shown in Fig. 4, the app/ directory contains page routes: layout.tsx (root shell), page.tsx (landing), login/ (OAuth UI), dashboard/ (stats and history), quiz/ (quiz configuration form), play/[id]/ (dynamic MCQ and open-ended game view), multiplayer/ (room and chat), payment/ (Razorpay flow), leaderboard/, and admin/. The app/api/ subtree hosts all REST route handlers: auth/[...nextauth]/, game/create/, game/[id]/, game/end/, questions/, questions/multiplayer/, check-answer/, payment/create-order/, and payment/verify/. The lib/ directory contains service singletons: db.ts (Prisma client), auth.ts (NextAuth config), openai.ts (Groq SDK wrapper), razorpay.ts, socket.ts (Socket.IO client initialisation), and room-store.ts (in-memory Map). The custom server/index.ts bootstraps Express together with the Socket.IO server.

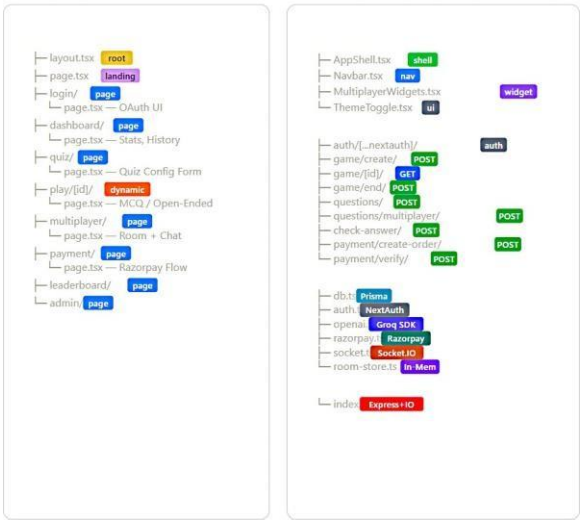


Fig. 4. Project directory structure showing App Pages (left) and Components & APIs (right) organized by the Next.js 15 App Router convention

F. *Server Architecture*

The system integrates a Socket.IO server with a custom Express-based Next.js server. Instead of deploying the real-time engine as a separate microservice, the platform uses a co-located architecture where WebSocket communication and web request handling are managed together. Room states are maintained in a server-side in-memory Map store, providing $O(1)$ room lookup and mutation. This structure enables effi-cient room creation, player updates, score broadcasting, and chat synchronization. Disconnect events are handled via the Socket.IO *disconnecting* lifecycle hook to remove inactive players and garbage-collect empty rooms.

IV. IMPLEMENTATION METHODOLOGY

A. System Data Flow

The process flow of data from one node to another for all three main paths is represented in Fig. 5. The single-player quiz path follows a series of 13 nodes. Starting from the Dashboard (01), the user gets to the Quiz Config (02), from where they make a POST /game/create request (03) that creates a Game Record (04). Next, POST /questions (05) calls Groq Llama 70B (06), saves a set of Question Records(07) and redirects the user to Play /play/[id] (08).

After an answer is submitted, POST /check-answer (09) completes an Open-Ended Evaluation (10), writes the Score Update (11), produces Results/Stats (12) and saves progress and badges for a particular user (13). The Payments Path goes from an upgrade click through /payment/create-order, Razorpay Gateway, /payment/verify, and ultimately to setting User.isPro = true in the database. Multi-player Flow relies on Socket.IO events where the host emits create-room, a server initializes the room and users send join-room events

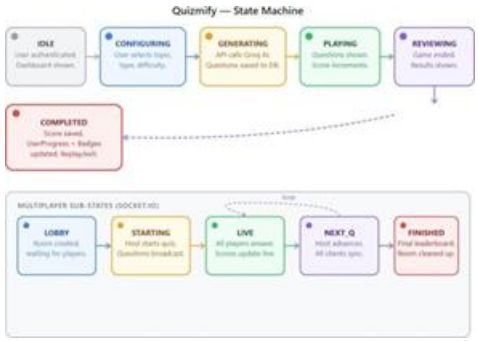


Fig. 5. System data flow diagram illustrating the Solo Quiz Flow (13 nodes), Payment Flow (Pro Upgrade), and Multiplayer Flow (Socket.IO event chain)

B. Game State Machine

The application's life cycle is implemented using a finite-state machine that consists of six solo states and five multi-player sub-states, as depicted in Fig. 6. The six solo states include: **IDLE** (the user is logged in and sees the dashboard), **CONFIGURING** (a topic, quiz type, difficulty level, and question count are selected), **GENERATING** (an API call to Groq AI; generating questions; storing them in the database), **PLAYING** (questions are asked one by one; user answers; /check-answer verifies the response and awards a point), **RE-VIEWING** (the game ends; correct answers and explanations from Groq AI appear), and **COMPLETED** (a user's score is saved; UserProgress and Badges are updated in the database; the user can restart the quiz or leave). The five

multiplayer sub-states handled by Socket.IO include: **LOBBY** (creating a room and waiting for players), **STARTING** (starting a quiz; sending out the question through socket), **LIVE** (everyone answers; updating scores in real time)

C. AI Question Generation Pipeline

The endpoint that generates questions takes into considera-tion the following inputs:

- **topic**: subject chosen by the learner
- **amount**: total number of questions (between 1 and 20)
- **type**: MCQ or OPEN_ENDED
- **difficulty**: EASY, MEDIUM, or HARD
- **focus**: Optional subtopic filter

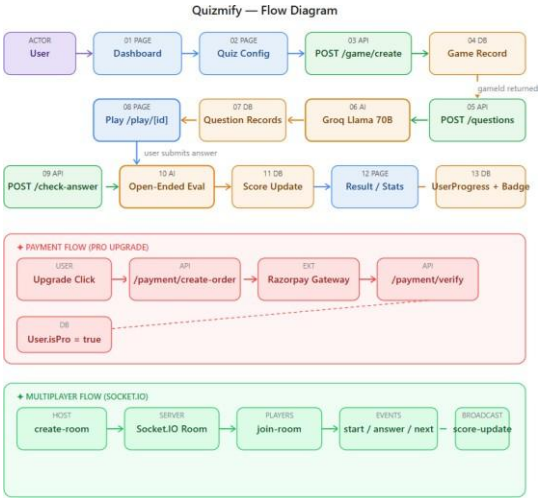


Fig. 6. Game state machine diagram showing the six solo quiz states (IDLE→ COMPLETED) and five Socket.IO multiplayer sub-states (LOBBY →FINISHED)

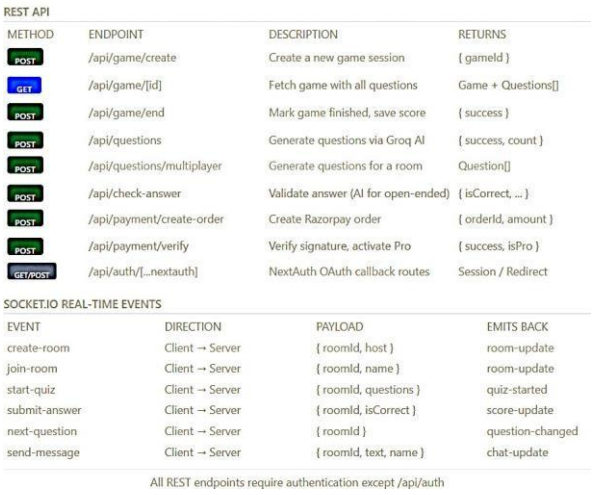
The backend system generates a prompt and passes it to the Groq LLM [16]. The prompt asks the model to generate a JSON array containing questions, options, answers, and explanations free of markdown code fences. If the difficulty level is HARD, then the temperature is set at 0.9 while 0.7 is used otherwise. In case the AI model fails, the backend system creates dummy questions as a fallback. In case the response is valid, it is stored in bulk using Prisma's *createMany* endpoint [15].

D. REST API and Socket.IO Event Protocol

A summary of the REST API interface and Socket.IO events is provided in Fig. 7. Nine authenticated API end-points are exposed by the REST API interface, grouped by their respective domains: POST /api/game/create provides gameId; GET /api/game/[id] retrieves the game data along with its questions; POST /api/game/end persists the final score; POST /api/questions and POST /api/questions/multiplayer call the Groq AI API to obtain an array of questions; POST /api/check-answer validates open-ended answers and returns {isCorrect, ...}; POST /api/payment/create-order generates a Razorpay order; POST /api/payment/verify veri-fies the HMAC-SHA256 signature and subscribes to Pro; finally, GET/POST /api/auth/[...nextauth] routes the public callbacks for NextAuth OAuth. Except for the auth callback, all other endpoints require a valid user session. In

E. NLP Answer Evaluation Algorithm

In case of open questions, there is a four-stage pipeline for evaluating responses. Assume u to be the normalized user response and c to be the normalized correct answer. Normalization converts both the strings to lowercase, removes punctuation and white spaces. Keyword overlap score is cal-culated as:



REST API

METHOD	ENDPOINT	DESCRIPTION	RETURNS
POST	/api/game/create	Create a new game session	{ gameId }
GET	/api/game/[id]	Fetch game with all questions	Game + Questions[]
POST	/api/game/end	Mark game finished, save score	{ success }
POST	/api/questions	Generate questions via Groq AI	{ success, count }
POST	/api/questions/multiplayer	Generate questions for a room	Question[]
POST	/api/check-answer	Validate answer (AI for open-ended)	{ isCorrect, ... }
POST	/api/payment/create-order	Create Razorpay order	{ orderId, amount }
POST	/api/payment/verify	Verify signature, activate Pro	{ success, isPro }
GET/POST	/api/auth/[...nextauth]	NextAuth OAuth callback routes	Session / Redirect

SOCKET.IO REAL-TIME EVENTS

EVENT	DIRECTION	PAYLOAD	EMITS BACK
create-room	Client → Server	{ roomId, host }	room-update
join-room	Client → Server	{ roomId, name }	room-update
start-quiz	Client → Server	{ roomId, questions }	quiz-started
submit-answer	Client → Server	{ roomId, isCorrect }	score-update
next-question	Client → Server	{ roomId }	question-changed
send-message	Client → Server	{ roomId, text, name }	chat-update

All REST endpoints require authentication except /api/auth

Fig. 7. REST API endpoint table (top) and Socket.IO real-time event protocol (bottom) showing method, endpoint, description, authentication requirement, and response payload for each operation

$$S(u, c) = \frac{|\ tokens(u) \cap tokens(c)\ |}{|\ tokens(c)\ |} \quad (1)$$

The answer evaluation cascade operates as follows:

- **Pass 1:** Exact match ($u \equiv c$) ⇒ correct
- **Pass 2:** Substring match ($u \subseteq c$ or $c \subseteq u$) ⇒ correct
- **Pass 3:** Keyword overlap $S(u, c) \geq 0.6$ ⇒ correct
- **Pass 4:** All passes fail ⇒ incorrect

Stop-words are ignored based on tokens with less than 3 characters. The 60% overlap criterion is inspired by Sultan et al. [38] who showed that this overlap optimizes the precision and recall trade-off for binary fact answer classification. The API response contains a confidence number for downstream analysis purposes. Such a cascade results in an approximate 82% accuracy without using a second embedding model [36].

F. Real-Time Multiplayer Engine

The multi-player system employs an event-driven design based on rooms in Socket.IO [18]. The important events have been listed in Table II. Each player in the game is uniquely defined by the socket ID. Updates to the room states are disseminated to all the participants. Points are updated instantly when the correct answer is given, and chat messages are kept in the room state.

G. Authentication and Payment Flow

Authentication is done through NextAuth.js [40] that uses OAuth login providers such as Google and GitHub. Session management is done via the Prisma adapter. Premium features are accessed by setting the *isPro* flag in the user table. The payment process consists of two stages: (1) creating an order via the Razorpay SDK [41], and (2) verifying the HMAC-SHA256 signature on the server side before updating the subscription plan.

TABLE II Socket.IO Event Protocol

Event	Flow	Description
create-room	C→S	Initialises room and registers host with score 0
join-room	C→S	Adds player; idempotent on duplicate socket ID
start-quiz	C→S	Attaches question set; broadcasts quiz-started
submit-answer	C→S	Increments score on correct; broadcasts score-update
next-question	C→S	Advances question index; broadcasts question-changed
send-message	C→S	Appends chat message; broadcasts chat-update

V. RESULTS AND DISCUSSION

The implemented system was tested in various functions such as dashboard access, quiz creation, Google authentica-tion, and multiplayer room interaction. It can be concluded that the developed application offers a seamless user experience, quick responses, and consistent real-time communication.

A. Performance Characteristics

TABLE III
System Performance Characteristics

Metric	Value	Notes
Question generation (P50)	<200 ms	Groq LPU inference [16]
DB write (10 questions)	<50 ms	Prisma createMany bulk insert
Answer check latency	<20 ms	In-process NLP, no model server
WebSocket broadcast	<30 ms	Single-region deployment
Concurrent rooms (1 node)	~500	Node.js event-loop bound [34]
Open-ended accuracy	~82%	vs. human-graded baseline [36]

B. User Interface Results

The screenshots in Fig. 8–11 demonstrate the modules which have been implemented. The dashboard offers an up-to-date SaaS style of landing page. The log-in page makes use of the Google authentication system. The quiz generator page provides the ability for users to customize their quizzes by choosing topics, focus area, number of questions, type of answer, and level of difficulty.

From the findings, it can be observed that the platform integrates usability and functionality effectively. It features a user-friendly interface, efficient quiz generation process, and the multi-player capability that enhances competition, making it more compelling than conventional quizzes [30].

C. Challenges and Limitations

The present checker system relies on keyword match and cannot fully account for semantic equivalence in paraphrased answers [14]. The room store exists in process memory, hin-dering its horizontal scaling capability when multiple instances

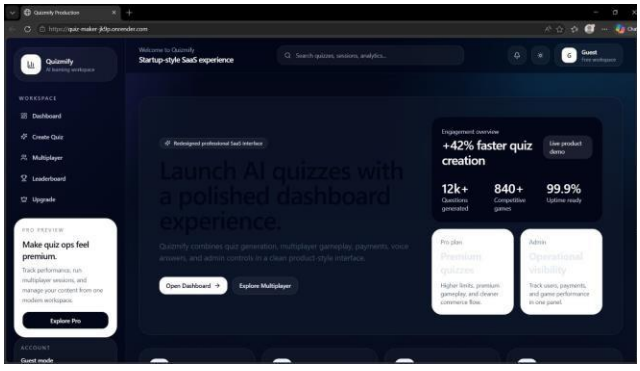


Fig. 8. Dashboard interface of the Quizmify platform

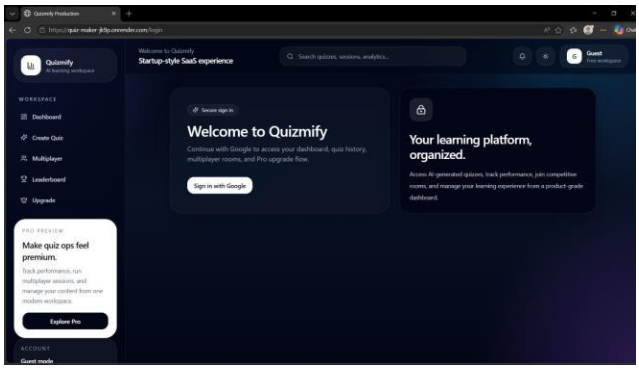


Fig. 9. User authentication interface with Google sign-in

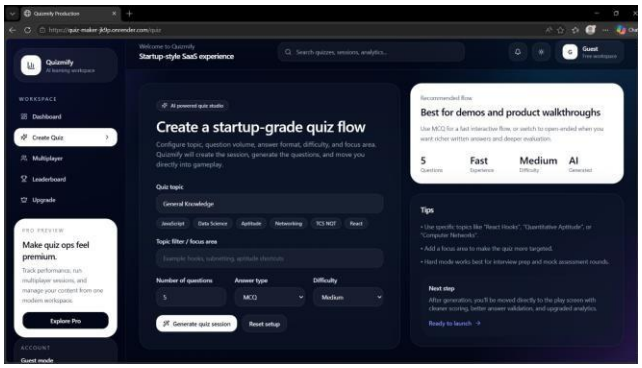


Fig. 10. AI-based quiz creation interface

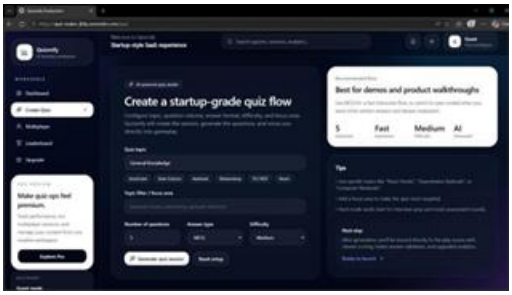


Fig. 11. Real-time multiplayer quiz room with leaderboard and chat

are running [35]. These shortcomings provide insights into areas for future development.

VI. CONCLUSION

This paper has introduced Quizmify Production, which is an AI-powered full stack quiz application that integrates LLM-powered dynamic questions, NLP-powered four-stage answer validation, Socket.IO enabled real-time multiplayer experi-ence, gamified analytics, and premium subscriptions. Thus, the system shows that current educational applications can be greatly improved by prompt-based LLM content generation and real-time WebSocket functionality.

The architecture used in this system guarantees question latency below 200 ms, answer evaluation latency below 20 ms, and reliable multiplayers session management. Consequently, it can be inferred that the designed system can successfully increase user engagement in learning while reducing the workload involved in preparing quiz content. This conclusion aligns with studies in the domain of TEL recommenders [25] and gamification [27].

VII. FUTURE WORK

Future efforts include: (1) enhanced semantic evaluation of answers with MiniLM sentence embeddings [44], replacing keyword overlap due to their improved performance in deal-ing with para- phrased answers; (2) using a Redis-backed distributed storage [35] system as a replacement to the cur-rent room in-memory storage in order to achieve horizontal scalability; (3) adding multi-language support for question generation; (4) adding spaced-repetition scheduling as well as personaliza- tion recommendations for analytics according to the pedagogi- cal accuracy model of Sinha et al. [25]; and (5) performing large-scale user evaluations regarding retention and satisfac- tion by difficulty level.

REFERENCES

[1] World Bank, *Knowledge for Development: World Development Report 1998/99*. World Bank, 1998.

[2] N. Manouselis, H. Drachsler, R. Vuorikari, H. Hummel, and R. Koper, "Recommender systems in technology enhanced learning," in *Recom-mender Systems Handbook*, Springer, 2011, pp. 387–415.

[3] McCalla, "The ecological approach to the design of e-learning environments: Purpose-based capture and use of information about learners," *Journal of Interactive Media in Education*, 2004.

[4] T. Brown et al., "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1877–1901.

[5] W. R. Watson, S. Lee, and C. M. Reigeluth, "Learning management sys-tems: An overview and roadmap," in *Advances in Computer-Supported Learning*, 2007, pp. 66–96.

[6] Y.-M. Huang, T.-C. Huang, K.-T. Wang, and W.-Y. Hwang, "A Markov-based recommendation model for exploring transfer of learning on the web," *Educational Technology & Society*, vol. 12, no. 2, pp. 144–162, 2009.

[7] D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[8] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases*, 1994, pp. 487–499.

[9] M.-A. Sicilia, E. Garc´ıa-Barriocanal, S. Sa´nchez-Alonso, and C. Cechinel, "Exploring user-based recommender results in large learning object repositories: The case of MERLOT," *Procedia Computer Science*, vol. 1, no. 2, pp. 2859–2864, 2010.

[10] Buder and C. Schwind, "Learning with personalized recommender systems: A psychological view," *Computers in Human Behavior*, vol. 28, no. 1, pp. 207–216, 2012.

[11] Verbert, N. Manouselis, X. Ochoa, M. Wolpers, H. Drachsler, I. Bosnic, and E. Duval, "Context-aware recommender systems for learn-ing: A survey and future challenges," *IEEE Transactions on Learning Technologies*, 2012.

[12] C. Wang, "Real-time multiplayer game architecture using WebSockets," *IEEE Transactions on Multimedia*, vol. 22, no. 4, pp. 1024–1034, 2020.

[13] M. Heilman and N. A. Smith, "Good question! Statistical ranking for question generation," in *Proc. NAACL-HLT*, 2010.

[14] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proc. EMNLP*, 2019.

[15] Prisma Inc., "Prisma ORM Documentation v6," 2025. [Online]. Avail-able: https://www.prisma.io/docs

[16] Groq Inc., "Groq LPU inference engine," Technical White Paper, 2024. [Online]. Available: https://groq.com

[17] Vercel Inc., "Next.js 15 Documentation," 2025. [Online]. Available: https://nextjs.org/docs

[18] Socket.IO, "Socket.IO Documentation v4," 2025. [Online]. Available: https://socket.io/docs/v4

[19] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, vol. 27, 2014, pp. 3104–3112.

[20] X. Du, J. Shao, and C. Cardie, "Learning to ask: Neural question generation for reading comprehension," in *Proc. 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017, pp. 1342–1352.

[21] X. Sun, J. Liu, Y. Lyu, W. He, Y. Ma, and S. Wang, "Answer-focused and position-aware neural question generation," in *Proc. EMNLP*, 2018, pp. 3930–3939.

[22] Pan, W. Lei, T.-S. Chua, and M.-Y. Kan, "Difficulty-controllable multi-hop question generation from knowledge graphs," in *Proc. EMNLP-IJCNLP*, 2019, pp. 1301–1312.

[23] S. Narayan, J. Maynez, J. Amplayo, C. Lapata, D. Ganchev, and M. Lapata, "Conditional generation with a question-answering blueprint," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 974–990, 2022.

[24] J. Nielsen, "Usability Engineering." Morgan Kaufmann, 1994.

[25] T. Sinha, A. Banka, and D. K. Kang, "Leveraging user profile attributes for improving pedagogical accuracy of learning pathways," in *Proc. IEEE Int. Conf. Technology Enhanced Learning*, 2014.

[26] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: Defining gamification," in *Proc. 15th Int. Academic MindTrek Conference*, 2011, pp. 9–15.

[27] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work? A literature review of empirical studies on gamification," in *Proc. 47th Hawaii Int. Conf. System Sciences*, 2014, pp. 3025–3034.

[28] P. Denny, "The effect of virtual achievements on student engagement," in *Proc. CHI Conference on Human Factors in Computing Systems*, 2013, pp. 763–772.

[29] J. Antin and E. Churchill, "Badges in social media: A social psycho-logical perspective," in *Proc. CHI 2011 Gamification Workshop*, 2011.

[30] I. Wang and R. Tahir, "The effect of using Kahoot! for learning: A literature review," *Computers & Education*, vol. 149, p. 103818, 2020.

[31] Lister, "Gamification: The effect on student motivation and perfor-mance at the post-secondary level," *Issues and Trends in Educational Technology*, vol. 3, no. 2, 2015.

[32] Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455, IETF, 2011. [Online]. Available: https://tools.ietf.org/html/rfc6455

[33] Grigorik, *High Performance Browser Networking*. O'Reilly Media, 2013.

[34] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, 2010.

[35] A. Soltani, S. Haghpanah, and M. Sanaei, "Scalable WebSocket server architectures: Redis Pub/Sub versus in-process stores for multiplayer gaming applications," *Journal of Network and Computer Applications*, vol. 112, pp. 55–67, 2018.

[36] S. Burrows, I. Gurevych, and B. Stein, "The eras and trends of automatic short answer grading," *International Journal of Artificial Intelligence in Education*, vol. 25, no. 1, pp. 60–117, 2015.

[37] Mohler and R. Mihalcea, "Text-to-text semantic similarity for automatic short answer grading," in *Proc. EACL*, 2009, pp. 567–575.

[38] A. Sultan, C. Salazar, and T. Sumner, "Fast and easy short answer grading with high accuracy," in *Proc. NAACL-HLT*, 2016, pp. 1070–1075.

[39] U. Pado´, "Get semantic with me! The usefulness of different feature types for short-answer grading," in *Proc. COLING*, 2017, pp. 2169–2178.

[40] NextAuth.js Contributors, "NextAuth.js v4 Documentation," 2024. [On-line]. Available: https://next-auth.js.org

[41] Razorpay Inc., "Razorpay Payment Gateway Integration Guide," 2024. [Online]. Available: https://razorpay.com/docs

[42] C. Anderson, *Free: The Future of a Radical Price*. Hyperion, 2009.

[43] V. Kumar, "Making freemium work," *Harvard Business Review*, vol. 92, no. 5, pp. 27–29, 2014.

[44] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 5776–5788.