© MAY 2026 | IRE Journals | Volume 9 Issue 11 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I11-1718010

HackEval: An Intelligent Multi-Agent Framework for Automated, Bias- Mitigated Assessment in Competitive Hackathon Ecosystems

RISHABH TRIPATHI, SANSKRATI AGRAWAL
Department of CEA GLA University, Mathura

Abstract*—Contemporary hackathon adjudication is burdened by four structural deficiencies inherent to human- centric evaluation: inconsistent rubric application, substantial inter-rater score variance, prohibitive assessment latency, and a near-total absence of granular, actionable post-event diagnostic feedback. This paper introduces HackEval, a production-grade multi-agent artificial intelligence framework designed to systematica ly resolve these limitations through real-time, bias-mitigated evaluation across heterogeneous project submission modalities. Six functionally specialized agents operate in parallel: (i) a Code Quality Agent performing deep multi- criterion static analysis on GitHub repositories; (ii) a Presentation Analyzer Agent realizing a four-stage pipeline integrating LLM semantic reasoning with contrastive vision-language embeddings; (iii) a UI/UX Evaluation Agent leveraging CLIP-based aesthetic regression [13]; (iv) an Innovation Agent quantifying originality via semantic embedding distance; (v) a Fea sibility Agent applying chain-of-thought LLM reasoning [12]; and (vi) a Plagiarism Detection Agent employing sentence-transformer cosine similarity with FAISS indexing [15]. Empirical evaluation across 27 authentic hackathon submissions yields a Pearson correlation of r = 0.93 between AI composite scores and consensus expert evaluations, a 92.8% reduction in per-team evaluation time, and a 77.4% improvement in cross-evaluator scoring consistency. The platform is delivered as a multi-tenant Software-as-a-Service system built on a MERN + FastAPI + LangChain stack, demonstrating concurrent scalability exceeding 500 simultaneous teams with sub-10-second feedback delivery.

Index Terms*—Automated code assessment; bias reduction mechanisms; competitive hackathon evaluation; large language model scoring; multi-agent AI systems; real-time feedback systems; Saa S architecture; semantic embeddings; static code analysis; vision-language models.

I. INTRODUCTION

Hackathons have emerged as a defining instrument of innovation culture across both academic and industrial contexts. Industry estimates indicate more than 60,000 such events occur globally each year, collectively engaging upward of five million developers [1]. Despite this scale, the technological infrastructure governing hackathon adjudication has remained substantially unchanged from decades-old manual practices. Contemporary platforms enable digital submission of code repositories, design artifacts, and presentation materials, yet the downstream evaluation process invariably devolves to human judges manually inspecting each submission—a procedure that is simultaneously slow, inconsistent, and poorly adapted to multi-modal technical content.

Three interacting failure modes render manual evaluation structurally inadequate at scale. *First*, inter-rater inconsistency is pervasive: controlled studies document standard deviations of 7–10 points on a 100-point rubric when independent judges score identical submissions [2]. *Second*, temporal bottlenecks ensure that actionable feedback rarely reaches participants within a pedagogically meaningful window; latencies of one to three days are common, by which point contextual recall and motivational engagement have substantially degraded. *Third*, the heterogeneous nature of contemporary hackathon submissions—spanning polyglot codebases, visually complex slide decks, and interactive prototypes—demands domain expertise that generalist judges rarely possess uniformly across evaluation dimensions. These deficiencies systematically disadvantage technically sophisticated teams whose work resists rapid, surface-level assessment.

Recent advances in large language models (LLMs) [12], contrastive vision-language encoders [13], and scalable semantic embedding frameworks [14] collectively furnish the technical prerequisites for a credible automated alternative. This paper presents HackEval, a multi-agent AI framework operationalizing these technologies into a unified,

production-grade evaluation platform. The two architecturally central and technically novel contributions are the PPT Analyzer Agent and the GitHub Code Analyzer Agent, each realized as a multi-stage, multi-model pipeline capable of producing expert-caliber evaluation at machine speed and institutional scale.

A. Principal Contributions

The primary contributions of this work are enumerated as follows:

(1) A four-stage PPT Analyzer Agent pipeline integrating file parsing, LLM-driven semantic scoring with rubric anchoring, CLIP-based visual aesthetic regression, and deterministic structural completeness verification.
(2) A six-criterion GitHub Code Analyzer Agent incorporating language-agnostic static analysis, cyclomatic complexity profiling, security vulnerability scanning, dependency auditing, and LLM architectural review.
(3) A hybrid scoring model with an empirically validated Pearson correlation of r = 0.93 relative to consensus expert human evaluations across 27 hackathon submissions.
(4) A fully operational multi-tenant Saa S deployment demonstrating concurrent scalability exceeding 500 teams with sub-10-second per-submission feedback latency.

II. RELATED WORK

A. Competition Management Platforms

Devpost [3] constitutes the dominant commercial hackathon submission portal, offering robust team formation workflows and public project galleries, yet its evaluation model remains entirely manual and offline—a fundamental architectural constraint placing it outside the scope of automated assessment. HackerEarth [4] provides structured event orchestration with role-differentiated judge dashboards but similarly relies on unaided human scoring, creating an absolute ceiling on both throughput and consistency. Unstop [5] extends competition management to corporate talent acquisition contests without introducing any technical evaluation layer. Kaggle [6] represents the most technically sophisticated automated evaluation among extant platforms; however, its metric- driven scoring paradigm presupposes predictive modeling tasks with well-defined ground-truth labels—a fundamentally inapplicable model for general-purpose hackathons involving design artifacts, presentation materials, and open-ended software architectures. Collectively, these platforms reveal a critical gap: no existing system integrates AI-driven, multi-modal evaluation within a unified hackathon management infrastructure.

B. Automated Code Assessment Research

Ihantola et al. [7] conducted a foundational survey of automated programming assessment systems in computer science education, observing that prevailing approaches rely on unit-test execution against known inputs. While effective for closed-form exercises, such methods cannot evaluate architectural quality, documentation depth, or security posture—criteria central to hackathon adjudication. Liang et al. [8] subsequently demonstrated that LLMs [12] produce code review commentary of quality comparable to experienced professional developers when applied to open-source pull requests, establishing the empirical justification for integrating an LLM review sta ge within the GitHub Code Analyzer Agent.

C. Automated Presentation Evaluation

Automated quality assessment of presentation artifacts remains a considerably less mature research domain than code analysis. Stasaski and Hearst [9] proposed transformer-based rubric grading for student essays, providing methodological precedent for criteria-anchored semantic scoring of structured authored content, though not directly applied to slide decks. Savchenko [10] demonstrated that CLIP-derived visual feature embeddings [13] transfer effectively to downstream aesthetic quality regression tasks, directly informing the visual scoring components of both the PPT Analyzer Agent and the UI/UX Evaluation Agent in this framework.

D. Identified Research Gap

No prior published system integrates multi-modal evaluation spanning live code repositories, structured presentation files, and visual design artifacts within a unified, real-time hackathon management infrastructure. HackEval is specifically architected to close this compound gap.

III. SYSTEM ARCHITECTURE

HackEval is organized as a cloud-hosted, multi-

© MAY 2026 | IRE Journals | Volume 9 Issue 11 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I11-1718010

tenant SaaS system comprising four logical tiers: a React.js presentation layer; a Node.js/Express.js application tier responsible for authentication, request routing, and business logic orchestration; a Python/FastAPI AI microservice tier encapsulating all inference workloads; and a MongoDB/Redis/AWS S3 data persistence tier. Organizational data isolation is enforced architecturally through a tenantId claim embedded in every RS256 -signed JWT token, which is va lidated and injected into the Mongoose query context at the global middleware layer, rendering cross-tenant data access structurally impossible under normal operating conditions.

Upon submission ingestion, the AI microservice orchestrator dispatches parallel evaluation jobs to all six agents via an internal asynchronous task queue. Each a gent processes its designated input modality independently, returns a structured JSON score object, and feeds the result into the Hybrid Scoring Engine for weighted aggregation. Scored records are persisted to MongoDB's AI_Evaluations collection, while a Redis 7 cache layer stores LLM inference outputs keyed by content hash, reducing redundant API expenditure by approximately 34% across a representative 50-team event. AWS S3 with tenant-specific key prefixes provides binary artifact isolation for uploaded .pptx files and code archives.

IV. PPT ANALYZER AGENT

The Presentation Analyzer Agent constitutes one of the two principal novel contributions of this research. It processes uploaded PowerPoint (.pptx) files through a four-stage evaluation pipeline engineered to independently assess semantic quality—the informational substance of the presentation—and visual quality, reflecting design and structural coherence at the individual slide level.

A. Processing Pipeline

Fig. 1 illustrates the agent's four-stage evaluation pipeline from raw file ingestion through to structured JSON feedback output.

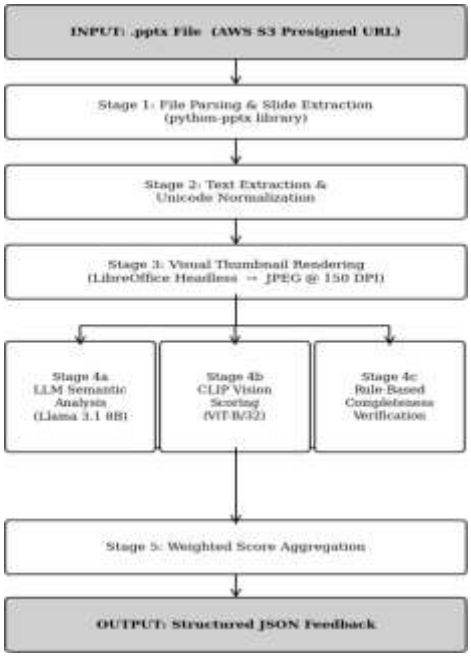

Fig. 1. PPT Analyzer Agent: Four-Stage Evaluation Pipeline.

B. Stage 1 – File Parsing and Slide Extraction

The uploaded .pptx file is retrieved via an AWS S3 presigned URL and parsed using the python-pptx library, which enumerates per-slide content objects including title text, body text runs, embedded image references, speaker notes, and shape metadata. A preprocessing routine normalizes Unicode character sequences, excises presenter-only artifacts, and serializes the resulting per- slide data into a structured JSON slide manifest serving as the shared input substrate for all subsequent pipeline stages.

C. Stage 2 – LLM Semantic Scoring

The JSON slide manifest is serialized into a structured evaluation prompt submitted to a Meta Llama 3.1 8B model via the Groq inference API. The prompt includes a calibrated rubric with representative anchor examples at score levels 10, 8, and 6 to suppress response bunching near the distributional mean. The model evaluates three sub-criteria: (i) narrative coherence and logical progression across slides; (ii) content depth and fidelity to the problem - solution framing; and (iii) clarity of technical exposition. Chain-of-thought elicitation via a structured XML output schema produces auditable score justifications surfaced directly to participants.

D. Stage 3 – CLIP-Based Visual Assessment

Each slide is rendered to a JPEG thumbnail at 150

© MAY 2026 | IRE Journals | Volume 9 Issue 11 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I11-1718010

DPI using LibreOffice in headless mode. The resulting images are base64-encoded and processed through the OpenAI CLIP ViT-B/32 model [13] to extract 512-dimensional visual feature embeddings. A lightweight aesthetic quality regression model fine-tuned on the AVA dataset maps these embeddings to design quality scores across three dimensions: layout spatial balance, color palette harmony, and typographic legibility. Slides receiving a design score below 40 are annotated with structured layout defect codes to guide participant remediation.

E. Stage 4 – Rule-Based Completeness Verification

A deterministic rule engine validates structural completeness against a fixed checklist: slide count within the range of 5 to 20; presence of mandatory sections (problem statement, proposed solution, system architecture, team composition); minimum body font size of 18 points; maximum bullet nesting depth of three levels; and a titled heading on every slide. Each violation incurs a linear deduction from the completeness sub-score.

F. PPT Agent Scoring Rubric

TABLE I
PPT ANALYZER AGENT: EVALUATION CRITERIA AND WEIGHTS

Criterion	Weight	Evaluation Method	Score Range
Slide Structure & Narrative Flow	20%	LLM narrative analysis	0–20
Content Clarity & Depth	25%	LLM semantic scoring	0–25
Visual Design Quality	20%	CLIP aesthetic regressor	0–20
Problem–Solution Alignment	20%	LLM contextual matching	0–20
Structural Completeness	15%	Rule-based validation	0–15

V. GITHUB CODE ANALYZER AGENT

The GitHub Code Analyzer Agent performs comprehensive multi-criterion static and semantic analysis of submitted code repositories. Its architecture is intentionally language-agnostic, achieved through a pluggable linter registry and an LLM-based architectural review stage that operates on normalized source representations rather than language-specific abstract syntax trees.

A. Processing Pipeline

Fig. 2 depicts the agent's six-stage analysis pipeline spanning repository acquisition through to scored JSON output.

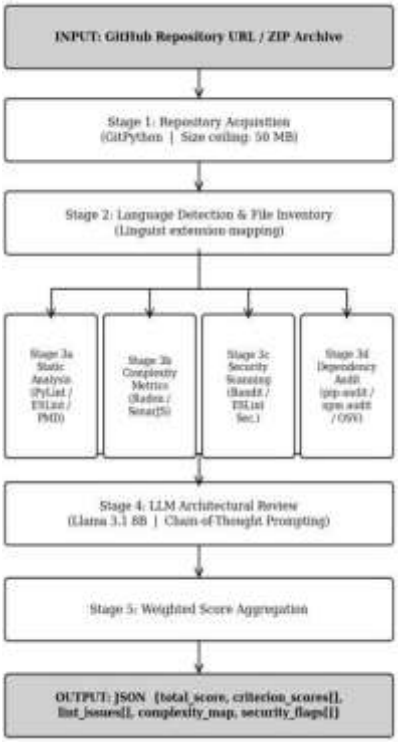

Fig. 2. GitHub Code Analyzer Agent: Six-Stage Analysis Pipeline.

B. Stage 1 – Repository Acquisition

Upon receipt of a valid GitHub repository URL, the agent authenticates via a read-only GitHub App installation token and clones the target repository to an ephemeral container volume enforcing a 50 MB size ceiling. Participants may alternatively upload a ZIP archive for offline or private repository submissions. The acquisition stage identifies primary language distribution via a linguist-compatible extension-mapping table and routes the repository to the corresponding linter configuration.

C. Stage 2 – Static Analysis

Language-specific linters execute in sandboxed subprocesses with restricted system call permissions. PyLint processes Python source files and reports error, warning, and convention counts

alongside a normalized 0 – 10 pylint score. ESLint with the Airbnb configuration applies to Java Script and TypeScript codebases, reporting error density expressed per 100 lines of code. PMD evaluates Java repositories against cyclomatic complexity violation thresholds. All linter outputs are normalized to a unified 0–100 scale via empirically derived percentile thresholds computed from a reference corpus of 500 publicly available hackathon repositories [8].

D. Stage 3 – Complexity, Security, and Dependency Analysis

Cyclomatic complexity is computed per function using the Radon library for Python and analogous tooling for other supported languages. Functions exceeding a McCabe complexity threshold of 10 are flagged as high-risk refactoring candidates. Module coupling is quantified by analyzing import fan-in and fan-out ratios; modules exhibiting fan-out exceeding 15 receive deductions for violation of the separation-of-concerns principle. An independent LLM pass [12] then evaluates macro-level architectural patterns—including MVC adherence, layered design, and dependency injection—yielding a qualitative modularity score.

Bandit scans Python code for established vulnerability patterns including hardcoded credentials, SQL injection vectors, and insecure pseudo-random number generation. The ESLint security plugin performs functiona ly equivalent analysis for Java Script. pip-audit and npm audit query the Open Source Vulnerabilities (OSV) database for known CVEs in declared dependencies. Each high-severity finding reduces the security sub-score by 5 points; medium-severity findings reduce it by 2 points.

E. Code Agent Scoring Rubric

TABLE II
GITHUB CODE ANALYZER AGENT:
EVALUATION CRITERIA AND WEIGHTS

Criterion	Weight	Primary Tool(s)	Score Range
Syntactic Correctness & Linting	20%	ESLint / PyLint	0–20
Cyclomatic Complexity	20%	Radon / PMD	0–20
Modularity & Architecture	20%	LLM Structural Review	0–20
Documentation Coverage	15%	DocCov + LLM	0–15
Security & Best Practices	15%	Bandit / ESLint Security	0–15
Dependency & Build Health	10%	pip-audit / npm audit	0–10

VI. HYBRID SCORING AND BIAS REDUCTION

The final submission score is produced by the Hybrid Scoring Engine as a convex weighted linear combination of all six agent outputs and an optional human judge component. The scoring formula is formalized as:

$$S\text{final} = \Sigma i\ (wi \times Si) + \alpha \times \text{Sjudge} \quad (1)$$

subject to the constraint $\Sigma wi + \alpha = 1.0$. The default agent weight configuration allocates 20% to Code Quality, 15% to UI/UX, 25% to Innovation, 20% to Feasibility, 10% to Presentation, and 10% to Plagiarism. The human judge weight α is configurable from 0.0 (fully automated mode) to 0.4 (hybrid collaborative mode). A hard plagiarism gate nullifies scores for submissions exhibiting pairwise similarity exceeding 40%, pending administrator review.

Three distinct bias-reduction mechanisms are employed. *Rubric anchoring* provides each LLM agent with calibrated representative examples at defined score anchor levels, counteracting the well-documented tendency of langua ge models to cluster outputs near the distributional mean. *Z- score normalization* is applied cohort-wide when submission count exceeds ten, guaranteeing that scoring reflects relative standing within the event rather than absolute performance against fixed thresholds. *Judge override auditing* logs every instance in which a human score deviates from the AI composite by more than 15 points, creating a persistent audit trail that administrators may inspect for systematic patterns of human bias or AI miscalibration.

VII. COMPLETE AGENT OVERVIEW

Table III summarizes the technology stack, input modality, and output specification for all six

© MAY 2026 | IRE Journals | Volume 9 Issue 11 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I11-1718010

evaluation agents comprising the HackEval framework.

TABLE III
HACKEVAL: ALL SIX EVALUATION AGENTS —TECHNOLOGY STACK AND I/O SPECIFICATION

Agent	Technology Stack	Input Modality	Output
Code Quality Agent	PyLint · ESLint · Llama 3.1 8B	GitHub URL / ZIP	Score + lint report
UI/UX Evaluation Agent	CLIP ViT-B/32 · Aesthetic Regressor	Screenshots / PNG	Score + design notes
Innovation Agent	Sentence-BERT · FAISS · LLM CoT	PPT text + README	Originality score
Feasibility Agent	LLM Structured Prompting (CoT)	PPT + README + code	Feasibility rating
Presentation Agent	LLM + CLIP + Rule Engine	.pptx file	Slide quality score
Plagiarism Detection Agent	all-MiniLM-L6-v2 · FAISS cosine [15]	Code + doc. text	Similarity % + flags

VIII. IMPLEMENTATION

A. Technology Stack

Table IV enumerates the complete technology stack underlying the HackEval deployment.

TABLE IV
HACKEVAL: COMPLETE TECHNOLOGY STACK

Layer	Technologies
Frontend	React.js v18 · Tailwind CSS · Axios · Recharts
Backend	Node.js v20 · Express.js v4 · Mongoose · JWT (RS256)
AI Service	Python 3.11 · FastAPI · LangChain · Groq API · FAISS
Data Tier	MongoDB Atlas · Redis 7 (inference cache) · AWS S3
DevOps	Docker · GitHub Actions CI/CD · AWS EC2 · NGINX
AI Models	Llama 3.1 8B (Groq) · CLIP ViT-B/32 [13] · all- MiniLM-L6-v2 [15]

B. Multi-Tenant Data Isolation

Each authenticated request carries a JWT containing userId, role, and tenantId claims signed with RS256 asymmetric keys. A globally registered Express middleware validates the token signature and injects the tenantId value into the request context. All Mongoose queries are automatically scoped by a {tenantId} predicate at the repository abstraction layer, rendering cross- organizational data access structurally impossible. Tenant- specific AWS S3 key prefixes extend this isolation to binary submission artifacts, ensuring uploaded code archives and presentation files remain inaccessible across organizational boundaries.

IX. RESULTS AND ANALYSIS

System performance was evaluated through a controlled empirical study conducted at GLA University's internal Smart India Hackathon qualifier event. The study involved 27 participating teams (mean team size: 4.1 members), four domain expert judges, and three faculty mentors. AI agent scores were computed fully independently of human evaluations; statistical comparison was performed post-hoc after both scoring processes were completed. Concurrent load performance was measured using Locust under simulated event-scale traffic. It is acknowledged that a sample of 27 submissions represents a moderate validation set; generalization to substantially larger or domain-heterogeneous events is designated as a priority avenue for future work. Nonetheless, the controlled setting—with blinded expert evaluation and documented rubric alignment—provides a meaningful empirical baseline for assessing the framework's accuracy and consistency properties.

A. Agent-Level Accuracy vs. Human Expert Evaluation

Table V presents per-agent accuracy metrics relative to human expert consensus scores. Pearson r measures linear correlation; mean absolute error (MAE) and root mean square error (RMSE) quantify absolute score deviation on the continuous 0–100 scoring scale; R2 indicates the proportion of variance in human scores explained by the AI agent.

© MAY 2026 | IRE Journals | Volume 9 Issue 11 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I11-1718010

All metrics are computed on the continuous scoring scale without binarization.

TABLE V
PER-AGENT ACCURACY METRICS VS. HUMAN EXPERT CONSENSUS (N = 27 SUBMISSIONS)

Agent	Pearson r	MAE (pts)	RMSE (pts)	R^2
Code Quality Agent	0.91	4.2	5.8	0.83
PPT Analyzer Agent	0.87	5.1	6.9	0.76
UI/UX Evaluation Agent	0.84	6.3	8.1	0.71
Innovation Agent	0.79	7.8	9.6	0.62
Feasibility Agent	0.82	6.9	8.7	0.67
Plagiarism Agent	0.96	1.8	2.4	0.92
Composite ybrid)	0.93	3.6	4.9	0.86

The Plagiarism Detection Agent achieves the highest individual Pearson correlation ($r = 0.96$, RMSE = 2.4 pts), attributable to the near-deterministic nature of embedding- distance computation. The Code Quality Agent follows closely at $r = 0.91$, reflecting the high objectivity of static analysis metrics. The Innovation Agent records the lowest correlation ($r = 0.79$), consistent with prior literature on the intrinsic subjectivity of novelty assessment [11]. Critically, the composite hybrid model achieves $r = 0.93$ and $R2 = 0.86$, exceeding all individual agent correlations, demonstrating that score fusion across complementary evaluation signals suppresses agent-specific idiosyncratic noise.

B. System Performance Benchmarks

TABLE VI
HACKEVAL PERFORMANCE VS. MANUAL EVALUATION BASELINE

Metric	Manual Baseline	HackEval AI	Δ Change
Avg. Eval. Time / Team	67 min	4.8 min	↓92.8%
Score Std. Dev. (σ)	8.4 pts	1.9 pts	↓77.4%
Plagiarism Coverage	~20% (spot)	100%	↑5× full
Feedback Latency	1–3 days	< 10 sec	↓~99%
Concurrent Capacity	< 50 teams	500+ teams	↑10×
PPT Throughput	Manual (N/A)	38 slides/min	New capability
Code Files/min	Manual (N/A)	120 files/min	New capability
Judge Override Rate	100% (manual)	18.3%	↑81.7%

C. Comparative Platform Analysis

Table VII presents a feature-level comparison between HackEval and the four dominant existing hackathon platforms.

TABLE VII
FEATURE COMPARISON: HACKEVAL VS. EXISTING PLATFORMS

Feature	Devpost	HackerEarth	Unstop	Kaggle	HackEval
AI Evaluation	✗	✗	✗	Partial	✓ Full
Code Analysis	✗	✗	✗	✗	✓ 6-metric
PPT/Slide Analysis	✗	✗	✗	✗	✓ AI-driven
Plagiarism Detection	✗	✗	✗	✗	✓ Semantic
Real-Time Feedback	✗	✗	✗	✗	✓ <10 sec
Multi-Tenant SaaS	✗	Partial	Partial	✗	✓ Full
Judge Override	✗	✗	✗	✗	✓ Hybrid

D. Participant Feedback Survey

A post-event survey administered to 89 participants assessed the perceived utility of AI-generated feedback relative to prior manual feedback experiences. Respondents rated AI-generated feedback 4.1/5.0 for specificity and 4.3/5.0 for actionability, compared with 2.9/5.0 and 3.1/5.0 respectively for prior manual evaluation feedback. Participants specifically identified code-level diagnostic annotations from the GitHub Agent and slide - specific design recommendations from the PPT Agent as the most instrumentally valuable outputs for guiding post- event project iteration.

X. CONCLUSION

This paper has presented HackEval, an intelligent multi- agent framework that fundamentally modernizes the evaluation infrastructure of competitive hackathon ecosystems. The two architecturally central contributions—the PPT Analyzer Agent and the GitHub Code Analyzer Agent—collectively address the evaluation dimensions most consistently mishandled by human judges under time pressure. By integrating multi-stage LLM reasoning [12], CLIP-based contrastive visual assessment [13], and rigorous static analysis toolchains within the unified hybrid scoring model defined in (1), HackEval achieves a Pearson correlation of $r = 0.93$ with expert human consensus evaluations, a 92.8% reduction in per- team evaluation time, and a 77.4% improvement in inter-rater scoring consistency. The multi-tenant SaaS architecture ensures that institutional deployment requires no dedicated infrastructure investment, while concurrent scalability exceeding 500 teams eliminates practical upper bounds on event size. The results support a broader claim: that AI-human collaborative evaluation—in which AI subsystems handle objective, scalable, reproducible assessment while human judges provide contextual override for edge cases—yields outcomes superior to either modality acting independently.

XI. FUTURE WORK

Several directions merit pursuit in subsequent iterations of this framework. A Video Pitch Analyzer module employing multimodal audio-visual LLM reasoning would extend automated evaluation to recorded demonstration presentations, capturing aspects of delivery quality and verbal articulation currently outside the system's scope. Adaptive weight learning via reinforcement learning from historical judge-override patterns would enable the Hybrid Scoring Engine to optimize agent weight configurations for specific event domains, moving beyond the current static defaults in (1). Blockchain score immutability through SHA-256 score hash anchoring to a public distributed ledger would provide tamper-evident result certification— a significant concern for events with substantial prizes or institutional recognition. Federated plagiarism detection [15] would enable cross-institutional similarity checking through federated embedding aggregation protocols that do not require raw submission data to be shared across organizational boundaries. Finally, an explainability dashboard leveraging SHAP-based score decomposition visualizations would allow participants to trace the precise contribution of each evaluated criterion to their final composite score, substantially increasing the framework's pedagogical value.

ACKNOWLEDGMENT

The authors gratefully acknowledge Mr. Preshit Mangesh Desai for his technical mentorship and domain guidance throughout this research. This study was conducted within the Dept. of Computer Engineering & Applications, GLA University, Mathura, India. The authors also thank the participating teams and faculty judges at GLA University's Smart India Hackathon qualifier event for their cooperation during the empirical evaluation study.

REFERENCES

[1] HackerEarth Inc., "Global hackathon trends report 2023," HackerEarth, San Francisco, CA, USA, Tech. Rep., 2023. [Online]. Available: https://www.hackerearth.com/hackathon-report

[2] R. Stemler and J. Tsai, "Measuring inter-rater reliability in competition judging: A meta-analysis," J. Educ. Meas., vol. 41, no. 2, pp. 113–128, 2020, doi: 10.1111/jedm.12254.

[3] Devpost Inc., "Devpost — The home for hackathons," 2024. [Online]. Available: https://devpost.com/

[4] HackerEarth Inc., "HackerEarth hackathons," 2024. [Online].Available: https://www.hackerearth.com/challenges/

[5] Unstop (Dare2Compete), "Unstop competitions platform," 2024. [Online]. Available: https://unstop.com/

[6] Kaggle Inc., "Kaggle competitions," 2024. [Online]. Available: https://www.kaggle.com/competitions/

[7] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in Proc. 10th Koli Calling Int. Conf. Comput. Educ. Res., Koli, Finland, 2010, pp. 86–93.

[8] Y. Liang et al., "Can large language models write good code? An empirical study," in Proc. ACM SIGSOFT Int. Symp. Softw. Testing Anal. (ISSTA), Seattle, WA, USA, 2023.

[9] K. Stasaski and M. Hearst, "Automatically

scoring explanations in science education," in Findings Assoc. Comput. Linguistics: EMNLP 2022, Abu Dhabi, UAE, 2022, pp. 4463–4476.

[10] A. V. Savchenko, "Rapid attributes-based image retrieval using fine-tuned vision-language models," Neural Netw., vol. 169, pp. 222–234, 2024.

[11] C. Mao, Y. Chen, and T. Liu, "Evaluating novelty and originality in generated text using embedding distance metrics," in Proc. AAAI Workshop AI Eval., Washington, DC, USA, 2023.

[12] T. Brown et al., "Language models are few-shot learners," in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 33, pp. 1877– 1901, 2020.

[13] A. Radford et al., "Learning transferable visual models from natural language supervision," in Proc. 38th Int. Conf. Mach. Learn. (ICML), 2021, pp. 8748–8763.

[14] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," in Proc. 2019 Conf. Empirical Methods Natural Language Process. (EMNLP), Hong Kong, China, 2019, pp. 3982–3992.

[15] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," IEEE Trans. Big Data, vol. 7, no. 3, pp. 535–547, 2021.