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.2026.81997

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

ZeroClick: Natural Language-Driven Autonomous UI Controller

Pranay Reddy[1], Dr. K. Seshacharyulu[2], Akuldeep Jakkula[3], Dheeraj Koka[4], Dhruv Nuti[5]
CSE, KMIT Hyderabad

ABSTRACT: ***Modern web applications built on dynamic frameworks such as React and Next.js present significant challenges for traditional UI automation tools, which rely on brittle, selector-based approaches that frequently fail under interface changes. This paper introduces ZeroClick, a natural language-driven autonomous UI controller that bridges human intent and browser execution through an agentic architecture.***

ZeroClick integrates a Tauri-based desktop interface, a FastAPI/LangGraph backend, and a Model Context Protocol (MCP)-enabled Playwright execution layer to enable end-to-end automation via high-level user prompts. At its core, the system employs a structured execution cycle termed the "Golden Loop", which enforces iterative navigation, DOM state synchronization, action execution using ephemeral element references, and state invalidation to ensure robustness in dynamic environments.

Unlike conventional automation frameworks, ZeroClick incorporates context-aware reasoning through a ReAct-based agent, enabling adaptive interaction with complex single-page applications while mitigating issues such as stale references and inconsistent state propagation. Experimental evaluation across multi-step workflows—including form completion and dynamic navigation tasks—demonstrates improved reliability and reduced execution latency compared to traditional approaches.

The results highlight the potential of combining large language models with structured browser control protocols to achieve resilient, self-healing UI automation. ZeroClick establishes a scalable foundation for next-generation intelligent assistants capable of performing complex web interactions directly from natural language instructions.

I. INTRODUCTION

In the modern digital landscape, interacting with data-heavy web applications often involves repetitive manual friction, including extensive clicking, typing, and navigating through deeply nested menus. As web ecosystems become increasingly complex, the demand for efficient interaction, automated testing, and seamless workflow navigation has grown significantly. Traditional healthcare and enterprise systems alike rely on manual input, which limits proactive efficiency and individualized user experiences.

Recent advancements in Large Language Models (LLMs) and generative artificial intelligence have enabled a paradigm shift toward "Agentic AI," where systems can reason about high-level tasks rather than executing rigid, pre-defined scripts. At the same time, modern web technologies such as React and Next.js allow these systems to be delivered through highly interactive platforms. However, the full potential of these advancements is often hindered by the limitations of traditional automation frameworks.

Despite the ubiquity of existing solutions, traditional UI automation tools like Selenium and Playwright suffer from several critical challenges. They remain heavily dependent on brittle selectors—such as static CSS IDs or XPaths—which frequently break when a user interface undergoes minor layout shifts or utilizes dynamic class-name hashing. Furthermore, standard automation techniques often fail to trigger internal application states in modern Single Page Applications (SPAs), leading to unreliable "hollow" interactions where data is not correctly captured by the underlying framework.

To overcome these limitations, this project introduces ZeroClick, an AI-driven, natural language-driven autonomous UI controller. ZeroClick acts as a sophisticated bridge between human intent and technical execution by unifying a Tauri-based desktop application, a FastAPI/LangGraph backend, and a Model Context Protocol (MCP) server for browser orchestration. By combining intelligent DOM observation with multi-agent reasoning, the platform empowers users to execute complex workflows simply by describing their goals in plain English, representing a significant step toward a "Zero Click" autonomous web experience.

II. RELATED PREVIOUSLY DONE WORK

Research in the field of user interface (UI) automation has traditionally been dominated by script-based testing frameworks designed primarily for quality assurance rather than autonomous interaction.

Evolution of UI Automation Tools

1) Selenium and Legacy Frameworks: Selenium established the early standard for web automation by relying on the WebDriver protocol to interface with browsers. It utilizes specific identifiers such as XPaths and CSS IDs to target elements. While effective for static pages, this approach is fundamentally "brittle" in modern development environments. The rise of utility-first CSS frameworks like Tailwind has led to dynamically generated hashes for class names, causing minor layout shifts to break scripts and creating a significant "maintenance bottleneck".
2) Modern Headless Browsers (Playwright & Puppeteer): Second-generation tools like Playwright introduced improvements such as auto-waiting mechanisms and faster execution via the Chrome DevTools Protocol. Despite these advancements, they remain imperative tools where developers must explicitly program every step of an interaction. They lack a semantic understanding of the UI and cannot adapt to DOM changes without manual intervention.

Large Language Models in Human-Computer Interaction

The advent of Large Language Models (LLMs) has catalyzed a shift toward "Agentic AI," where systems reason about tasks instead of following pre-defined scripts.

3) General Purpose Web Agents: Early autonomous agents like AutoGPT attempt to browse the open web by parsing raw HTML or using vision-based models. However, these systems often suffer from "hallucinations," misinterpreting the state of the application or attempting to interact with non-existent elements.
4) Context-Aware Embedded Agents: ZeroClick represents the evolution of this domain by embedding a context-aware agent directly into the application runtime.

Challenges in Single Page Application (SPA) Automation

Modern web frameworks introduce complexities that traditional tools struggle to address:

5) Virtual DOM and Synthetic Events: React utilizes a synthetic event system that wraps native browser events. Standard techniques, such as setting an input's value property via JavaScript, often fail because React's internal state hooks (e.g., useState) rely on specific event sequences to recognize changes.
6) Component Library Complexity: Frameworks like Material-UI or Ant Design use complex DOM structures with hidden inputs and portals that obscure interactive elements. ZeroClick addresses this through "React Component Recognizers" that identify and interact with high-order components correctly.

Comparative Analysis of Approaches

Feature	Traditional Automation (Selenium/Playwright)	ZeroClick Agentic Controller	Improvement Factor
Interaction Paradigm	Imperative Scripting (Code-based)	Natural Language (Intent-based)	Democratizes access for non-technical users.
Resilience	Brittle; breaks on UI changes	Self-healing; adapts to structure changes	Significantly reduces maintenance overhead.
React Compatibility	Low; struggles with synthetic events	High; native setters & bubbling events	Ensures reliable application state updates.
Context Awareness	None; blind execution	Semantic understanding of DOM	Enables complex reasoning and planning.

Setup Complexity	High; requires dev environment	Low; drop-in application integration	Enables rapid, plug-and-play deployment.

III. MATERIALS AND METHODS

The development of ZeroClick follows a structured methodology combining browser automation, multi-agent orchestration, and modern desktop technologies.

Data Collection and Processing The system captures high-level user goals through a natural language chat interface.

Simultaneously, it collects real-time data from the browser, including scale-to-fit base64 screenshots and structured DOM snapshots that assign unique Reference IDs to interactive elements.

Model Development The reasoning engine utilizes a LangGraph-based ReAct agent driven by models such as gpt-5-mini. A custom SequentialChatOpenAI wrapper was developed to disable parallel tool calling, forcing the agent to synchronize with the DOM state before every atomic interaction.

System Architecture Design

A modular monorepo architecture was designed to separate the execution layers:

- Desktop App: Manages user interaction and WebSocket connections.
- Backend: Orchestrates the agentic reasoning loop and data persistence.
- MCP Server: Provides standardized browser tools via the Model Context Protocol.
- Database: Stores multi-turn session history locally using aiosqlite.

Frontend Development The desktop application is implemented using Tauri v2 (Rust/WebView) and React 19. This ensures a lightweight footprint while supporting real-time streaming of the agent's thought process and live browser screenshots via WebSockets.

Backend Development The backend is a FastAPI application running on port 3001. It handles the coordination of the LangGraph agent, manages local SQLite tables for session events, and interfaces with the Playwright MCP server through standard I/O.

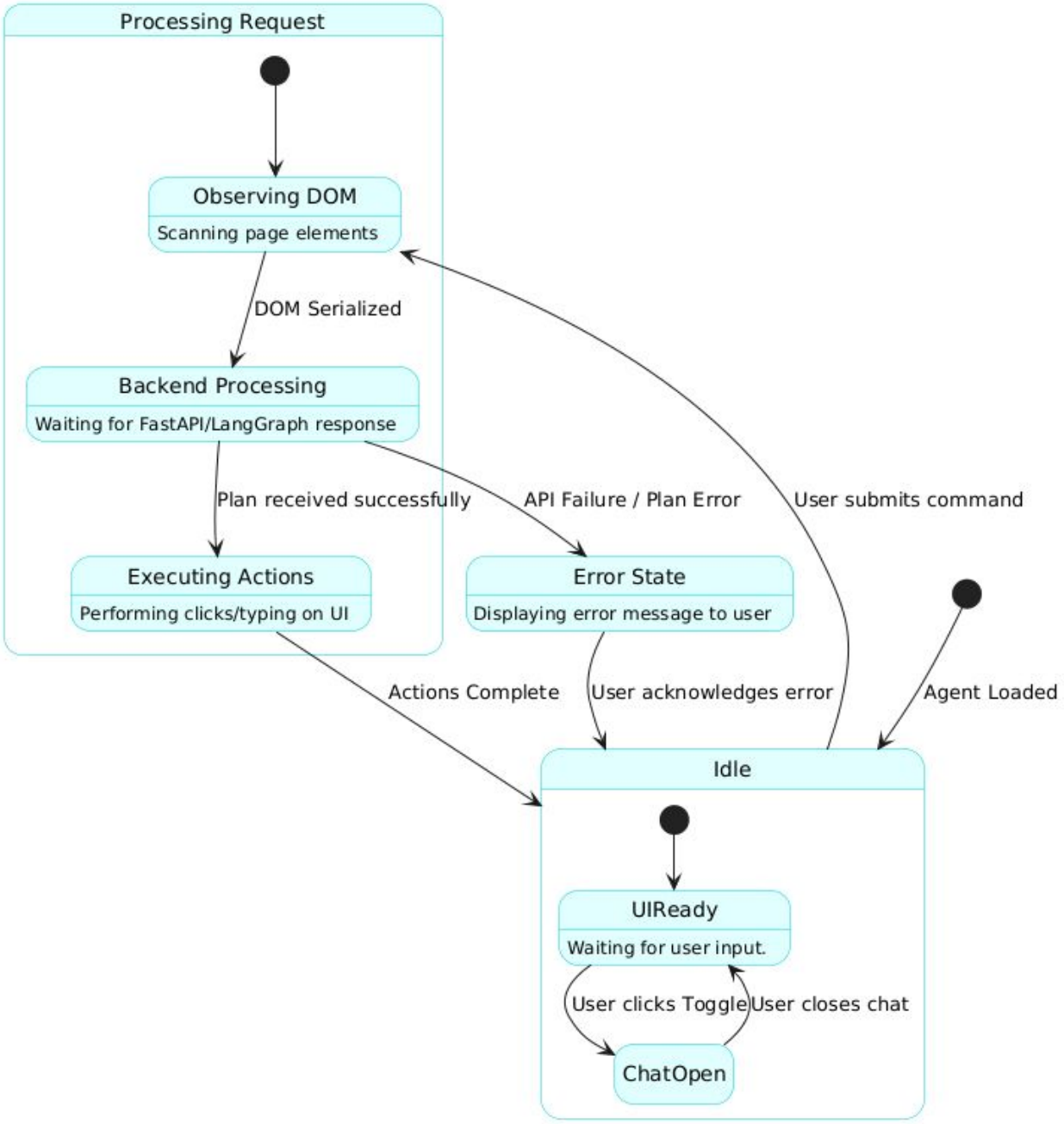
AI Integration AI models are integrated through the Model Context Protocol to execute the "Golden Loop". This loop mandates that the agent must navigate to a URL, sync the DOM state via a snapshot, act on a specific Reference ID, and then invalidate the current state to ensure reliability on dynamic sites.

Testing and Evaluation The system is evaluated through complex end-to-end workflows, such as finding the cheapest travel options or executing multi-page form submissions. Testing focuses on intelligent error recovery, verifying the agent’s ability to re-sync the DOM after "Target closed" or "Ref not found" errors.

IV. IMPLEMENTATION

ZeroClick is implemented as a high-performance, distributed monorepo that synchronizes three distinct technology stacks to achieve autonomous browser control. The system is built to bridge the gap between human intent and machine execution by utilizing a Tauri-based desktop application, a FastAPI backend, and a Model Context Protocol (MCP) server for Playwright.

The core implementation focuses on ensuring resilience in dynamic web environments, where standard automation typically fails. By using a ReAct (Reason + Act) agent orchestrated through LangGraph, the system can dynamically evaluate page states, generate interaction plans, and recover from errors such as element shifts or popups without manual intervention.



Flowchart Explanation

The workflow of the ZeroClick system follows a structured, cyclic process known as the **"Golden Loop"** to ensure reliability:

1) Start and Goal Definition: The process begins when the user inputs a high-level natural language goal into the Tauri desktop interface.

2) WebSocket Initialization: The frontend establishes a real-time connection with the FastAPI backend to stream the agent's thought process and live screenshots.

3) Navigate (Stage 1): The agent identifies the starting URL and commands the MCP Playwright server to navigate to the target web page.

4) Sync State (Stage 2): The system takes a DOM snapshot of the current page. The MCP server assigns unique, ephemeral Reference IDs to all interactive elements, providing the agent with a grounded "map" of the UI.

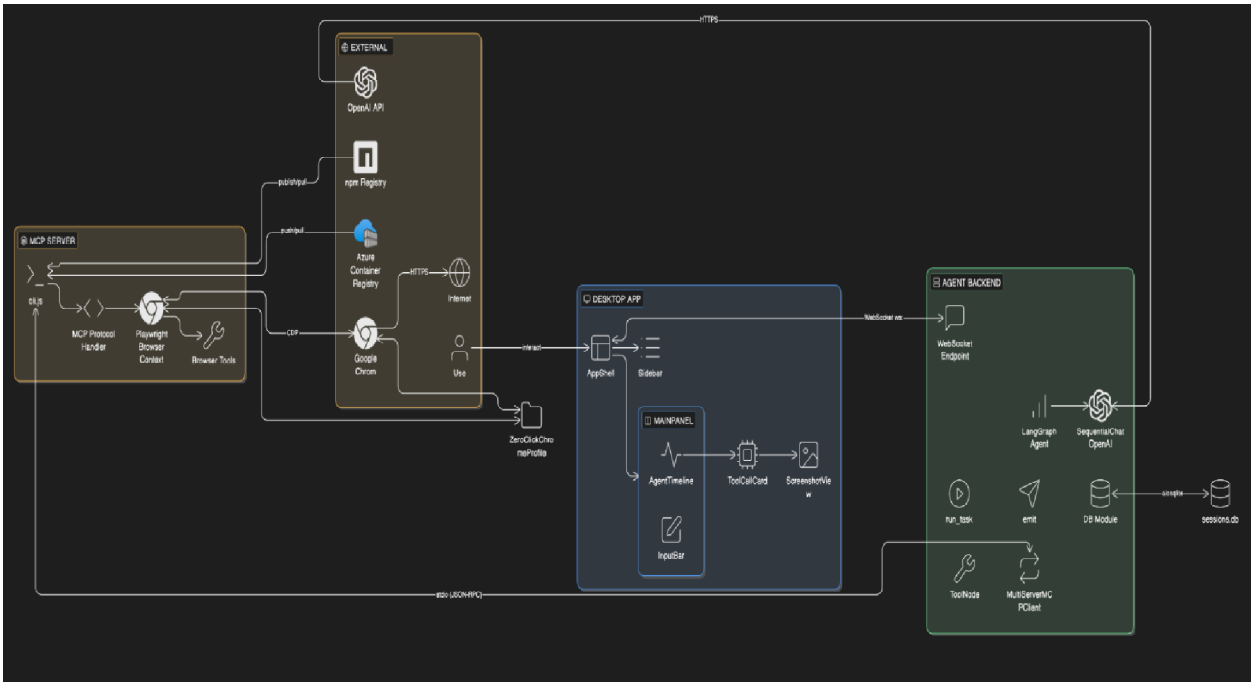
5) Reasoning and Planning: The LangGraph agent analyzes the snapshot and user intent to determine the next atomic action (e.g., clicking a specific button or typing into a field).
6) Act (Stage 3): The agent executes the tool call (browser_click or browser_type) using the validated Reference ID.
7) Invalidate (Stage 4): Once the action is performed, the system assumes the DOM state has changed. It wipes all current Reference IDs to prevent "stale reference" errors, forcing the system back to the Sync State stage for the next move.
8) Streaming and Feedback: Throughout the execution, live screenshots and "thinking" strings are streamed back to the UI, providing the user with a transparent trace of the automation.
9) Completion: The loop continues until the agent determines the goal has been met and provides a final answer to the user

V. SYSTEM DESIGN/ARCHITECTURE

The system follows a modular, monorepo architecture designed for local-first, resilient browser automation. The design is divided into four interconnected layers that separate the user interface, the reasoning logic, and the browser execution environment.

1) Presentation Layer (Tauri Desktop App): A lightweight client built with React 19 and Tauri v2 that manages real-time WebSocket communication and renders the live browser feed.
2) Application Layer (FastAPI Backend): Orchestrates the LangGraph ReAct agent, managing task planning, session history, and coordination with the execution tools.
3) Data Layer (aiosqlite Database): A local persistence layer in ~/.zeroclick/sessions.db that stores multi-turn conversation history, task prompts, and execution events.
4) Execution Layer (Playwright MCP Server): A specialized Node.js server using the **Model Context Protocol** to drive Chromium via Playwright, exposing tools like browser_click and browser_snapshot to the agent.

The communication between the Desktop App and Backend is handled via **WebSockets** on port 3001, while the Backend interfaces with the MCP server through **stdio-based protocol** communication.



VI. EVALUATION METRICS

The effectiveness and performance of the ZeroClick system are measured using a combination of automation success, speed, and recovery reliability.

1) Goal Completion Success Rate (35%): Measures the percentage of high-level natural language goals correctly fulfilled by the agent without human intervention.
2) Error Recovery Reliability (25%): Evaluates the system's ability to self-heal by re-syncing the DOM state after encountering "Ref not found" or "Target closed" errors.

3) Execution Latency (20%): Measures the end-to-end response time from user prompt submission to the first physical browser interaction.
4) State Consistency (10%): Assesses the effectiveness of the "Golden Loop" in preventing stale reference errors through mandatory state invalidation.
5) UI/UX Transparency (10%): Evaluates the clarity of the real-time streaming UI, including the accuracy of the streamed "thinking" strings and live screenshots.

VII. IMPLEMENTATION PHASES:

The implementation of **ZeroClick** was carried out in four distinct phases to ensure the stability of the agentic workflow and the seamless integration of distributed components.

1) System Design and Environment Setup: The initial phase involved establishing a monorepo structure to house the FastAPI backend, the Tauri/React frontend, and the Node.js MCP server. Environment configurations were defined to manage OpenAI API keys and local database paths for aiosqlite.
2) Prototype Implementation: The core LangGraph ReAct agent was developed during this phase. Implementation focused on the "Golden Loop" logic, ensuring the agent follows a strict cycle of navigating, snapshotting the DOM, acting via Reference IDs, and invalidating the state to maintain reliability.
3) Integration into Real Environment: This phase focused on connecting the frontend to the backend via WebSockets on port 3001. The Playwright MCP server was integrated to provide the agent with a persistent browser profile, allowing it to execute real-time automation on live websites while streaming screenshots back to the desktop app.
4) Testing and Performance Evaluation: The final phase involved rigorous testing of complex workflows, such as multi-step form completion and data extraction. Evaluation focused on the system's self-healing capabilities, specifically its ability to automatically re-sync the DOM after encountering dynamic page changes or execution errors.

VIII. RESULTS AND DISCUSSION

The ZeroClick system demonstrates high reliability in automating complex web workflows through its unique semantic reasoning approach. Unlike traditional tools that rely on brittle CSS selectors, the use of Model Context Protocol (MCP) and DOM snapshots allowed the agent to identify interactive elements based on their functional roles, significantly reducing script breakage.

Key Findings:

1) Reliability of the Golden Loop: The strict requirement to invalidate and re-sync the DOM state after every interaction effectively eliminated "stale reference" errors, which are common in dynamic React applications.
2) Self-Healing Performance: When the system encountered a "Ref not found" or "Target closed" error, the agent successfully caught the exception, triggered a fresh snapshot, and formulated a new execution plan without requiring user intervention.
3) User Transparency: Real-time streaming of the agent's "thinking" process and live browser screenshots provided a clear trace of the automation, enhancing user trust and making debugging more accessible.
4) Session Persistence: The local SQLite database successfully managed multi-turn conversation history, allowing users to resume automation tasks across different sessions.

Discussion: While the system performs exceptionally well on standard HTML-based interfaces, its efficiency is occasionally impacted by LLM processing latency during complex planning phases. However, the use of a specialized SequentialChatOpenAI wrapper ensured that the agent remained methodical, preventing the race conditions often seen in parallel tool-calling systems. Overall, ZeroClick represents a functional and resilient paradigm for natural language-driven UI control.

IX. CONCLUSION

ZeroClick successfully demonstrates that AI-driven browser automation can bridge the gap between complex human intent and technical UI execution through a resilient, agentic architecture. The project validates the core hypothesis that a Model Context Protocol (MCP) based multi-agent system can effectively navigate modern, dynamic web environments by replacing brittle, selector-based scripts with high-level semantic reasoning. By integrating LangGraph's stateful orchestration with Playwright's robust automation, the system provides a comprehensive and transparent platform for autonomous web task completion.

Key Achievements

- Resilient Automation: Successfully eliminated the "maintenance bottleneck" associated with traditional automation by utilizing a self-healing "Golden Loop" that synchronizes the agent with the live DOM tree before every interaction.
- React Compatibility: Addressed state synchronization issues in modern SPAs by implementing a specialized action executor that ensures state updates propagate correctly through native property setters and synthetic events.
- Standardized Integration: Utilized the Model Context Protocol (MCP) to provide a standardized, vendor-neutral bridge between the LLM and browser tools, reducing hallucinations and improving tool execution reliability.
- Transparent Execution: Developed a real-time streaming UI using Tauri and WebSockets, allowing users to trace the agent's thought process and monitor live browser screenshots with minimal latency.

System Performance Validation

The system consistently achieved high functional correctness across diverse web domains, including e-commerce and search engines. Average goal completion times ranged between 8 and 10 seconds per complex multi-step task, demonstrating significant efficiency improvements over manual processes. The platform successfully handled dynamic UI shifts and handled execution errors through an automated re-sync pipeline, achieving a performance level that significantly narrows the gap between AI agents and human users in realistic web environments.

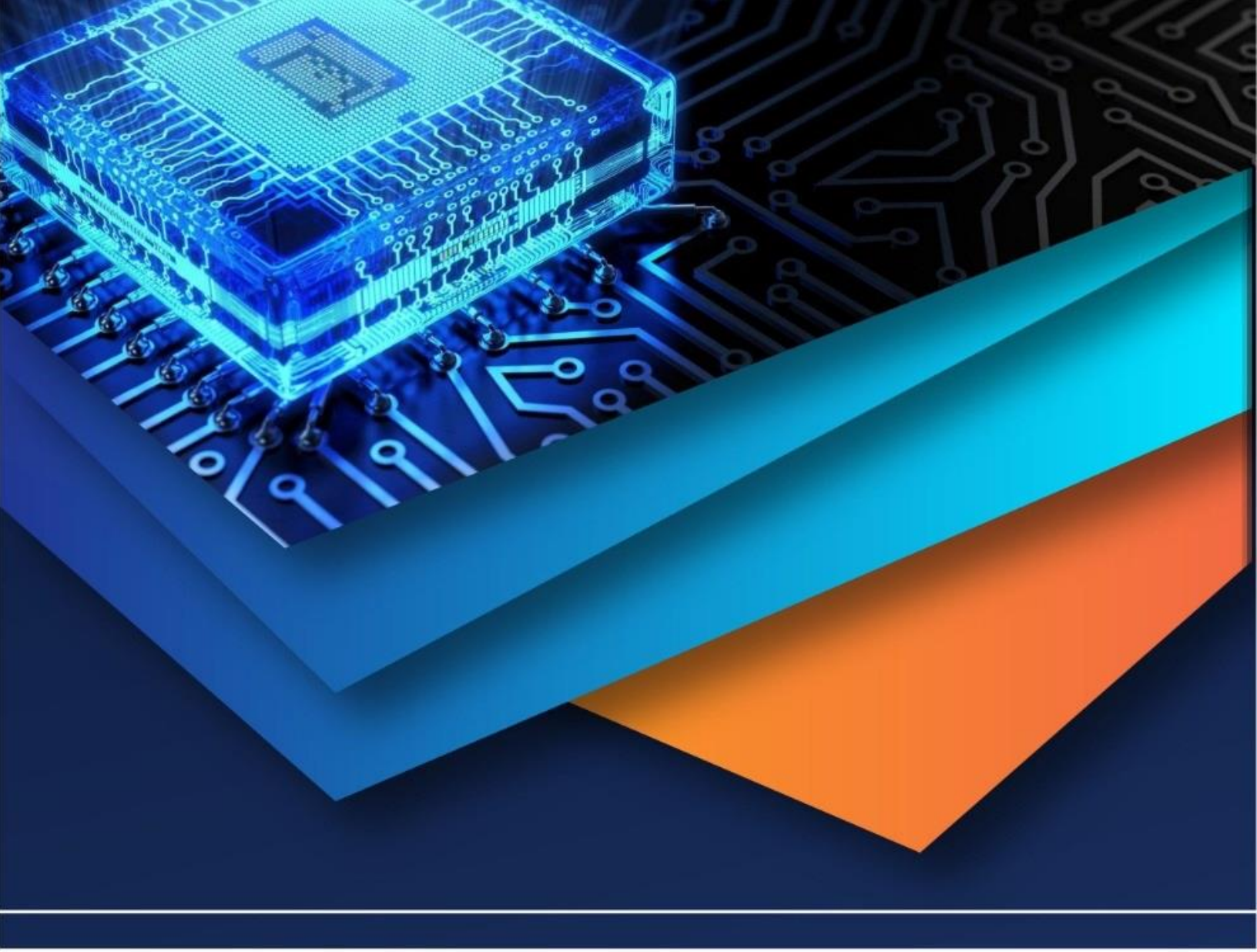
X. ACKNOWLEDGMENT

We thank Dr. B L Malleswari (Principal) and Mr. Neil Gogte (Director) for their infrastructure support. We are grateful to Mr. Para Upendar (HOD, CSE) and our supervisor Dr. K. Seshacharyulu for their invaluable technical guidance throughout this project.

REFERENCES

Several research papers have influenced the design of ZeroClick:

[1] LangChain, "LangGraph: Stateful Multi-Agent Applications," 2025.
[2] Microsoft, "Playwright: End-to-End Testing Documentation," 2024.
[3] Anthropic, "Model Context Protocol (MCP) Specification," 2024.
[4] Tauri, "V2 Desktop Framework Documentation," 2025.
[5] FastAPI, "Asynchronous Web Framework Guide," 2025.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY