



International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

HackMatch: Smart Hackathon Team Formation System

Dhanapathy G, Aarthi S, Pradeepa Shree S, Bendi Hasini

Department of Artificial Intelligence and Data Science, Sri Manakula Vinayagar Engineering College (SMVEC), Madagadipet, Puducherry, India
ghanapathy.aid@smvec.ac.in, aarthiselvamurugamurugan07@gmail.com, pradeepas072@gmail.com, hasinibendi86@gmail.com

ABSTRACT:

Hackathon participants commonly form teams through informal social networks, which can lead to redundant skills, poor role diversity, and misaligned interests and availability. This paper presents HackMatch, a full-stack web system for automated hackathon team formation. Team compatibility is modeled with eight normalized features covering skill coverage, skill complementarity, domain alignment, role diversity, availability compatibility, experience diversity, team-size suitability, and domain coverage. A Gradient Boosting Regressor estimates compatibility scores, and K-Means clustering is used to reduce the candidate search space for larger pools. Formation is integrated with invitations, team setup, team chat, rankings, analytics, email notification, and post-event feedback, and each team receives a rule-based natural-language explanation of its score. A preliminary evaluation with 25 university participants produced an average balance score of 0.73 and a formation time of 1.8 s for a pool of 25 participants and a team size of four. Results are preliminary: the model is trained on synthetic compatibility labels and evaluated on a small cohort without a baseline.

Index Terms: hackathon team formation, machine learning, gradient boosting, K-Means clustering, explainable AI, collaborative systems, full-stack application, explainable AI.

I. INTRODUCTION

Hackathons are short-duration innovation events in which participants from different technical and academic backgrounds collaborate to build prototypes, typically within 24 to 72 hours. They are run by universities, companies, and public bodies to accelerate innovation, identify talent, and build community. Their outcome depends not only on the idea but also on the composition of the team: teams need complementary technical skills, shared problem interests, compatible schedules, and a useful mix of experience.

In many university events, however, teams form through personal contacts, messaging groups, and informal conversation. This practice has several recurring problems. Participants without an existing network are disadvantaged. Teams often duplicate skills (for example, three frontend developers and no backend developer). Domain interests are rarely aligned, availability and experience are ignored, and coordinators spend considerable time assigning people by hand. The problem grows with the size of the cohort.

Team formation is a constrained grouping problem. A keyword-based matcher can find common skills but cannot represent role diversity, domain alignment, availability, or experience balance. Research on team formation is largely graph- or constraint-based and assumes social-network data or hand-specified constraints that are rarely available for a new hackathon cohort. Many research prototypes also stop at the algorithm and do not address the operational workflow of an event: inviting members, handling refusals, letting teams organize themselves, and monitoring progress.

This paper presents HackMatch, which combines engineered compatibility features, machine-learning-based scoring, candidate search, and an integrated event-management platform. The main contributions are: (1) an eight-feature compatibility representation for hackathon teams; (2) a Gradient Boosting scoring component with rule-based, human-readable explanations; (3) a single-team formation workflow with a permanent team lock and invitation-response handling; (4) a candidate-reduction strategy using K-Means clustering for larger pools, together with an analysis of search-space growth; and (5) a deployed full-stack system that includes authentication, privacy-aware discovery, chat, rankings, analytics, notifications, and feedback.

The remainder of the paper is organized as follows. Section II reviews related work. Section III formulates the problem and lists design requirements. Section IV describes the system architecture, and Section V the formation engine. Section VI details the participant workflow and Section VII the application modules. Section VIII reports a preliminary evaluation. Section IX discusses limitations, threats to validity, and ethics, and Section X concludes.

II. RELATED WORK

A. Team Formation in Social Networks

Lappas et al. formulated team formation as selecting a set of experts from a social network who together cover the skills required by a task while minimizing communication cost [1]. Kargar and An extended the formulation to teams with a designated leader and to top-k teams [2]. Anagnostopoulos et al. studied online team formation, in which tasks arrive over time and the workload of experts must be balanced [3], and Li and Shan considered generalized tasks with richer skill requirements [4]. These methods are elegant and come with approximation guarantees, but they assume that a collaboration network exists among candidates. In a hackathon cohort whose members are meeting for the first time, such a network is generally unavailable. Their objective, skill coverage at low communication cost, also does not capture domain interest, availability, role balance, or experience mix, and the resulting teams are hard to explain to participants in non-graph terms.

B. Hackathons and Collaborative Events

Porras et al. reviewed a decade of hackathons in software engineering education and reported both learning benefits and organizational difficulties [5]. Nolte et al. examined the outcomes of a corporate hackathon, including what happened to projects after the event [6]. This literature motivates careful attention to team composition and event logistics, but it is descriptive; it does not provide a deployable formation tool or evaluate automated assignment.

C. Group Composition in Crowd and Online Teams

Lykouroutzou et al. showed that balancing personality types can improve the outcomes of crowd teams [7]. Their result suggests that skill-only formation leaves useful signal unused. HackMatch does not model personality. It uses experience level and preferred role as coarse proxies for working style and lists validated personality instruments as future work.

D. Learned Scoring and Explanation

Learned scoring functions can capture interactions between features that a linear rule misses, and gradient boosting [8] is a strong general-purpose choice for tabular data of this kind. Post-hoc explanation methods such as LIME [9] and SHAP [10] can attribute a model prediction to input features. HackMatch instead takes a simpler route: its inputs are eight human-interpretable features, and explanations are generated by threshold rules on those features. This keeps explanations faithful to the quantities participants can inspect, at the cost of not explaining the model's internal interactions.

E. Positioning

HackMatch differs from the cited work in four respects: it scores teams from profile attributes alone without a social graph; it produces an explanation for each formed team; it treats formation as one step of an operational workflow with invitations, refusals, and participant-controlled team setup; and it is delivered as a running system. We do not claim better team quality than the cited methods, because no head-to-head comparison has yet been performed (Section VIII).

III. PROBLEM FORMULATION

Let $P = \{p_1, \dots, p_N\}$ be the set of registered participants. Each participant p_i has a skill set S_i , domain set D_i , preferred role r_i , availability a_i (in hours per day), experience level e_i , and experience indicators (past hackathons and projects). For a candidate team T of size k , the eight-dimensional feature vector $x(T) = [f_1, \dots, f_8]$ is computed, and a trained regressor g returns a compatibility score in $[0, 1]$. The selected team is

$$T = \arg \max_{T \in F} g(x(T)) \quad (1)$$

with feasible teams F of size k formed from the unassigned participants. The following operational constraints apply: a participant belongs to at most one active team; the members of a locked team are never selected again; a participant who declined an invitation is back in the waiting pool; and a formation request creates exactly one team.

A. Design Requirements

The system was designed to meet the following requirements, which will be revisited in later sections.

R1. Compatibility: teams should combine complementary skills and related domains, have varied roles and mixed experience.

R2. Transparency: every formed team should carry an explanation that participants understand.

R3. Stability: a formed team must not be reshuffled by later formation requests.

R4. Agency: participants can decline and the members choose their team name and leader.

R5. Privacy: profiles are not exposed for casual browsing, and credentials never reach the browser.

R6. Operability: organizers can monitor progress and communicate with participants without direct access to the database.

IV. SYSTEM ARCHITECTURE

A. Overview

HackMatch follows a standard three-tier client-server architecture (Fig. 1). The presentation tier is a React 18 single-page application built with Vite, React Router, Tailwind CSS, and Recharts. It connects to the backend via REST calls made with Axios, and a development proxy forwards /api requests to the backend, avoiding cross-origin problems. The application tier is a FastAPI service (Python 3.11, served by Uvicorn) with automatic OpenAPI documentation and Pydantic validation of request objects. The data tier is MySQL 8.0 accessed with PyMySQL. The trained model is loaded once at startup from a joblib artifact and used for all inference requests, thus avoiding any per-request model loading cost.

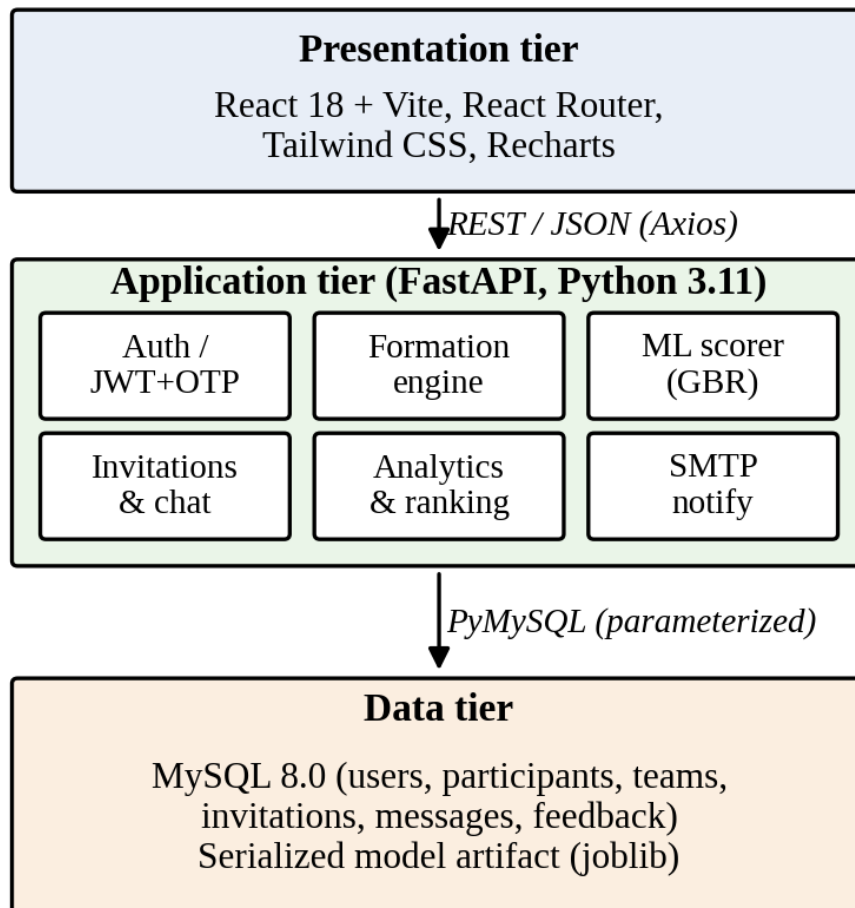


Fig. 1. Three-tier architecture of HackMatch.

B. Frontend

The frontend is organized into 13 pages and 9 reusable components, including profile modals, domain and rating badges, score bars, a floating chat panel, and an assistant widget. A routing layer protects authenticated pages. Shared components render profile cards, invitation cards, chat panels, and analytics widgets, and API calls are centralized so that token handling, error messages, loading states, and response parsing are consistent. The layout is responsive for desktop and mobile browsers.

C. Backend Services

The backend exposes more than 45 REST endpoints in ten groups: authentication, participants, teams, invitations, machine learning, analytics, rankings, administration, feedback, and assistance. Table I lists a representative subset. Separating these responsibilities makes the system easier to test and allows a later move from a monolithic deployment to independently scaled services. The formation endpoint coordinates candidate generation, feature computation, model prediction, persistence, and invitation creation.

TABLE I

REPRESENTATIVE REST ENDPOINTS

Group	Endpoint	Method	Purpose
auth	/auth/register	POST	Create account; send OTP
auth	/auth/verify-otp	POST	Verify email OTP
auth	/auth/login	POST	Authenticate; return JWT
participants	/participants	GET, POST	List or create profiles
teams	/teams/form	POST	Form one new team
teams	/teams/{id}/setup	PATCH	Set team name and leader
invitations	/invitations/my	GET	List own invitations
invitations	/invitations/{id}/respond	PUT	Accept or decline
analytics	/analytics/dashboard	GET	Dashboard data
rankings	/rankings/teams	GET	Team rankings
admin	/admin/notify	POST	Send email notification
feedback	/feedback	POST	Submit feedback

D. Data Model

The data model separates account information from event participation (Table II). The users table stores authentication and verification state, and the participants table stores event-specific profile attributes and team membership. Because of this separation the same account could take part in several events if an event identifier were added; in the current implementation the system is designed for one discrete event, and a future version should add an events table with event_id foreign keys. Schema creation runs at first start-up, and a lightweight migration routine issues ALTER TABLE statements for missing columns so that existing deployments can evolve without manual intervention.

TABLE II

CORE DATA ENTITIES

Entity	Key attributes	Purpose
users	email, password_hash, role, is_verified, otp_expiry	Authentication and verification
participants	skills, domains, preferred_role, availability, experience, rating, team_id	Profile and assignment state
teams	name, balance_score, ml_scores, explanation, status	Team record and evidence
invitations	team_id, participant_id, status, responded_at	Membership proposals and audit trail
messages	team_id, sender_id, text, attachment	Private team chat
feedback	participant_id, ratings, responses	Post-event evaluation

E. Authentication and Security

New accounts are verified by a six-digit one-time password (OTP) sent by SMTP and valid for five minutes; unverified accounts cannot log in. Passwords are hashed with bcrypt [11] at work factor 12. Sessions use JSON Web Tokens [12] signed with HS256 and valid for 24 hours. All SQL statements use parameter binding to reduce injection risk. Database credentials, SMTP passwords, and the token-signing secret are read from environment variables in files excluded from version control, and SMTP credentials are used only on the server. Table III summarizes the controls.

Authorization is enforced on the server. Participants may view and modify their own profile, respond to their own invitations, use their own team chat, and submit their own feedback. Administrators may view aggregate analytics and send notifications. The backend does not rely on frontend route hiding,

because browser-side restrictions can be bypassed. Several further controls are recommended for production and are not claimed as implemented: HTTPS, rate limiting on OTP and login endpoints, validation of uploaded file type and size, storage of uploads outside the source tree, and structured audit logging.

TABLE III

IMPLEMENTED SECURITY CONTROLS

Concern	Implementation
Password storage	bcrypt hash, work factor 12
Sessions	JWT, HS256, 24 h expiry
Email verification	Six-digit OTP, 5 min expiry
SQL injection	Parameterized queries
Secrets	Environment variables; not in repository
SMTP credentials	Server-side only

F. Deployment

The repository includes deployment descriptors for DigitalOcean App Platform, Render, and Railway, and a docker-compose file for local containers. In the platform deployments the backend runs under Uvicorn and the frontend is built to static files with npm run build. Evaluation in this paper used a local server rather than a cloud platform.

V. FORMATION ENGINE

A. Participant Profile and Rating

Each participant supplies the attributes in Table IV. The skill, domain, and role inventories are fixed lists of 15 options each; the skills are Python, Java, C/C++, JavaScript, React.js, Node.js, Machine Learning, Data Science, UI/UX Design, Cloud Computing, Cybersecurity, Database Management, Flutter, DevOps, and Data Analytics. Fixing the inventories makes features f_1 and f_8 well defined and avoids free-text ambiguity, at the cost of excluding skills outside the list.

TABLE IV

PARTICIPANT PROFILE ATTRIBUTES

Attribute	Input	Options
Skills	Multi-select	15 canonical skills
Domain interests	Multi-select	15 domain tracks
Preferred role	Dropdown	15 roles
Experience level	Toggle	Beginner / Intermediate / Advanced
Availability	Number	Hours per day
Hackathon count	Number	0–50
Project count	Number	0–100

A deterministic performance rating in $[1, 5]$ is computed at registration as

$$R = \text{clamp}(b_{\text{exp}} + s + h + p, 1, 5) \quad (2)$$

where b_{exp} is 2.0, 3.5, or 4.5 for beginner, intermediate, or advanced participants; $s = \min(0.1n_s, 1)$ with n_s the number of skills; $h = \min(0.4n_h, 1)$ with n_h the number of past hackathons; and $p = \min(0.25n_p, 0.5)$ with n_p the number of projects. Ratings map to the labels Top Performer (≥ 4.5), Well Performer (≥ 3.5), Active Member (≥ 2.5), and Getting Started (< 2.5). The rating is used as a tie-breaker, as a pre-filter for large pools, and in team rankings. It is not a measure of compatibility or of leadership ability.

B. Compatibility Features

Team compatibility is encoded as eight features, each normalized to [0, 1] (Table V). Their rationale is as follows. Skill coverage (f_1) rewards breadth of technical capability. Complementarity (f_2) is maximal when members bring distinct skills instead of repeating one another. Domain match (f_3) promotes shared problem-space interest and motivation. Role diversity (f_4) penalizes teams made of one role, such as all frontend developers. Availability match (f_5) penalizes large gaps in daily time commitment. Experience balance (f_6) favors mixed experience that supports peer learning. The size score (f_7) peaks at the empirically preferred size of four members, and domain coverage (f_8) rewards breadth of problem areas represented.

TABLE V

COMPATIBILITY FEATURES FOR A TEAM OF SIZE N

ID	Feature	Definition
f_1	Skill coverage	$ \cup S_i / 15$
f_2	Complementary skills	unique skills / $\sum S_i $
f_3	Domain match	pairs sharing ≥ 1 domain / $C(n,2)$
f_4	Role diversity	distinct preferred roles / n
f_5	Availability match	$\min(a_i) / \max(a_i)$
f_6	Experience balance	distinct levels / $\min(3, n)$
f_7	Size score	$1 - n - 4 / 4$
f_8	Domain coverage	$ \cup D_i / 15$

Availability match measures similarity of daily time commitment, not overlap of actual time slots; modeling real schedules is left for future work.

C. Worked Example

Table VI shows a hypothetical team of four (not drawn from the study data) with availabilities of 6, 6, 5, and 8 h/day and experience levels intermediate, beginner, advanced, and intermediate for M1 to M4. Eleven distinct skills appear among twelve skill mentions (Python is repeated), so $f_1 = 11/15 = 0.733$ and $f_2 = 11/12 = 0.917$. Five of the six member pairs share a domain, giving $f_3 = 0.833$. All four roles differ ($f_4 = 1.0$), the availability ratio is $5/8 = 0.625$, all three experience levels occur ($f_6 = 1.0$), the team has the preferred size ($f_7 = 1.0$), and four of the 15 domains are covered ($f_8 = 0.267$). Applying the weights of Eq. (3) without noise gives a value of about 0.82; the trained regressor approximates such a value.

TABLE VI

HYPOTHETICAL TEAM USED IN THE WORKED EXAMPLE

M	Skills	Domains	Role
M1	Python, ML, Data Science	AI, Healthcare	ML Engineer
M2	React.js, JavaScript, UI/UX	Education, AI	Frontend Dev.
M3	Node.js, Databases, Cloud	Healthcare, FinTech	Backend Dev.
M4	Flutter, Python, DevOps	Education, Healthcare	Mobile Dev.

D. Compatibility Model

A Gradient Boosting Regressor [8] implemented in scikit-learn [13] is trained on 10,000 synthetically generated teams, each represented by its eight-feature vector. The target for each synthetic team is a weighted combination of the features plus Gaussian noise:

$$y = 0.22f_1 + 0.18f_2 + 0.18f_3 + 0.14f_4 + 0.12f_5 + 0.08f_6 + 0.04f_7 + 0.04f_8 + \varepsilon, \varepsilon \sim N(0, 0.04) \quad (3)$$

The weights sum to one. They were assigned by the authors to prioritize skill coverage and complementarity, followed by domain match and role diversity, and are a design prior, not weights learned from observed project outcomes. The noise term prevents the model from fitting the exact formula trivially. Hyperparameters, selected by 5-fold cross-validation, are given in Table VII. On a held-out 20% test split the model obtains MAE = 0.032 and $R^2 = 0.94$.

These figures need careful reading. Because the label noise has standard deviation 0.04, the expected absolute error of a perfect predictor of the noiseless formula is $0.04 \cdot \sqrt{(2/\pi)} \approx 0.032$. The reported MAE therefore indicates that the model has learned the generating formula almost to the noise floor. It does

not show that the score predicts real project success. What the learned model adds over the linear formula is the ability to capture interactions and non-linearities once real outcome labels replace the synthetic ones.

TABLE VII

MODEL CONFIGURATION

Parameter	Value
Training samples	10,000 synthetic teams
Estimators	200
Maximum depth	5
Learning rate	0.05
Subsample	0.8
Min. samples per leaf	5
Validation	5-fold CV; 20% held-out test

E. Single-Team Formation

Fig. 2 and Algorithm 1 summarize formation. One formation request creates exactly one team of the requested size k ($2 \leq k \leq 8$). Applying one user's size preference to all remaining participants at once would reshuffle people unexpectedly and could produce partial teams, so the restriction is deliberate. Once a team is created, its members are excluded from later requests; we call this the team lock. A participant who declines is released to the waiting pool and becomes eligible for a later run.

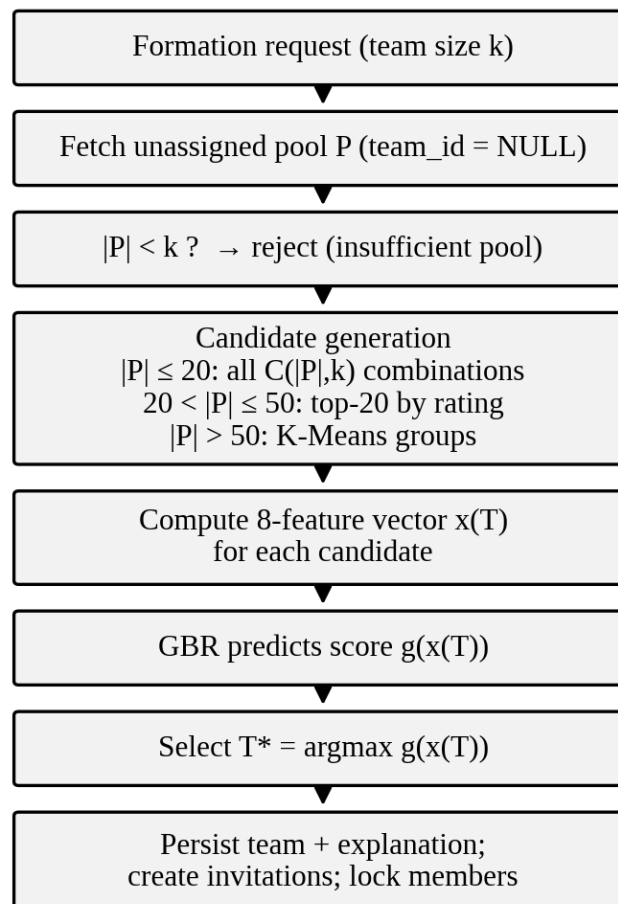


Fig. 2. ML formation pipeline and candidate search.

Algorithm 1: Single-team formation

1. Input: unassigned pool P ($\text{team_id} = \text{NULL}$), team size k .
2. If $|P| < k$, return an insufficient-participants error.
3. If $|P| \leq 20$, enumerate all $C(|P|, k)$ combinations; otherwise enumerate $C(20, k)$ combinations of the 20 highest-rated participants (K-Means groups when $|P| > 50$).
4. For each candidate T , compute $x(T)$ and the score $g(x(T))$.
5. Select $T^* = \arg \max g(x(T))$.
6. Persist T^* with its score, features, and explanation; create one invitation per member; lock the members.
7. Return T^* and the remaining pool as the waiting list.

F. Search-Space Analysis and Clustering

Exhaustive search over a pool of n unassigned participants evaluates $C(n, k)$ candidates, which grows quickly (Fig. 3 and Table VIII). Restricting the search to the 20 highest-rated participants bounds the count at $C(20, k)$: 4,845 candidates for $k = 4$ but 125,970 for $k = 8$. The restriction is a heuristic. It can exclude a participant whose profile would complete an otherwise strong team but whose rating is low, and it therefore trades optimality for speed. In the 25-participant study ($k = 4$), the reported formation time of 1.8 s corresponds to at most 0.37 ms per candidate if it were spent entirely on scoring; in fact database access and persistence contribute, so the true per-candidate cost is lower.

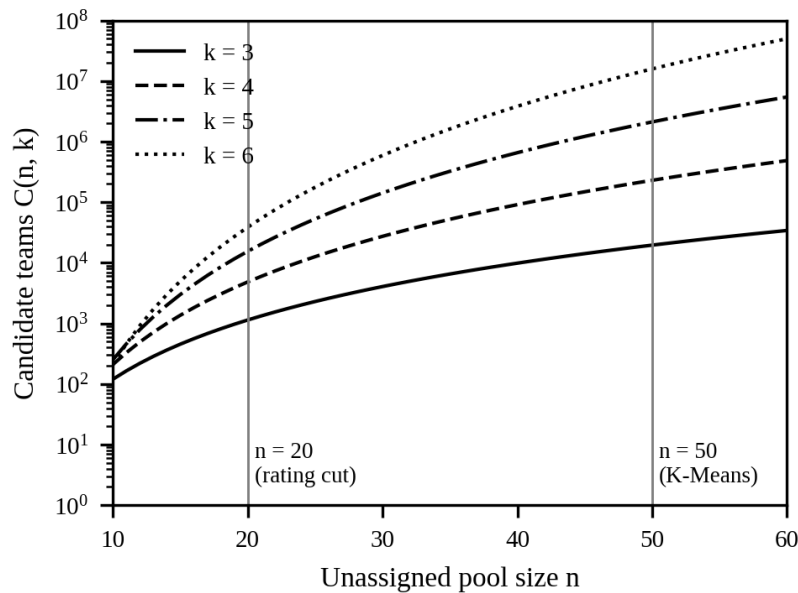


Fig. 3. Number of candidate teams $C(n, k)$ versus pool size.

TABLE VIII

CANDIDATE TEAMS EVALUATED

k	C(20, k)	C(25, k)	C(50, k)
3	1,140	2,300	19,600
4	4,845	12,650	230,300
5	15,504	53,130	2,118,760
6	38,760	177,100	15,890,700

For pools above 50 unassigned participants, HackMatch applies K-Means clustering [14] with $\lfloor n / k \rfloor$ clusters to TF-IDF-style [15] skill and domain vectors and generates candidates within or around clusters. Members of a cluster share related skills and interests, so scoring effort is concentrated on plausible teams, and the search per cluster is $C(\text{cluster size}, k)$. Clustering is sensitive to initialization, and better seeding such as k-means++ [16] is a natural refinement. The clustering path is implemented but was not benchmarked in the present evaluation, which used pools of at most 25 participants; we therefore make no quality or speed claim for it.

G. Explanations

Each team receives a natural-language explanation generated from its feature values by threshold rules; for example, a statement on skill coverage is emitted when $f_1 \geq 0.7$, on complementarity when $f_2 \geq 0.8$, and on role diversity when $f_4 \geq 0.75$. The text names measurable factors, such as the number of distinct skills and the list of roles, and is stored with the team record. It is displayed to members in the team view and invitation card. These explanations are rule-based descriptions of the features and not a learned or post-hoc explanation of the model.

VI. PARTICIPANT WORKFLOW

A. Registration and Formation Request

The participant logs in, confirms the email address via OTP, and fills in their profile. An organizer or another participant can then choose to request a team of a specific target size within a defined range. The backend looks for eligible participants who are not yet placed into a team, confirms that the pool size matches the target size, prepares candidates, builds feature vectors, predicts, and finally selects the most matching participant. The team record contains the score, features contributing to the score, covered skills and roles, and the match explanation.

B. Invitation Workflow

Each candidate gets an in-app invitation, with a visual indicator in the navigation bar showing how many invitations are pending, and an email. The invitation card displays the team name and the match explanation. Fig. 4 and Table IX below represent the invitation life cycle. Accepting creates the team, while declining removes the tentative assignment and puts the participant back in the pool, with the declined invitation stored for audit control. If there are enough candidates, the formation can happen multiple times, thus filling the vacancy.

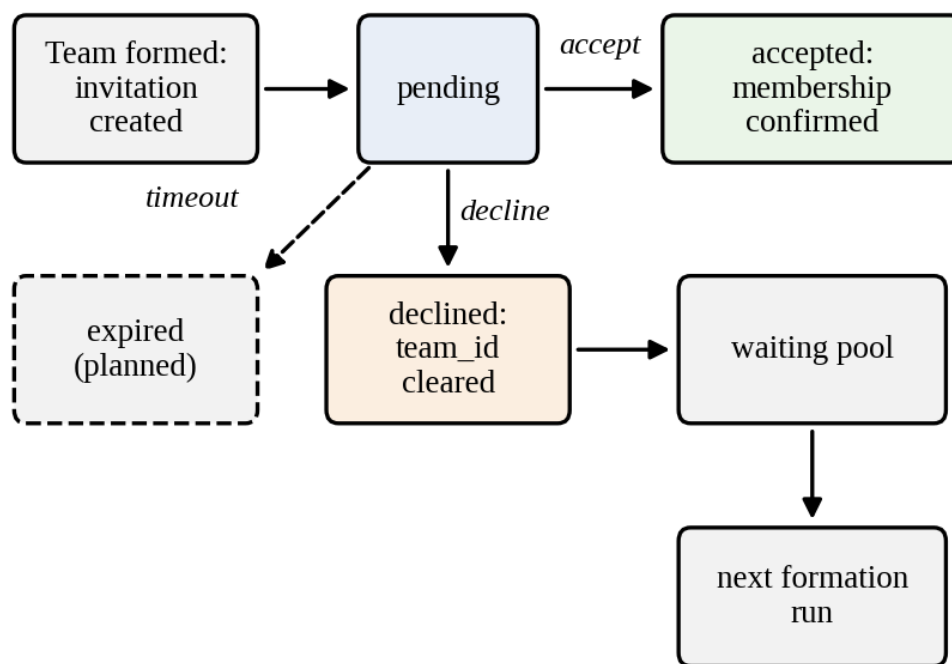


Fig. 4. Invitation life cycle. Dashed elements are to be implemented.

TABLE IX

IMPLEMENTED INVITATION TRANSITIONS

State	Event	Target	System action
pending	Accept	accepted	Confirm membership; update progress
pending	Decline	declined	Clear team_id; return to waiting pool

An expiry state, where unanswered invitations become expired and notify the organizer so that the vacancy can be filled, is not yet implemented and is considered future work. Similarly, a stricter formation policy will make teams active only when a certain percentage of members have accepted, whereas now, each member's reaction is processed individually.

C. Team Setup

The team members can then configure the team, naming it and choosing one of their members as the leader via a setup dialog. The backend validates whether the selected leader is part of the team, while the chosen name is then propagated to the member records. The leader is not selected automatically, for instance, based on the highest score, to avoid rewarding the participants' strategic behavior, since the selection score is not based on the leader's skills but rather their overall match to the team needs.

D. Privacy-Aware Discovery

The participants page lists only the current user's profile by default and allows searching for others based on name, skills, domains, and roles. This approach prevents recommendation-driven skimming through profiles and contacting those who have not accepted an invitation. The profiles that match the search criteria are listed below the search bar in a scrollable list. In its current state, the filtering is implemented in the client, meaning that all records are downloaded to the browser, where they are filtered against search queries. In a production environment, the filtering should be implemented on the server side with pagination to ensure that the dataset is not too large. Similarly, the email addresses should not be publicly visible by default but rather masked or completely hidden until the participant agrees to share their details.

VII. APPLICATION MODULES

A. Analytics Dashboard

The administrator dashboard tracks the overall, assigned, waiting, and pending counts, team invitation outcomes with a special indicator if someone was left waiting after decline, formation progress as the percentage of filled positions, cards with member count, balance score, and completeness, the waiting list with a search option, the distribution of the twelve most common skills, and the activity feed. The endpoints return only the necessary data for each widget and utilize role-based checks to ensure that only the relevant data is visible to the administrators. The organizers used the waiting-list dashboard to form teams based on the vacancy fill schedule.

B. Rankings

The teams are ranked based on the average member score, scaled up to 0-100 (x20), with labels excellent, strong, good, and needs improvement, with the corresponding thresholds. The teams can be viewed in podium and table views, with the table being sorted by the rank and searchable. The ranking is updated whenever the team members change but does not correlate to the compatibility score or the projected outcome of the project. It is a performance ranking based on the participants' skills and should not be confused with the quality assessment of the project itself. The future feature of outcome assessment and the comparison of the projected outcome to the actual project quality can be implemented in a later stage.

C. Team Chat

The team chat is available for each formed team, with message threads, usernames, timestamps, emojis, file attachments, and read receipts. The component is accessible via a floating chat icon at the bottom of the screen. Messages can be sent as text or images with appropriate size limitations, media types, permissions, and virus scanning in production.

D. Email and Feedback

The notification module handles the sending of emails for team assignment, invitation acceptance, team creation confirmation, reminders, and other notifications with customizable templates. The emails are sent via SMTP with an application password, which should be kept secure on the server. The post-event feedback form asks eleven questions and collects responses in star ratings, multiple-choice options, and open-text fields to analyze participant satisfaction, skill balance, domain fit, communication, contribution, and likelihood of reuse. The responses should be anonymized if possible or kept private with consent to use the data for research purposes before being analyzed.

VIII. EXPERIMENTAL EVALUATION

A. Setup

HackMatch was deployed on a local server and used by 25 university students who registered independently over two weeks and completed full profiles. Six formation runs were performed with team sizes of three and four. We recorded balance score, skill coverage, domain overlap, role diversity, availability match, and response time. Because the cohort is small and no randomized or skill-only baseline was run, this is a feasibility study and not a comparative evaluation.

B. Formation Quality

TABLE X

FORMATION QUALITY OVER SIX TEAMS

Metric	Observed value
Average balance score	0.73 / 1.00
Average skill coverage	58.9%
Average domain overlap	67.0%
Average role diversity	0.75
Average availability match	1.00
Teams formed	6

The balance score of 0.73 indicates moderate compatibility. Skill coverage is bounded by the 15-skill inventory and by teams of only three or four members: a team of four with three skills each can cover at most 12 of 15 skills, or 80%, so 58.9% is well below the ceiling but not negligible. The availability match of 1.00 arises because all participants in the sample reported identical availability, so this feature carried no discriminating information in this experiment. Invitation responses were recorded, and every declined invitation returned the participant to the waiting list as designed.

C. Computational Performance

TABLE XI

AVERAGE RESPONSE TIME

Operation	Time
Participant registration	120 ms
Formation (n = 25, k = 4)	1.8 s
Formation (n = 10, k = 3)	0.35 s
Dashboard data load	280 ms
Team ranking computation	95 ms
Email notification send	2.1 s
Invitation response update	45 ms

Formation latency is suitable for interactive use at the tested scale. With $n = 25$ the rating cut leaves 20 participants, so 4,845 candidates were scored in 1.8 s. With $n = 10$ and $k = 3$ only 120 candidates exist, yet formation took 0.35 s. The run time therefore does not scale in proportion to the candidate count, which suggests a substantial fixed component from database access and persistence; this should be verified by profiling. Email sending is slowest because it depends on an external SMTP service. Larger studies should report latency distributions and separate candidate-generation, database, inference, and clustering costs. Because $C(20, 8)$ exceeds 125,000, teams of eight may need batched scoring or a smaller cut to remain interactive.

D. Informal Observations

During the study, the team recorded informal observations, without a structured instrument. Participants did not report feeling that their contact details were exposed under the search-first participants page. In all six cases team members completed the setup dialog within five minutes of formation. The dashboard let the organizing committee monitor formation status without direct database access, and the email templates reduced the time spent drafting messages. These observations are anecdotal and are not evidence of improved outcomes.

IX. DISCUSSION, LIMITATIONS, AND ETHICS

A. Comparison with Existing Practice

Compared with manual formation, HackMatch removes the effort of the coordinator and produces reproducible and explainable assignments. Compared with keyword matching, it also considers availability, experience, and role balance. Compared with graph-based methods, it requires no prior social

relationships and is therefore usable for new cohorts. Compared with constraint-based methods, its scoring needs no per-event constraint specification, although the weights embody a fixed design prior. These advantages derive from the design and have not yet been evaluated against controlled baselines.

B. Limitations

The main limitation is synthetic supervision: the target is generated from a fixed formula, so the model can reproduce that formula but cannot show that its score predicts project quality or satisfaction. The evaluation of 25 participants is too small for statistical generalization, and no random, skill-only, or constraint-based baseline was compared. Soft skills, communication style, and work habits are not modeled, the availability feature does not capture time-slot overlap, the rating-based cut may exclude good candidates, and the system is designed for a single event without cross-event history.

C. Threats to Validity

Construct validity: the balance score is the system's own objective, so a high score shows that the optimizer works, not that the teams perform well. Internal validity: without a control group we cannot attribute any participant experience to the algorithm rather than to the interface or novelty. External validity: the sample is one university group with uniform availability, so results may not transfer to corporate or multi-institution events with varied schedules. Statistical validity: with six teams, averages carry wide uncertainty, and no confidence intervals are reported.

D. Ethical and Privacy Considerations

Team formation affects participants' opportunities, so human oversight, correction of errors, and explanations are important. Profile fields should be limited to what formation needs, sensitive attributes should not act as hidden proxies, and messages, emails, and feedback require access control, retention limits, and informed consent for research use. Explanations should describe measurable factors without revealing another participant's private information. Because the performance rating is derived from experience counts, it could disadvantage newcomers if used beyond tie-breaking and ranking; the system therefore keeps it out of the compatibility score.

E. Future Work

Priorities are: collecting project scores and satisfaction ratings to retrain the model on real labels; comparing against random, skill-only, and constraint-based baselines with confidence intervals; modeling availability as time-slot overlap; adding an invitation-expiry state and an activation rule; profiling and benchmarking the clustering path; supporting multiple tracks and mentor matching; recommending replacements after declines; and adding validated soft-skill instruments.

X. CONCLUSION

This paper presented HackMatch, a full-stack system for hackathon team formation that combines eight compatibility features, Gradient Boosting scoring, candidate search with K-Means reduction for large pools, rule-based explanations, invitation management, and event-support modules. A preliminary study with 25 participants gave a mean balance score of 0.73 and formation in under two seconds. These results demonstrate technical feasibility but not improved hackathon outcomes, because the model uses synthetic labels and no controlled baseline was evaluated. Validation on real outcome data with larger cohorts is the next step.

AVAILABILITY

The source code is available at <https://github.com/722007aarthi/Hackmath>. For reproducibility, a tagged release matching this paper should document software versions, random seeds, the synthetic-data generation procedure, the train-test split, and the candidate-generation limits, and it should exclude passwords, API keys, and participant data.

REFERENCES

- [1] T. Lappas, K. Liu, and E. Terzi, "Finding a team of experts in social networks," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Paris, France, 2009, pp. 467–476.
- [2] M. Kargar and A. An, "Discovering top-k teams of experts with/without a leader in social networks," in *Proc. 20th ACM Int. Conf. Inf. Knowl. Manage.*, 2011, pp. 985–994.
- [3] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi, "Online team formation in social networks," in *Proc. 21st Int. Conf. World Wide Web*, Lyon, France, 2012, pp. 839–848.
- [4] C.-T. Li and M.-K. Shan, "Team formation for generalized tasks in expertise social networks," in *Proc. IEEE 2nd Int. Conf. Social Comput.*, 2010, pp. 9–16.
- [5] J. Porras *et al.*, "Hackathons in software engineering education: Lessons learned from a decade of events," in *Proc. 2nd Int. Workshop Softw. Eng. Educ. Millennials*, 2018, pp. 40–47.
- [6] A. Nolte *et al.*, "You hacked and now what? Exploring outcomes of a corporate hackathon," *Proc. ACM Hum.-Comput. Interact.*, vol. 2, no. CSCW, Art. 129, 2018.

-
- [7] I. Lykourantzou *et al.*, “Personality matters: Balancing for personality types leads to better outcomes for crowd teams,” in *Proc. ACM Conf. Comput.-Supported Cooperative Work Social Comput.*, 2016, pp. 260–273.
- [8] J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, 2001.
- [9] M. T. Ribeiro, S. Singh, and C. Guestrin, ““Why should I trust you?”: Explaining the predictions of any classifier,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1135–1144.
- [10] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in *Proc. Adv. Neural Inf. Process. Syst. 30*, 2017, pp. 4765–4774.
- [11] N. Provos and D. Mazières, “A future-adaptable password scheme,” in *Proc. USENIX Annu. Tech. Conf.*, 1999, pp. 81–91.
- [12] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” IETF, RFC 7519, May 2015.
- [13] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
- [14] S. P. Lloyd, “Least squares quantization in PCM,” *IEEE Trans. Inf. Theory*, vol. IT-28, no. 2, pp. 129–137, Mar. 1982.
- [15] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” *Inf. Process. Manage.*, vol. 24, no. 5, pp. 513–523, 1988.
- [16] D. Arthur and S. Vassilvitskii, “k-means++: The advantages of careful seeding,” in *Proc. 18th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2007, pp. 1027–1035.