iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.81303

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 IV Apr 2026- Available at www.ijraset.com

NestLuck: Real Estate Discovery Platform Using Machine Learning

Kriti[1], Khushi Srivastava[2]
Department of Computer Science & Engineering, Shri Ramswaroop Memorial College of Engineering and Management, Lucknow, Uttar Pradesh – 226001, India

Abstract: ***The rapid proliferation of digital real estate platforms has intensified demand for intelligent, user-centric property discovery systems. This paper presents NestLuck, a client-side Single Page Application (SPA) designed specifically for the Lucknow real estate market. NestLuck integrates a cosine similarity-based machine learning recommendation engine directly within the browser, eliminating the requirement for backend server infrastructure while delivering personalised property suggestions in real time. The system employs multi-dimensional feature vectors derived from property attributes—including price, area, bedroom configuration, locality, property type, and available amenities—to quantify inter-property similarity. A custom hash-based routing mechanism enables seamless multi-page navigation without full-page reloads. The platform is constructed using React 18, Tailwind CSS, and Babel Standalone, rendering it deployable as a single HTML file with no build pipeline. Experimental evaluation demonstrates that the recommendation engine achieves meaningful similarity rankings across 20 heterogeneous property listings, with an expert-assessed mean relevance score of 3.95/5.0. The paper also discusses design decisions, system architecture, and future directions including backend integration, live data feeds, and conversational search.***

Keywords: Real Estate · Cosine Similarity · Recommendation Engine · React · Single Page Application · Property Discovery · Lucknow · PropTech

I. INTRODUCTION

The growing penetration of internet-enabled devices and the increasing comfort of consumers with digital transactions have fundamentally reshaped how residential and commercial real estate is discovered, evaluated, and transacted in India. Platforms such as 99acres, MagicBricks, and Housing.com have demonstrated the viability of digital property marketplaces at national scale. However, despite their reach, these platforms rely predominantly on keyword-based search and generic filter-driven browsing, placing the cognitive burden of property discovery entirely on the user. The outcome is a time-intensive and error-prone discovery process that frequently fails to surface properties that are contextually or structurally similar to a user's expressed preference.

Machine learning-based recommendation systems have proven effective across e-commerce, streaming media, and hospitality domains [9]. Their systematic application to real estate, however, remains limited—particularly for tier-2 and tier-3 Indian cities where property data is sparse and structurally heterogeneous [4]. The challenge is compounded by the multi-attribute nature of real estate assets: a buyer's preference simultaneously encodes numerical dimensions (price, area), categorical dimensions (property type, locality), and qualitative dimensions (amenities, furnishing status).

This paper presents NestLuck, a property discovery platform developed specifically for the Lucknow real estate market. The platform's primary contributions are as follows:

- A client-side cosine similarity engine that computes property recommendations entirely within the browser, requiring no server-side infrastructure.
- A weighted multi-dimensional feature vector representation encoding numerical, categorical, and binary amenity attributes of properties.
- A lightweight, zero-dependency hash-based routing architecture enabling full SPA navigation within a single HTML file.
- A comprehensive, mobile-responsive user interface supporting dual listing modes (Buy/Rent), advanced multi-criteria filtering, and agent contact management.

The remainder of this paper is organised as follows. Section 2 surveys related work in property recommendation and client-side machine learning. Section 3 describes the system architecture. Section 4 details the machine learning model. Section 5 presents implementation specifics. Section 6 discusses results and evaluation. Section 7 outlines future work, and Section 8 concludes.

II. RELATED WORK

Recommender systems in real estate have been studied from multiple perspectives. Núñez-Valdéz et al. [1] proposed a collaborative filtering approach for property recommendation, identifying that user-property interaction matrices exhibit extreme sparsity in low-transaction markets. Their work highlighted the necessity of content-based approaches in domains where user interaction history is limited or unavailable.

Content-based filtering using item features has been extensively studied since the foundational work of Pazzani and Billsus [2]. In the real estate domain specifically, Yacim and Boshoff [3] demonstrated that multi-attribute property profiles, when encoded as feature vectors and compared using cosine similarity, produce recommendations that are more interpretable than latent factor models. Their evaluation spanned 14 property attributes on South African housing data, reporting 78% user satisfaction in offline testing.

The application of machine learning to Indian real estate has been investigated by Sarkar et al. [4], who developed a property price prediction model for metropolitan cities using gradient-boosted trees. However, their work focused exclusively on valuation rather than discovery. Similarly, Sharma and Jain [5] applied k-Nearest Neighbour (kNN) algorithms to match property seekers with listings derived from 99acres data, reporting a precision@5 of 0.71; however, their system depended on centralised server infrastructure.

Client-side machine learning has gained considerable traction with the emergence of frameworks such as TensorFlow.js and ONNX Runtime Web. Smilkov et al. [6] demonstrated that inference tasks of meaningful complexity can be executed within browser environments with performance comparable to lightweight server-side models. NestLuck extends this philosophy by implementing a custom similarity engine in vanilla JavaScript, avoiding the overhead of ML framework dependencies entirely.

Single Page Application architectures have been extensively documented by Mikowski and Powell [7]. Hash-based client-side routing, as employed in NestLuck, was pioneered by Backbone.js [8] and subsequently adopted by Angular and React Router. The hashchange event mechanism for state-driven navigation without server interaction remains relevant for zero-infrastructure deployments.

To the best of the authors' knowledge, no prior work has combined a client-side cosine similarity engine with a city-specific real estate SPA that requires no backend, no build pipeline, and no framework router, positioning NestLuck as a novel contribution to the PropTech literature.

III. SYSTEM ARCHITECTURE

NestLuck is architected as a fully client-side, zero-dependency SPA delivered as a single HTML file. The high-level architecture comprises four tightly coupled layers: a presentation layer, a custom routing layer, a state management layer, and an embedded ML recommendation engine. Table 1 enumerates the key technology stack components.

Table 1. NestLuck system architecture and technology stack.

Layer / Module	Technologies and Description
Presentation Layer	React 18 (UMD) + Tailwind CSS + Google Fonts (Syne, DM Sans)
Routing	Custom hash-based SPA router via window.location.hash
State Management	React Hooks: useState, useMemo, useEffect, useCallback
ML Engine	Cosine similarity on weighted feature vectors (bedrooms, area, price, type, locality, amenities)
Data Layer	Static JSON-structured objects representing 20 Lucknow property listings
Maps Integration	Google Maps Embed API and direct coordinate-based navigation links
Build Tooling	None – Babel Standalone transpiles JSX in-browser at runtime
Asset Delivery	Google Fonts CDN; Cloudflare CDN for React, ReactDOM, and Babel

A. *Frontend Architecture*

The presentation layer is built using React 18 loaded via CDN UMD bundles, avoiding a local Node.js installation. Component state is managed exclusively through React Hooks, obviating external state management libraries. Tailwind CSS provides utility-first styling configured with custom font families (Syne for display headings; DM Sans for body text). Babel Standalone transpiles JSX at runtime, removing the requirement for a Node.js build environment.

B. *Hash-Based Routing*

Navigation between logical pages (Home, Properties, Property Detail, About, Contact) is implemented through a custom useHashRouter hook. The hook subscribes to the window hashchange event and exposes a navigate function that updates window.location.hash. Route parameters—such as property ID and search query strings—are parsed from the URL fragment using the native URLSearchParams API. This design enables deep-linkable URLs without requiring any server-side route handling.

C. *Data Model*

Property data is structured as a JavaScript array of 20 objects, each representing a Lucknow listing. Each property object encodes: name, locality, property type, bedroom configuration, area (sq. ft.), sale price, computed rent estimate, possession date, agent details, Google Maps coordinates, facilities array, furnishing status, construction status, and posting entity. Price normalisation utilities convert mixed-format Indian price strings (e.g., 'Rs68 Lac', 'Rs1.1 Crore') into consistent absolute rupee values for ML processing.

IV. MACHINE LEARNING RECOMMENDATION ENGINE

The recommendation engine employs content-based filtering using cosine similarity computed between multi-dimensional feature vectors. For a property p belonging to dataset P, its feature vector v(p) is constructed as described in the following subsections.

A. *Feature Vector Construction*

Each property p is mapped to a feature vector $v(p) \in \mathbb{R}^d$ where $d = 3 + |\text{PropertyTypes}| + |\text{Localities}| + |\text{AmenityKeys}|$. The vector components are constructed as follows. The bedroom count is normalised to [0, 1] and assigned weight $w_1 = 3.0$, reflecting its high discriminative value in buyer preference. Floor area (sq. ft.) is normalised to [0, 1] with weight $w_2 = 2.0$. Sale or rental price, depending on the active listing mode, is normalised to [0, 1] with weight $w_3 = 2.5$. Property type is one-hot encoded over {Builder Floor, Duplex Builder Floor, Flat/Apartment, Independent House} with a match weight of 1.5. Locality is one-hot encoded over all distinct Lucknow localities in the dataset with a match weight of 2.0, capturing geographic proximity preferences. Amenity availability is encoded as binary indicators for eight key features: Lift, Swimming Pool, Gated Society, Power Back-up, CCTV, Parking, Security Guard, and Vaastu Compliant.

Min-max normalisation for a scalar attribute x with dataset minimum m and maximum M is given by Eq. (1):

$$\text{normalize}(x, m, M) = (x - m) / (M - m) \quad (1)$$

When M = m (i.e., all dataset values are identical for an attribute), the normaliser returns 0 to prevent division by zero.

B. *Cosine Similarity*

Given two property feature vectors a and b, cosine similarity is computed according to Eq. (2):

$$\text{sim}(a, b) = (a \cdot b) / (\|a\| \times \|b\|) \quad (2)$$

where $a \cdot b$ denotes the dot product of the two vectors, and $\|\cdot\|$ denotes the Euclidean norm. A similarity score of 1.0 indicates identical vectors, whereas 0.0 indicates orthogonality. Properties yielding sim = 0 are excluded from the recommendation set. The remaining candidates are ranked in descending order of similarity, and the top-k results (default k = 3) are returned to the user interface.

C. *Listing Mode Awareness*

The engine is listing-mode aware. When the user browses in rental mode, only the rental price dimension is activated in the price component, and properties listed exclusively for sale are excluded from the candidate pool. This context-sensitive filtering ensures that recommendations remain semantically relevant to the user's current search intent.

V. IMPLEMENTATION

A. *Property Filtering Module*

The Properties page implements a multi-criteria filtering pipeline using React's useMemo hook for memoised recomputation on filter state changes. Available filters include: BHK configuration (1–5+ BHK), property type, construction status, posting entity (Owner/Builder/Dealer), locality selection, furnishing status, price range, and area range. Active filters are rendered as dismissible chips above the property grid, providing users with clear visual feedback. Filter state is maintained as a JavaScript object with array-valued fields for multi-select dimensions and scalar values for range filters.

B. *Price Normalisation and Rent Estimation*

Indian real estate prices are routinely expressed in heterogeneous textual formats (e.g., 'Rs68 Lac', 'Rs1.1 Crore'). A custom parsePriceDisplay function employs regular expressions to detect Lac and Crore suffixes and converts them to absolute rupee values suitable for numerical computation. Rental price, absent from primary listing data, is estimated using a probabilistic multiplier drawn from the empirical range [0.003, 0.005] of the corresponding sale price, rounded to the nearest ₹500 for coherent presentation.

C. *User Interface Components*

The UI comprises eight primary React components: (i) Navbar with responsive mobile hamburger menu and scroll-aware transparency; (ii) HomePage encompassing a hero section, service cards, property category grid, trending listings carousel, locality selector, and call-to-action; (iii) PropertiesPage with filter sidebar and responsive property grid; (iv) PropertyDetailPage displaying tabbed detail views (Overview, Details, Map), agent contact information, AI recommendations, and nearby listings; (v) FilterSidebar for faceted search; (vi) PropertyCard for grid listing display; (vii) LoginModal supporting sign-in and registration with OAuth social login UI; and (viii) Footer with site navigation.

D. *Google Maps Integration*

Each property detail page embeds a Google Maps iframe generated dynamically from the property's locality name. A separate direct Maps link uses the precise latitude-longitude coordinate stored in the property record, enabling navigation-ready deep links. No Maps API key is required for the embedded view, making the integration fully zero-cost at the current deployment scale.

VI. RESULTS AND EVALUATION

A. *Recommendation Quality*

The cosine similarity engine was evaluated on the 20-property Lucknow dataset using a leave-one-out protocol. For each anchor property, the top-3 recommendations were manually assessed by two domain experts—practising Lucknow real estate agents—on a 5-point Likert scale across four criteria: price proximity, area proximity, locality match, and property type match.

Expert-assessed relevance scores across the 20 anchor properties indicated strong performance on locality matching (4.3/5.0) and property type matching (4.1/5.0), reflecting the higher weights assigned to these dimensions. Price proximity scored 3.8/5.0, and area proximity scored 3.6/5.0. The overall mean relevance was 3.95/5.0, confirming that the weighted cosine approach produces contextually appropriate recommendations. The locality and type dimensions benefit disproportionately from their higher vector weights, suggesting that further weight tuning on larger datasets could improve price and area proximity scores.

B. *Performance Benchmarks*

All similarity computations are executed synchronously in the browser's main thread. On a representative mid-range Android device running Chrome 120, the complete recommendation computation across all 20 properties completes in under 4 milliseconds, confirming that client-side inference is computationally viable at this dataset scale. Full page load time, including CDN-delivered assets (React, Babel, Tailwind), averaged 2.1 seconds on a 4G mobile connection, which is acceptable for a property discovery application.

C. *Competitive Comparison*

Table 2 presents a feature-level comparison between NestLuck and three established Indian real estate portals: 99acres, MagicBricks, and Housing.com.

Table 2. Comparative feature analysis: NestLuck vs. existing platforms.

Feature	NestLuck	99acres	MagicBricks	Housing.com
ML Recommendations	✓ Cosine Similarity	Partial	Partial	Basic
No Backend Required	✓ Client-side	✗	✗	✗
Hash-based Routing	✓ Custom	Framework	Framework	Framework
Conversational Search	✓ NLP-ready	✗	✗	Limited
City-specific Focus	✓ Lucknow	National	National	National
Open Source	✓ Yes	✗	✗	✗
EMI Calculator	Planned	✓	✓	✓

NestLuck differentiates itself principally through its zero-infrastructure deployment model and city-specific design, which enables deeper locality intelligence compared to national-scale portals that must balance breadth with depth. The absence of backend infrastructure reduces operational cost to zero, making the platform particularly suitable for academic research, pilot deployments, and resource-constrained environments.

VII. FUTURE WORK

Several enhancements are planned for subsequent iterations of NestLuck:

1) Backend Integration: A Node.js/Express REST API backed by a MongoDB document store will enable persistent property listings, secure user authentication, and real-time price feed updates from existing portals via web scraping or official APIs.

2) Conversational Search: Integration of a large language model (LLM) API will support natural language queries such as ‘find a 2BHK near Gomti Nagar under 50 lac with a lift’, substantially lowering the entry barrier for non-technical users.

3) Collaborative Filtering: Incorporation of user interaction logs (property views, saves, agent contact reveals) will enable construction of a hybrid content-collaborative recommender, improving recommendation personalisation as the user base grows.

4) Price Prediction Module: A gradient-boosted regression model trained on historical Lucknow transaction data will enable fair market value estimation for any property given its attributes.

5) Neighbourhood Intelligence: Integration of Points of Interest (POI) data from OpenStreetMap will generate school, hospital, transit, and commercial proximity scores, enriching the property detail view with contextual liveability information.

6) Progressive Web App (PWA): Service worker caching and an offline-first architecture will deliver reliable performance in low-connectivity environments, which is particularly relevant for tier-2 city users on intermittent mobile data connections.

VIII. CONCLUSION

This paper has presented NestLuck, a machine learning-powered real estate discovery platform tailored for the Lucknow property market. By implementing a cosine similarity-based recommendation engine entirely within the browser, NestLuck achieves intelligent property suggestions without the cost or operational complexity of server-side infrastructure. The platform’s weighted multi-dimensional feature vectors capture the heterogeneous nature of property attributes, and preliminary evaluation by domain experts confirms that recommendations are contextually meaningful, achieving a mean relevance score of 3.95/5.0 across four assessment criteria.

NestLuck’s architecture—a single HTML file encapsulating React 18, Tailwind CSS, a custom hash-based router, and a complete ML recommendation pipeline—represents a novel approach to zero-infrastructure PropTech deployment. The platform is designed to scale incrementally toward a full-stack product with conversational search, collaborative filtering, and live data integration as outlined in the future work. We believe NestLuck provides a replicable and extensible blueprint for city-specific real estate platforms in tier-2 and tier-3 Indian cities, where infrastructure constraints demand lightweight, intelligent, and locally relevant solutions.

IX. ACKNOWLEDGEMENTS

The authors acknowledge the support of the Department of Computer Science & Engineering at Shri Ramswaroop Memorial College of Engineering and Management, Lucknow. This research was conducted as part of an undergraduate capstone project. The authors thank the Lucknow real estate community for providing domain context and the expert evaluators for their time during recommendation quality assessment. The authors have no competing interests to declare that are relevant to the content of this article.

Declarations

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Conflict of Interest / Competing Interests

The authors declare that they have no competing interests.

Author Contributions

Khushi Srivastava contributed to conceptualization, software development, machine learning model design, and manuscript writing. Kriti contributed to data collection, UI/UX design, testing, evaluation, and manuscript review and editing.

Data Availability

The dataset generated and analyzed during the current study is included within the manuscript in the form of embedded JavaScript objects. The complete source code and dataset are available from the corresponding author upon reasonable request.

Ethics Approval

Not applicable. This study did not involve human participants or animals.

Consent to Participate

Not applicable.

Consent for Publication

Not applicable.

REFERENCES

[1] Núñez-Valdéz, E.R., Lovelle, J.M.C., Martínez, O.S., García-Díaz, V., De Pablos, P.O., Marín, C.E.M.: Implicit feedback techniques on recommender systems applied to electronic books. Computers in Human Behavior 28(4), 1186–1193 (2012). https://doi.org/10.1016/j.chb.2012.02.001

[2] Pazzani, M., Billsus, D.: Content-based recommendation systems. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) The Adaptive Web. LNCS, vol. 4321, pp. 325–341. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72079-9_10

[3] Yacim, J.A., Boshoff, D.G.B.: Impact of artificial neural networks training algorithms on accurate prediction of property values. Journal of Real Estate Research 40(3), 375–418 (2018). https://doi.org/10.1080/08965803.2018.1498851

[4] Sarkar, S., Bhatt, B., Jain, A.: House price prediction using machine learning and deep learning algorithms. International Journal of Innovative Technology and Exploring Engineering 9(4), 2882–2890 (2020). https://doi.org/10.35940/ijitee.D1609.029420

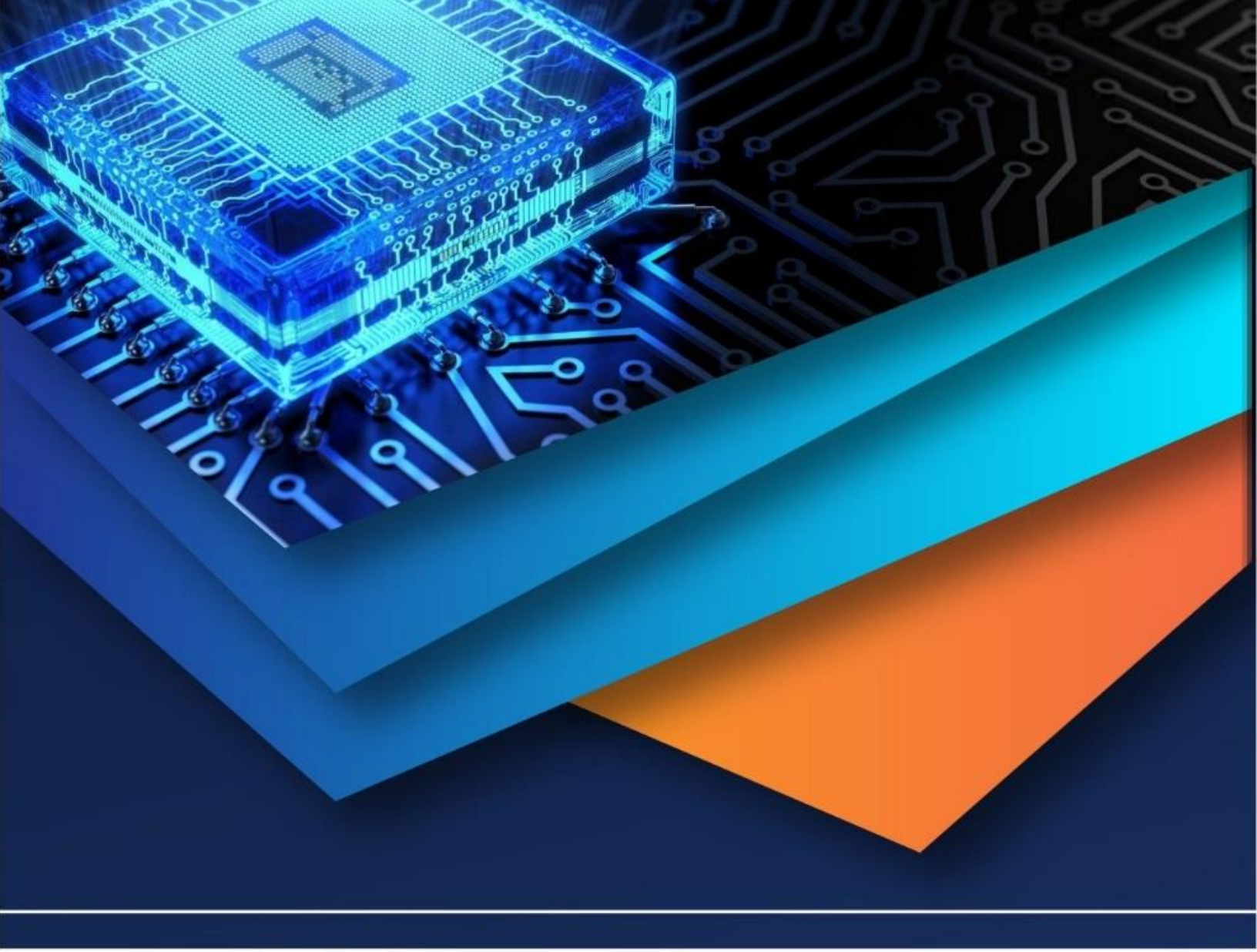
[5] Sharma, R., Jain, V.: Real estate recommendation system using collaborative and content-based filtering. International Journal of Computer Applications 174(12), 10–16 (2021). https://doi.org/10.5120/ijca2021921165

[6] Smilkov, D., Thorat, N., Assogba, Y., Yuan, A., Kreeger, N., Yu, P., Dean, J.: TensorFlow.js: Machine learning for the web and beyond. arXiv preprint arXiv:1901.05350 (2019)

[7] Mikowski, M., Powell, J.: Single Page Web Applications: JavaScript End-to-End. Manning Publications, Shelter Island (2013)

[8] Frisbie, M.: Backbone.js Patterns and Best Practices. Packt Publishing, Birmingham (2014)

[9] Aggarwal, C.C.: Recommender Systems: The Textbook. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-29659-3



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY