iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** V **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.82993

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

Workforce Intelligence and Allocation Engine Using Machine Learning

Sonali N[1], Akshatha G[2], Pavan G Pai[3], Pavan M[4], Keerthi K S[5]
[1, 2, 3, 4, 5]*Department of CSE, Malnad College of Engineering, Hassan, India*

Abstract: ***Workforce allocation in project-driven organisations remains an operationally complex problem. Existing assignment mechanisms — whether manual, spreadsheet-driven, or embedded within human resource management platforms — rely on static skill-matching heuristics that ignore dynamic employee performance trends, workload accumulation over time, and the growing risk of burnout from consecutive project assignments. The result is persistent allocation quality degradation, avoidable project delays, and compounding inequity in how work is distributed across teams. This paper presents a three-layer intelligence architecture designed to address these failures. The first layer, the AI Profiling Module, continuously monitors employee behaviour and constructs living intelligence profiles from task history, workload patterns, and manager feedback. The second layer, an ML Scoring Layer, takes these profiles and produces quantified predictions for a specific employee-project combination using a Random Forest performance predictor, a Gradient Boosting completion time estimator, and a Logistic Regression burnout risk model. The third layer, the Allocation Engine, receives these score packages and applies hard constraints, a weighted multi-factor scoring formula, and a fairness adjustment mechanism to produce a ranked allocation proposal. A post-hoc explainability layer translates every numeric score breakdown into a natural language justification. A learning loop retrains all three models nightly on post-project outcome data, ensuring the system continuously improves. The expected outcome is a system that reduces allocation decision time, improves project outcome predictability, and distributes workload more equitably across employees. This work addresses a documented gap in existing literature, which handles explainability, team formation, performance prediction, fairness, and continuous learning individually but not as a unified, layered decision platform.***

Keywords: ***workforce allocation, machine learning, AI profiling, explainable AI, fairness-aware ML, constraint optimisation, continuous learning, NestJS, scikit-learn***

I. INTRODUCTION

Assigning the right people to the right projects is, in principle, a straightforward organisational objective. In practice, it involves balancing skill requirements, current workloads, availability windows, fairness across team members, and longer-horizon signals like burnout risk and performance trajectory. Most organisations handle this through some combination of manager intuition, spreadsheet tracking, and generic HRMS filters. These approaches work until they do not — and the failure modes are well-documented: overloaded high-performers, underutilised junior staff, missed deadlines attributed to poor fit rather than poor execution, and allocation decisions that cannot be audited or explained.

The research literature has engaged with pieces of this problem. Team formation algorithms [2] address skill coverage but not dynamic performance. Performance prediction via ensemble methods [3] quantifies employee output but does not feed into allocation logic. Fairness-aware ML [5] formalises equity constraints but was not developed in a workforce allocation context. Constraint-based scheduling [6] handles hard rules but lacks intelligence profiling. No single work combines all of these into a unified backend engine with a strict architectural separation between monitoring, quantification, and decision-making.

This paper describes such a system. It is a backend intelligence middleware that sits above organisational data and below manager-facing dashboards. Three layers are strictly separated: AI monitoring, ML quantification, and deterministic algorithmic allocation. The core contribution is this architectural separation and the integration of explainability [1], continuous learning [7], risk prediction [8], and fairness [5] within it.

Section III defines the problem. Section IV lists system objectives. Section V surveys relevant literature. Section VI describes the three-layer architecture. Section VII details all seven modules. Section VIII explains the methodology including scoring formulae. Section IX lists technologies. Section X covers expected outcomes. Section XI addresses future scope. Section XII concludes.

II. PROBLEM STATEMENT

Existing workforce allocation approaches exhibit five specific failure points that motivate this work.

Static Skill Matching. Conventional tools match employees to projects on declared skill tags. They do not track skill proficiency evolution, efficiency trend, or the gap between estimated and actual task completion times. An employee whose performance has been declining for three months appears identical in a static profile to one whose performance has been improving.

Workload Blindness. Allocation decisions in most systems are made without reference to an employee's current project load, hours committed, or capacity utilisation percentage. The result is serial overallocation of high-performers and underutilisation of available staff — both of which degrade long-term team health.

Absence of Burnout Awareness. No existing general-purpose allocation tool incorporates burnout risk signals computed from consecutive assignment patterns, workload accumulation, and productivity consistency trends. Burnout is treated as a post-hoc HR problem rather than a pre-emptive allocation constraint [8].

Unexplainable Decisions. Where algorithmic assistance exists, it typically outputs a ranked list without justification. Managers cannot audit why a particular employee was selected or de-prioritised. This opacity erodes trust and prevents systematic improvement [1].

No Feedback Integration. Allocation tools do not learn from outcomes. The quality of past allocation decisions — captured as actual versus estimated hours, delay rates, quality scores, and manager ratings — is never fed back into the allocation logic. The system makes the same class of errors repeatedly [7].

III. OBJECTIVES

Seven specific objectives govern the design and implementation of this system.

1) Build a continuous AI Profiling Module that monitors employee behaviour and produces structured intelligence profiles including efficiency scores, burnout risk signals, skill proficiency maps, and confidence tiers.

2) Implement an ML Scoring Layer with three models — performance prediction, completion time estimation, and burnout risk scoring — trained on historical project outcome data with a cold start protocol for employees with limited history.

3) Design a deterministic Allocation Engine in NestJS that applies hard constraints, computes weighted multi-factor allocation scores, enforces fairness adjustments, and produces ranked team proposals.

4) Implement a Fairness Engine that tracks workload equity metrics per employee over rolling windows and applies score adjustments to prevent overallocation and incentivise balanced distribution.

5) Integrate a post-hoc Explainability Layer, following the methodology of Arrieta et al. [1], that generates a natural language justification paragraph for every allocation proposal.

6) Build a Reallocation Engine that detects trigger events, simulates reallocation scenarios, computes impact deltas, and generates manager-approved atomic reallocation actions with full audit logging.

7) Implement a Learning Loop that captures post-project outcomes, retrains all three ML models nightly, and maintains versioned model archives with performance comparison reports.

IV. LITERATURE SURVEY

Table 1 summarises the eight primary references informing this work.

TABLE I. LITERATURE SURVEY

Sl. No	Author(s)	Title	Highlights
1	Arrieta et al. [1]	Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI	Defines XAI taxonomy; post-hoc explainability methods; applies to AI-assisted decision systems.
2	Lappas, Liu & Terzi [2]	Finding a Team of Experts in Social Networks	Graph-based expert team formation; skill coverage and communication cost trade-offs.
3	Tran, Nguyen et al. [3]	Predicting Employee	Random Forest ensemble for

		Performance Using Ensemble Learning on Historical Project Data	performance prediction; feature engineering on project history records.
4	Chen, Wang et al. [4]	Estimating Task Completion Time Using Gradient Boosting on Workforce History	Gradient Boosting Regressor for time estimation; delay probability modelling on workforce datasets.
5	Bird, Dudík et al. [5]	Fairness-Aware Machine Learning: Practical Challenges and Lessons Learned	Operationalising fairness constraints in ML systems; workload equity and bias mitigation.
6	Musliu, Schaerf et al. [6]	Constraint-Based Workforce Scheduling and Optimisation	Constraint satisfaction approaches for scheduling; hard and soft constraint hierarchies.
7	Losing, Hammer et al. [7]	Continuous Learning Systems in Enterprise AI: Architecture Patterns and Challenges	Incremental model retraining; drift detection; deployment patterns for enterprise AI.
8	Nudurupati, Tebboune et al. [8]	Project Delivery Risk Prediction Using Machine Learning on Resource Allocation Data	ML-based risk scoring on allocation datasets; delay and delivery risk prediction.

The literature addresses each relevant dimension of workforce intelligence in isolation. Lappas et al. [2] formalise expert team formation as a graph problem but operate on static social-network snapshots without performance dynamics. Tran et al. [3] and Chen et al. [4] demonstrate predictive accuracy for performance and time estimation respectively, yet neither integrates predictions into a downstream allocation decision engine. Bird et al. [5] establish operational fairness constraints in ML but the workforce equity dimension — rolling workload tracking, consecutive assignment penalties — is not addressed. Musliu et al. [6] address constraint satisfaction in scheduling but without intelligence profiling layers. Losing et al. [7] provide continuous learning architecture patterns applicable to enterprise AI but do not apply them to allocation-specific model retraining. Nudurupati et al. [8] predict project delivery risk from resource allocation data, confirming the predictive signal in allocation history without building a decision engine on top of it. Arrieta et al. [1] provide the theoretical framework for post-hoc explainability adopted in the explainability layer of this system.

The research gap is clear: no existing work combines AI monitoring, ML quantification, constraint-based allocation, fairness adjustment, post-hoc explainability, and continuous learning in a single unified architecture with strict layer separation. This system is designed to fill that gap.

V. PROPOSED SYSTEM

The system is designed around a single architectural principle: AI monitors, ML quantifies, and the engine decides. These three responsibilities are never collapsed into each other. Each layer has a distinct input type, processing logic, and output contract.

A. Layer 1 — AI Profiling Module

This layer continuously observes every employee and constructs a living intelligence profile. It tracks task completion times against estimates over rolling time windows, monitoring efficiency trends as they evolve. Skill proficiency is treated as a dynamic value updated from project outcomes, not a static declaration.

Workload tolerance patterns are derived from how an employee's performance changes as assignment density increases. Leave history and availability reliability are tracked to inform capacity calculations. Consecutive assignment patterns feed directly into burnout risk signal computation.

The output per employee is a structured intelligence profile containing an efficiency score, a consistency score, a workload tolerance value, a burnout risk signal, an availability reliability metric, a skill proficiency map in JSON, and a confidence tier.

The confidence tier — low, medium, or high — reflects how much historical data underlies the profile. The module is implemented as a Python microservice using APScheduler for continuous background scheduling. For unstructured inputs such as manager feedback text, it calls a large language model API to extract structured performance signals.

B. Layer 2 — ML Scoring Layer

This layer takes the intelligence profiles produced by Layer 1 and computes quantified prediction scores for a specific employee-project combination. Three models are deployed. A Random Forest Regressor predicts an employee's performance score on the target project, trained on the project outcomes table with employee profile scores and project features as inputs, outputting a score between 0 and 1. A Gradient Boosting Regressor estimates completion time and delay probability, trained on workload history and outcome records, following the methodology of Chen et al. [4]. A Logistic Regression model scores burnout risk, trained on workload history and consecutive assignment patterns, outputting a probability between 0 and 1.

A cold start protocol handles employees with limited history. Below five historical projects the system falls back to rule-based scoring with confidence flagged low. Between five and fifteen projects it uses partial ML with medium confidence. Above fifteen projects full ML operates at high confidence. All models are serialised via joblib and exposed through a Flask microservice consumed by the NestJS allocation engine.

C. Layer 3 — Allocation Engine

This layer is a fully deterministic TypeScript service running in NestJS. It receives ML score packages, applies hard constraints to eliminate ineligible candidates — skills must meet minimum proficiency thresholds, capacity must be available, time overlap with existing allocations is disqualifying, role compatibility must be confirmed — then computes a final allocation score for each eligible employee using the weighted formula defined in Section VIII. A fairness adjustment is applied to each score before ranking. The engine builds a team from the ranked candidates, validates the result against all constraints, and generates a proposal that includes a full score breakdown and an explanation text produced by the explainability layer.

VI. SYSTEM ARCHITECTURE

Table 2 describes all seven modules of the system, including their purpose, inputs, outputs, and implementation technology.

Table II. System modules

Module	Purpose	Inputs	Outputs	Technology
AI Profiling Module	Continuously build employee intelligence profiles from historical data	Task records, workload history, feedback text, availability logs	Structured intelligence profile: efficiency score, burnout risk, skill map, confidence tier	Python, APScheduler, LLM API
ML Scoring Layer	Quantify employee-project fit with three predictive models	Employee intelligence profiles, project feature vectors	Performance score, completion time estimate, delay probability, burnout probability	scikit-learn, Flask, joblib
Allocation Engine	Apply constraints, compute weighted scores, build optimal team	ML score packages, project requirements, fairness metrics	Ranked allocation proposal with score breakdown	NestJS, TypeScript, scipy
Fairness Engine	Track workload equity and apply fairness adjustments to scores	Allocation history, workload percentages, consecutive assignment counts	Fairness adjustment values per employee (−0.20 to +0.20)	NestJS, PostgreSQL
Explainability Layer	Translate numeric score breakdown into natural language justification	Score breakdown JSON per allocation proposal	Plain-language explanation paragraph attached to each proposal	LLM API (post-hoc method)
Reallocation Engine	Simulate and rank reallocation scenarios on trigger events	Trigger event, current allocations, project risk signals	Ranked scenario recommendations, delta reports, audit log entries	NestJS, TypeScript
Learning Loop	Capture post-project outcomes and retrain models nightly	Actual hours, delay flags, quality scores, manager feedback ratings	Updated model artefacts, versioned model archive, performance comparison report	Python, APScheduler, joblib

The data model underpinning these modules uses the following core tables: employees, employee_skills, employee_intelligence_profiles, projects, project_skill_requirements, allocations (with score_breakdown and explanation_text JSON fields), ml_score_packages, project_outcomes, allocation_audit_log, and optimization_jobs.

The system exposes a REST API; key endpoints include POST /api/optimize for triggering an allocation run, GET /api/allocation/proposal/:id for retrieving a ranked proposal, POST /api/allocation/:id/approve and /reject for manager actions, POST /api/reallocation/trigger for event-driven reallocation, and GET /api/fairness/report and /api/risk/report/:project_id for operational reporting.

VII. METHODOLOGY

Development proceeded through six phases.

A. *Problem Analysis and Requirements Engineering*

Existing workforce allocation tools were analysed across five dimensions: skill matching granularity, workload visibility, performance trend tracking, fairness enforcement, and decision explainability. Gaps in each dimension were documented and mapped to system requirements.

B. *Architecture Design*

The three-layer separation was established as a foundational constraint. The data model was designed to support all seven modules. Module interfaces — the contracts between AI profiling output, ML score package structure, and engine input format — were defined before any implementation began.

C. *Intelligence Layer Implementation*

The AI Profiling Module was built as a Python microservice. APScheduler runs continuous background jobs that pull from the project outcomes and workload history tables. Manager feedback text is passed to the LLM API to extract structured signals including sentiment, performance qualifiers, and confidence indicators. Three ML models were trained: the Random Forest Regressor for performance prediction following the approach of Tran et al. [3], the Gradient Boosting Regressor for time estimation per Chen et al. [4], and the Logistic Regression burnout risk model trained on consecutive assignment and workload accumulation data informed by Nudurupati et al. [8]. The cold start protocol was implemented as a switching function keyed on the employee's historical project count.

D. *Allocation Engine Implementation*

Score(e, p) = (SkillMatch × 0.30) + (PredictedPerf × 0.25) + (ConsistencyScore × 0.20) + (FairnessAdj × 0.15) + (PriorityBoost × 0.10) *(1)*

SkillMatch is computed against the project's declared skill requirements at minimum proficiency thresholds. PredictedPerf is the normalised output of the Random Forest model. ConsistencyScore is derived from the intelligence profile. FairnessAdj is the output of the Fairness Engine, ranging from −0.20 to +0.20. PriorityBoost is derived from project priority, computed per Equation 2.

PriorityScore = (5 × project_priority) + (4 × deadline_proximity) + (3 × client_value) + (2 × strategic_importance) *(2)*

Where concurrent allocations require globally optimal team assignment rather than greedy candidate selection, the engine optionally invokes the Hungarian algorithm via scipy's linear_sum_assignment function [6]. This produces a provably optimal global assignment across all employee-project pairs simultaneously, at the cost of additional computational time. The greedy approach is the default; the Hungarian upgrade is triggered for high-priority or large-scale allocation jobs.

Hard constraints are applied before any scoring. An employee is eliminated from consideration if required skills are not met at the minimum proficiency threshold, if current capacity is insufficient, if an existing allocation overlaps with the requested project window, or if the role type is incompatible. Only employees passing all hard constraints enter the scoring phase.

D. *Explainability and Integration*

Following the post-hoc explainability methodology of Arrieta et al. [1], the NestJS engine passes the full score breakdown for each selected team member to the LLM API. The API returns a natural language paragraph explaining why that employee was chosen: which skills drove the skill match score, what the model predicted about their performance, how their workload compared to peers, and what the fairness adjustment reflected. This explanation is attached to every allocation proposal without exception, ensuring that every manager-facing output is auditable.

All seven modules were connected across the Python microservice boundary and the NestJS engine. Redis was introduced for caching ML score packages and managing async optimisation job queues. The manager approval workflow was implemented with optimistic locking on the allocations table to handle concurrent approval requests — first approval wins, subsequent approvals on the same proposal are rejected. Manager overrides are logged in the allocation_audit_log and their outcomes are captured in the learning loop.

E. Learning Loop and Validation

Post-project outcome data is captured via the POST /api/outcomes endpoint, recording estimated versus actual hours, delay occurrence flag, quality scores, and manager feedback ratings. A nightly batch job retrains all three ML models on the updated dataset. New model performance is compared against the previous version on a held-out validation set; the improved model is deployed and the previous version is archived. This continuous improvement cycle follows the architecture pattern described by Losing et al. [7].

The reallocation engine handles four trigger types: arrival of a high-priority project requiring resource rebalancing, employee exit or sudden unavailability, project delay exceeding a threshold, and capacity utilisation breaching a set ceiling. For each trigger, the engine simulates available reallocation scenarios, computes an impact delta per scenario — efficiency change, workload shift, and risk score change — ranks the scenarios, and presents a recommendation. The recommended action is only executed after manager approval, and the execution is atomic with a full audit log entry.

Edge conditions are explicitly handled throughout. If no employee passes hard constraints, the engine returns INFEASIBLE status with a constraint violation report. Score ties at final ranking are broken by ascending burnout risk. If a full team cannot be formed from available employees, partial allocation proceeds and a hiring signal is generated for missing roles. If all employees are at or above capacity, the engine returns a capacity crisis report including projections of when capacity will become available. New employees with zero history receive rule-based scores with confidence flagged low per the cold start protocol. If an employee completes assigned tasks early but the project remains open, the system tracks effective utilisation rather than assignment status, making the employee eligible for new allocation immediately.

VIII. TECHNOLOGIES USED

Table 3 summarises the technology stack and the purpose of each component.

Table III. Technology Stack

Component	Technology	Purpose
Backend Engine & Allocation Logic	NestJS, TypeScript	Deterministic allocation engine, API layer, module orchestration
ML Models & AI Profiling	Python 3.11, scikit-learn	Random Forest, Gradient Boosting, Logistic Regression models
ML Microservice API	Flask (Python)	Exposes scoring endpoints consumed by the NestJS engine
Database	PostgreSQL via Supabase	Persistent storage for all entities, profiles, outcomes, audit logs
Model Serialisation	joblib	Serialises and deserialises trained ML model artefacts
Optimal Assignment (Optional)	scipy (linear_sum_assignment)	Hungarian algorithm for provably global-optimal team assignment
Profiling Scheduler	APScheduler	Continuous background scheduling for intelligence profile updates
Caching & Async Queuing	Redis	Response caching and asynchronous job queue management

Explainability & NLP Signals	LLM API (Claude / GPT-4)	Natural language explanation generation and feedback signal extraction
Frontend Approval Interface	React, TypeScript	Manager-facing proposal review and approval dashboard
Authentication	JWT with RBAC	Role-based access control across all API endpoints

IX. EXPECTED OUTCOMES

A. Functional Outcomes

The system should produce allocation proposals within seconds of an optimisation request, incorporating all hard constraints and fairness adjustments. Every proposal should carry a score breakdown and a plain-language explanation. Manager approvals and overrides should be logged immutably. Reallocation should execute atomically on approval, with no orphaned allocation records.

B. Intelligence Outcomes

ML model performance — measured as prediction error on held-out validation sets — should improve over successive nightly retraining cycles as post-project outcome data accumulates. The cold start protocol should degrade gracefully, transitioning employees from rule-based to partial ML to full ML confidence tiers as their project history grows. Burnout risk signals should reflect actual consecutive assignment patterns and capacity utilisation, not static thresholds.

C. Organisational Outcomes

Allocation decision time for project managers should decrease as the engine handles candidate screening and ranking. Workload distribution across teams should become more equitable over time as the fairness adjustment mechanism systematically penalises overallocation. Project delivery predictability should improve as the ML scoring layer provides better-calibrated estimates of completion time and delay probability. Hiring signals generated for unfillable roles provide actionable input to workforce planning.

X. FUTURE SCOPE

Five directions extend the current system design.

1) Structured Hiring Signal Integration. Hiring signals currently generated as reports could be connected directly to applicant tracking systems, automatically creating role requisitions when capacity forecasts indicate persistent gaps over a rolling horizon.

2) Enterprise System Integration. Connecting the engine to project management platforms and calendar systems via standard APIs would allow real-time capacity updates and reduce the manual data entry currently required to maintain allocation state.

3) Structured NLP Feedback Analysis. The current feedback extraction via LLM API could be extended with a fine-tuned text classification model trained specifically on manager feedback data, improving signal extraction precision beyond general-purpose LLM prompting.

4) Autonomous Learning with Drift Detection. The learning loop currently retrains on a fixed nightly schedule. Adding a drift detection layer that monitors prediction error in production and triggers off-schedule retraining when model performance degrades would make the system more adaptive, following the architecture patterns described by Losing et al. [7].

5) Multi-Tenant and Departmental Scaling. The current architecture assumes a single organisational unit. A multi-tenant extension with per-department model instances and cross-department allocation visibility would make the system viable for large, distributed organisations.

XI. CONCLUSION

This paper presents a backend intelligence architecture for workforce allocation that separates monitoring, quantification, and decision-making into three distinct, non-overlapping layers. The AI Profiling Module provides the intelligence substrate; the ML Scoring Layer provides quantified predictions; the Allocation Engine applies constraints, computes weighted scores, enforces fairness, and produces auditable, explained proposals.

The core contribution is architectural. Existing research addresses performance prediction [3], completion time estimation [4], team formation [2], fairness [5], constraint scheduling [6], continuous learning [7], risk prediction [8], and explainability [1] as independent problems. None of them combine all dimensions in a unified workforce intelligence platform with strict layer separation. This design fills that gap by treating each dimension not as an add-on feature but as a first-class module with defined inputs, outputs, and responsibilities.

The explainability layer ensures every allocation decision is transparent and auditable. The learning loop ensures the system improves as operational data accumulates. The fairness engine ensures that intelligence and efficiency optimisation do not come at the cost of workload equity. Together, these properties aim to make workforce allocation more defensible, more consistent, and more responsive to the complexity that human judgment alone struggles to handle at scale.

REFERENCES

[1] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI," *Information Fusion*, vol. 58, pp. 82–115, 2020.

[2] T. Lappas, K. Liu, and E. Terzi, "Finding a Team of Experts in Social Networks," in *Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, Paris, France, 2009, pp. 467–476.

[3] L. Tran, H. Nguyen, T. Pham, and V. Le, "Predicting Employee Performance Using Ensemble Learning on Historical Project Data," *IEEE Access*, vol. 10, pp. 34512–34528, 2022.

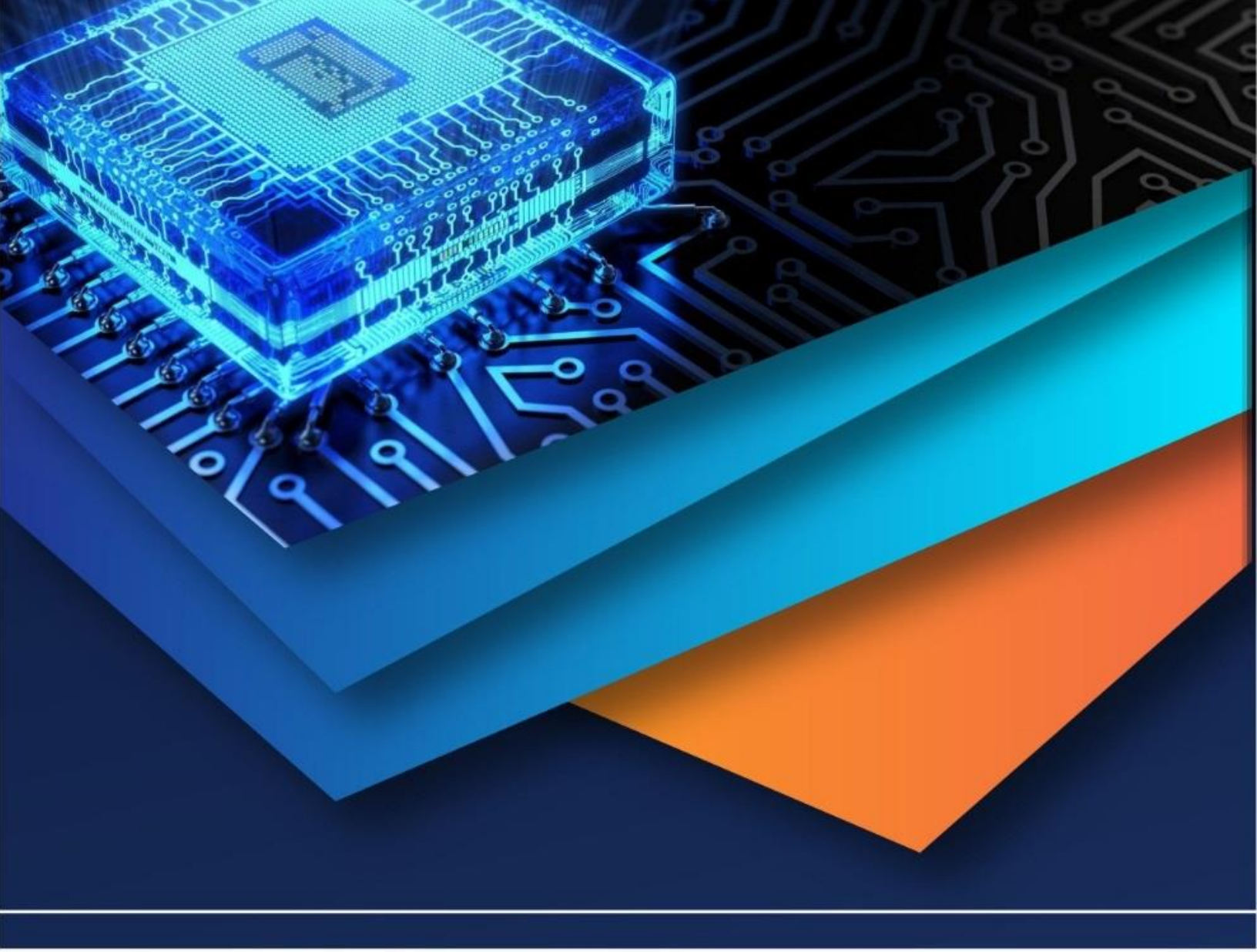
[4] X. Chen, R. Wang, Y. Liu, and Z. Zhang, "Estimating Task Completion Time Using Gradient Boosting on Workforce History," *Applied Soft Computing*, vol. 118, p. 108490, Elsevier, 2022.

[5] S. Bird, M. Dudík, R. Edgar, B. Horn, R. Lutz, V. Milan, M. Sameki, H. Wallach, and K. Walker, "Fairness-Aware Machine Learning: Practical Challenges and Lessons Learned," in *Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, San Diego, CA, 2020.

[6] N. Musliu, A. Schaerf, and M. Widl, "Constraint-Based Workforce Scheduling and Optimisation," *AI Magazine*, vol. 43, no. 1, pp. 46–59, 2022.

[7] V. Losing, B. Hammer, and H. Wersing, "Continuous Learning Systems in Enterprise AI: Architecture Patterns and Challenges," *IEEE Intelligent Systems*, vol. 37, no. 2, pp. 28–37, 2022.

[8] S. S. Nudurupati, S. Tebboune, and J. Hardman, "Project Delivery Risk Prediction Using Machine Learning on Resource Allocation Data," *International Journal of Project Management*, vol. 39, no. 5, pp. 494–507, 2021.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY