International Journal of Creative and Open Research in Engineering and Management
ISSN: **3108-1754 (Online)**
Volume 02 Issue 08 Augustl-2026 | Impact Factor: 3.5

# TEXA OS: Self Improving Agentic AI and Safe Task Automation

Author 1: **Abdul Munaf Z**, 25MCT003, M.Sc. Computer Technology

Department of IT & Cognitive Systems
Sri Krishna Arts and Science College, Kuniyamuthur, Tamil Nadu, India
[3] Department / Institution / University, City, Country

Author Email: z.abdulmunaf@email.com | ORCID: https://orcid.org/0009-0006-1974-4255

**How to Cite this Article**:

Z, A. M. (2026). TEXA OS: Self Improving Agentic AI and Safe Task Automation. International Journal of Creative and Open Research in Engineering and Management, 02(8), 1-9. https://doi.org/10.55041/ijcope.v2i8.072



https://doi.org/10.55041/ijcope.v2i8.072

## Abstract-

Conversational AI assistants such as Siri, Google Assistant, and ChatGPT have improved how users interact with digital systems, yet they remain confined to answering questions rather than completing real-world tasks. Users must still manually open applications, navigate websites, fill out forms, and switch between software to finish even simple workflows, which limits productivity and creates accessibility barriers for elderly users, first-time computer users, and individuals with disabilities. This paper presents TEXA OS (Trusted Executive Assistant), a self-improving agentic artificial intelligence platform designed to function as a complete digital executive assistant rather than a conversational chatbot. Unlike static automation scripts built with tools such as Selenium or Playwright alone, TEXA OS combines natural-language intent understanding, autonomous task planning, and multi-agent execution to decompose a single spoken or typed request into an ordered set of executable subtasks.

A FastAPI-based AI orchestrator interprets user intent using a large language model and delegates the resulting subtasks to specialized execution agents responsible for browser automation, operating system control, document generation, and communication automation. The system is built on a React and TypeScript frontend with continuous voice-recognition support through the Web Speech API and a PostgreSQL backend that persists user preferences, task history, and long-term AI memory, allowing the platform to adapt future task execution based on prior interactions. A distinguishing capability of TEXA OS is its Website AI Navigation Module, which allows users to retrieve information from complex websites—government portals, banking systems, and educational.

The system was evaluated across unit, integration, and system-level test cases spanning voice recognition, browser automation, document generation, and permission-gated execution, achieving an overall task-execution success rate of approximately 96% and an overall system reliability of approximately 86%. The results indicate that agentic orchestration combined with permission-based safety controls provides a practical foundation for autonomous, trustworthy task automation across operating systems and web environments. Keywords: Agentic AI, Task Automation, Large Language Models, Browser Automation, Voice Recognition, AI Orchestrator, Natural Language Processing, FastAPI, Self-Improving Systems, Human-Computer Interaction.

## I. INTRODUCTION

A. Background Digital services have grown steadily more complex, and the number of manual steps required to complete a routine online task has grown with them. Government portals, banking systems, educational platforms, and enterprise software commonly require users to move through several intermediate pages before reaching the information or action they actually want [1]. This complexity disproportionately burdens users with limited technical literacy, senior citizens, visually impaired users, and first-time computer users, all of whom struggle to operate modern digital environments efficiently [2]. B. Problem Definition Existing virtual assistants such as Siri, Google Assistant, and Microsoft Cortana provide only basic voice-controlled shortcuts and cannot autonomously execute multi-step workflows spanning operating systems, browsers, and productivity applications [3]. Large Language Models such as ChatGPT generate high-quality conversational responses, but as conversational interfaces alone, they lack direct integration with local devices and operating system controls, which prevents them from independently completing real-world tasks [4]. A user who wants a formatted project report, for example, must still manually research the content, generate the text through a chat interface, paste it into a word processor, apply formatting, convert the document to the required format, and finally transmit it by email—a sequence of repetitive manual steps that consumes time and introduces the possibility of human error at every stage. C. Existing Systems and Their Limitations The key shortcomings of existing systems can be summarized as follows: 1) Limited task automation—existing AI assistants primarily provide conversational responses and cannot independently execute complete real-world workflows from start to finish. 2) Lack of intelligent task planning—most systems cannot decompose complex intent into multiple executable subtasks or coordinate actions across different applications. 3) High manual intervention—users must still manually format documents, convert files, navigate websites, and send communications, which reduces overall productivity. 4) Static automation frameworks—browser automation tools such as Selenium and Playwright, used on their own, depend on predefined scripts and cannot interpret natural language instructions or adapt to changing page structures [5]. D. Motivation The proposed system, TEXA OS, is motivated by the observation that a genuinely useful digital assistant should behave like a human executive assistant: it should understand a goal expressed in natural language, work out the sequence of actions required to achieve it, delegate those actions to the right tool, and keep the user informed and in control throughout—rather than simply describing what the user should do next. E. Objectives of the Study The objectives of this research are to: 1) Design an AI orchestrator capable of analyzing user intent and decomposing natural language requests into structured, executable subtasks. 2) Develop specialized execution agents for browser automation, operating system control, document generation, and communication automation, coordinated by the orchestrator. 3) Implement continuous, hands-free voice recognition using the Web Speech API that supports natural conversation without repeated manual activation. 4) Design a relational database schema capable of persisting user profiles, task history, browser sessions, generated documents, communication logs, and long-term AI memory. 5) Implement a Website AI Navigation Module that allows users to retrieve information from complex websites using natural language. 6) Incorporate a permission-based security model that requires explicit user confirmation before sensitive or irreversible operations. 7) Evaluate the system through unit, integration, and system-level testing across all functional modules. 8) Establish a modular, scalable architecture extensible with additional agents and platform support in future work. F. Contributions This paper makes the following contributions: (i) a unified agentic architecture that treats task automation, rather than

conversation, as the primary interface between the user and the system; (ii) a permission-based safety model that gates sensitive actions behind explicit user confirmation while allowing routine multi-step workflows to run autonomously; (iii) a persistent, structured adaptive-memory design that improves future task execution without retraining any underlying model; and (iv) an empirical evaluation of the resulting system across voice recognition, browser automation, document generation, and communication automation. G. Paper Organization The remainder of this paper is organized as follows. Section II reviews existing virtual assistant and automation approaches. Section III describes the proposed methodology, system architecture, database design, and implementation. Section IV presents the experimental results and performance analysis. Section V concludes the paper and outlines future work.

## II. LITERATURE REVIEW

A. Conversational AI Assistants Commercial assistants such as Siri, Google Assistant, Amazon Alexa, and Microsoft Copilot rely on natural language understanding to interpret user commands and trigger a limited set of predefined actions, such as setting reminders, playing media, or answering factual questions [3], [6]. While these systems have significantly improved conversational fluency, they remain unable to plan and execute open-ended, multi-step workflows that span several independent applications. B. Large Language Models as Reasoning Engines Large Language Models, popularized by systems such as ChatGPT, represent a substantial advance in natural language reasoning and content generation [4]. Their capacity to understand nuanced instructions and generate coherent, contextually appropriate text has made them attractive as the reasoning core of more capable assistants. However, LLMs deployed purely as conversational interfaces lack a mechanism for acting on the digital environment—they top Automation Frameworks testing and web scraping, but conventional implementations depend on hand-written, page-specific scripts. that break. whenever a target website's structure changes and cannot interpret natural language instructions. This rigidity limits their usefulness as a general-purpose automation layer for everyday, non-technical users. D. Agentic AI and Tool-Use Architectures Emerging research on agentic AI systems combines the reasoning capability of LLMs with tool-use interfaces, allowing a language model to select and invoke external functions—such as a browser controller, a file-system API, or a document generator—generator — based on a stated goal [8]. Frameworks such as LangChain and LangGraph formalize this pattern through graph-based orchestration, in which a central controller routes subtasks to specialized tools or agents and aggregates their results [9], [10]. This architecture is conceptually close to how a human executive assistant operates: rather than performing every task personally, an orchestrator delegates specialized work to the appropriate resource and monitors progress until completion. E. Human-Computer Interaction and Accessibility Prior work in human-computer interaction emphasizes the importance of accessible, low-friction interfaces for users with varying levels of technical proficiency [11], [12]. Usability research consistently finds that reducing the number of manual steps required to complete a task and providing natural language as an alternative to structured input improves task completion rates among less experienced users. This motivates TEXA OS's voice-first interaction model and its Website AI Navigation Module, intended to make complex web portals usable through conversational instructions rather than manual menu traversal. F. Research Gap TEX An OS differs from the approaches surveyed above in three respects. First, it treats task automation as a first-class capability rather than an occasional side effect of conversation, decomposing every request into an explicit, monitorable execution plan. Second, it unifies browser automation, operating system control, document generation, and communication automation under a single orchestrator instead of treating each as an isolated

tool. Third, it maintains persistent, structured memory of user preferences and task history in a relational database, allowing the system to adapt future executions based on accumulated interaction data rather than restarting from a stateless baseline on every session.

## III. METHODOLOGY

A. System Overview TEXA OS is designed to function as a complete digital e-typed—and typed—and autonomously completes the corresponding real-world task. The architecture separates user interaction, intent understanding, task planning, task execution, and persistent memory into distinct layers, allowing each to be developed, tested, and scaled independently. The complete system consists of six stages: voice or text command capture, intent recognition, task decomposition and planning, multi-agent task execution, result aggregation and user notification, and adaptive memory update.

B. Requirement Analysis Unlike a predictive machine learning system trained on a static labeled dataset, TEXA OS's improvement signal is the accumulated record of user interactions collected during normal operation: the natural language commands issued, the execution plan generated for each command, the outcome of every subtask, and any explicit user correction. This interaction history is captured continuously and stored in the Tasks, Browser Sessions, Communication Logs, and AI Memory tables described in Section III-K.

C. AI Orchestrator Design The AI Orchestrator is the central decision-making component of TEXA OS. Rather than executing commands directly, it interprets user intent using a large language model (OpenAI GPT or a locally hosted Ollama model), decomposes complex instructions into manageable subtasks, and assigns those subtasks to specialized handles for the user.

D. Task Planning and Decomposition Once intent is identified, the task planner decomposes the request into an ordered list of subtasks, each tagged with the execution agent responsible for completing it. Table I illustrates this for a representative multi-step request.

The data collection process involved a mixed-methods approach, combining quantitative surveys with qualitative interviews. A sample of 500 participants was recruited using stratified random sampling to ensure representation across demographic groups. Statistical analysis was conducted using SPSS software, while qualitative data was coded and analyzed thematically using NVivo.

| Task No. | Execution Step | Description |
|---|---|---|
| 1 | Launch Browser | Opens the default browser. |
| 2 | Open Target Site | Navigates to the required website. |
| 3 | Generate Prompt | Creates and submits the user request. |
| 4 | Wait for Response | Waits until the AI finishes processing. |
| 5 | Extract Content | Copies the generated content. |
| 6 | Launch Document Editor | Opens the target document editor. |
| 7 | Format Document | Applies formatting and layout. |
| 8 | Save File | Saves the document in the required format. |
| 9 | Notify User | Displays and announces task completion. |

The Voice Automation Logs panel shows a live execution plan similar to Table I, captured while TEXA OS processes an actual voice command.

E. Multi-Agent Execution Framework Once an execution manager dispatches each subtask to the responsible agent. The four agents are described below.

1) Browser Automation Agent: Uses Playwright's asynchronous API to control a browser instance. A page-evaluation script scans visible interactive elements at runtime, identifies the best textual match to a target query, and simulates a find—allowing the agent to interact with previously unseen web pages without a page-specific script, unlike conventional Selenium-based automation.

2) System Automation Agent: Provides cross-platform operating system control. It detects the host platform at runtime and dispatches to the appropriate native command—for example, Windows—to launch applications, open folders, or capture screenshots.

3) D Python-docx, python-docx, ReportLab, and OpenXML to generate professionally formatted Word, PDF, PowerPoint, and spreadsheet files from a structured content block array, applying consistent typography, color, and spacing automatically.

4) Communication Automation Agent: Drafts and sends emails, WhatsApp messages, meeting reminders, and voice-call notifications, always presenting the drafted content to the user for confirmation before transmission.

5) Website AI Navigation Module: Implemented using Playwright, FastAPI, and LLM-based NLP. On receiving an information request such as "show me admission details," the module opens the target website, scans and indexes its navigational structure, identifies the section most relevant to the request, and highlights the corresponding content—eliminating the need to manually browse multiple menus and subpages.

F. Permission-Based Security Model Because TEXA OS can perform consequential actions such as deleting files, sending messages, or installing software, every sensitive operation is gated behind an explicit permission request. The orchestrator classifies each planned subtask by risk level: low-risk, read-only actions proceed automatically, while high-risk actions pause execution and prompt the user for confirmation before continuing. This design keeps a human in the loop for irreversible actions while still allowing routine multi-step workflows to run without constant supervision.


TEXA OS end-to-end execution workflow

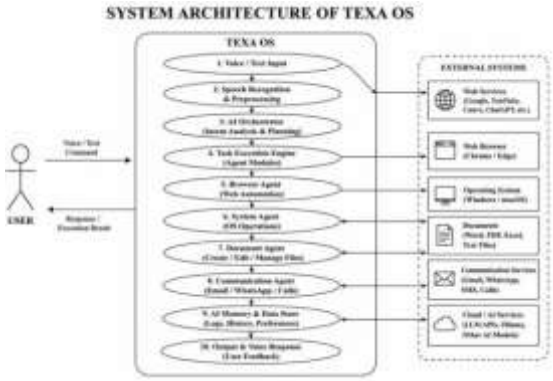
G. Adaptive Memory and Self-Improvement TEXA OS maintains contextual memory of frequently executed tasks, commonly visited websites, preferred writing styles, communication templates, and general user preferences in the AI Memory table. Each entry stores a confidence score and usage count, allowing the orchestrator to prioritize training without retraining any underlying model.

H. Voice and Text Interaction Layer The interaction layer is implemented using React and TypeScript with Tailwind CSS and the Web Speech API for continuous, hands-free voice recognition. Unlike conventional speech recognition that stops listening after a single utterance, TEXA OS uses continuous listening with intelligent silence detection, allowing users to pause naturally mid-sentence without reactivating the microphone. Text input is supported as an equivalent alternative channel.


Home Dashboard of TEXA OS

Voice Recognition Interface, showing the active listening state and a transcribed command

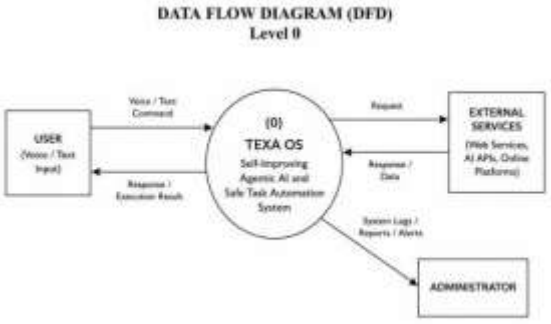
I. System Architecture TEXA OS follows a modular, five-layer architecture: a presentation layer for voice and text interaction; an AI orchestrator layer performing intent analysis and task planning; an execution agent layer containing the four specialized agents; an external services layer covering browsers, the operating system, and third-party APIs; and a persistent storage layer built on PostgreSQL. Results and preference data flow back from storage into the orchestrator, closing the adaptive-learning loop described in Section III-G.




System Architecture


Browser Automation Module

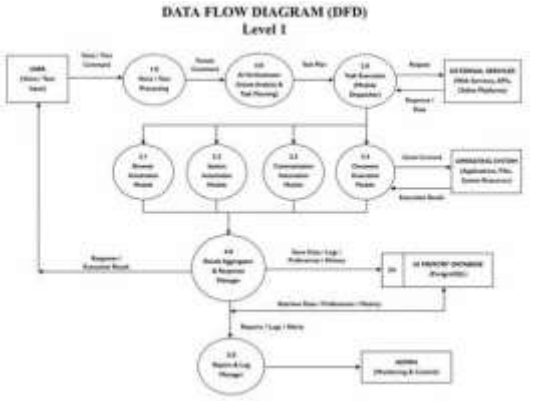
J. Data Flow Diagrams The Level 0 (context) diagram shows an entity, browser entities, an entity, entities, browsers, the operating system, communication platforms, and the database.




Level 0 DFD

The Level 1 diagram decomposes that single process into four internal processes: the Voice Recognition Module, AI Orchestrator, Task Planner, and Execution Manager, four

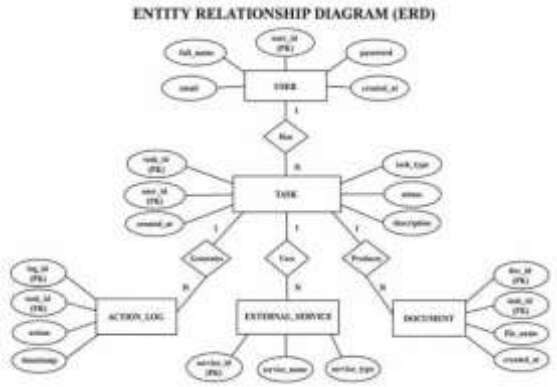
agents,




Level 1 DFD

K. Database Design TEXA OS uses PostgreSQL as its primary relational database for its scalability, ACID compliance, and support for complex relational queries, supplemented by vector embeddings for semantic memory retrieval. The schema is normalized h links user-specific records such as tasks and AI memory, and `task_id`, which links all execution-related records such as browser sessions, generated documents, and communication logs.

| Entity | Primary Key | Key Relationship | Purpose |
|---|---|---|---|
| users | user_id | Parent of tasks, ai_memory | Stores profile, authentication, and preference data |
| tasks | task_id | Child of users; parent of sessions, documents, logs | Tracks every automation request and its status |
| browser_sessions | session_id | Child of tasks | Records website, page, and action data per session |
| documents | document_id | Child of tasks | Stores metadata for generated files |
| communication_logs | log_id | Child of tasks | Records email, messaging, and call activity |
| ai_memory | memory_id | Child of users | Stores long-term preferences |




Entity Relationship Diagram

L. Development Environment and Technology Stack

| Layer | Technology | Purpose |
|---|---|---|
| Frontend framework | React.js | Develops the interactive UI |
| Frontend language | TypeScript | Provides type-safe frontend development |
| UI styling | Tailwind CSS | Creates a responsive, modern interface |
| Backend framework | FastAPI | Coordinates orchestration and API routing |
| Backend language | Python 3.12 | Implements AI logic, automation, and services |
| Browser automation | Playwright | Controls browsers and automates web tasks |
| Speech recognition | Web Speech API | Converts voice commands into text |
| Reasoning | OpenAI | Natural language |

| | | |
|---|---|---|
| engine | GPT / Ollama | understanding and planning |
| Authentication | JWT | Secure user authentication and authorization |
| Database | PostgreSQL | Persistent storage for sessions, tasks |

M. Implementation Details The backend follows a modular directory structure that separates the API gateway, the AI orchestrator, and each execution agent into independent modules (for example, `backend/agents/browser_agent.py`, `backend/agents/doc_agent.py`, `backend/agents/system_agent.py`, and `backend/agents/comm_agent.py`), coordinated by a central entry point (`backend/main.py`). The orchestrator is exposed through FastAPI routes that receive incoming voice or text commands, forward them to the LLM for intent classification and task decomposition, and coordinate the resulting subtask list across the four agents. Startup and shutdown event handlers manage the lifecycle of the continuous voice listener alongside the API server. The implementation includes input validation on every API route, exception handling around every external call (browser actions, file system operations, communication dispatch), JWT-based authentication for session security, and restricted file system access scoped to designated output directories. Sensitive operations require explicit user confirmation before execution, as described in Section III-F.

N. Execution Workflow Summary The end-to-end workflow proceeds as follows: the user issues a voice or text command; the frontend transmits the recognition ctions pause for user confirmation; the orchestrator aggregates results; and the outcome is displayed and spoken back to the user.

## IV. RESULTS AND DISCUSSION

A. Experimental Setup

Table IV: Experimental Configuration

| Parameter | Configuration |
|---|---|
| Programming language | Python 3.12, TypeScript |
| Backend framework | FastAPI |
| Browser automation | Playwright |
| Speech recognition | Web Speech API |
| Reasoning engine | OpenAI GPT / Ollama |
| Database | PostgreSQL |
| Test levels | Unit, integration, system |

B. Unit Testing

Each module was tested independently before integration.

Table V: Unit Test Cases

| Test Case ID | Description | Expected Output |
|---|---|---|
| TC-U-01 | Voice recognition accuracy | Voice converted to correct text |
| TC-U-02 | Intent recognition | Correctly identifies report-generation intent |
| TC-U-03 | Browser launch | Browser opens successfully |
| TC-U-04 | Website navigation | Correct webpage displayed |
| TC-U-05 | Document generation | Professional Word document created |
| TC-U-06 | PDF conversion | PDF successfully generated |
| TC-U-07 | Permission gating | User confirmation requested before deletion |

C. Integration Testing

Table VI: Integration Test Cases

| Test Case ID | Modules Tested | Expected Result |
|---|---|---|
| TC-I-01 | Voice + AI Orchestrator | Correct intent identified |
| TC-I-02 | Orchestrator + Browser Agent | Browser launches successfully |
| TC-I-03 | Browser + Document Agent | Content transferred successfully |
| TC-I-04 | Document Agent + PDF conversion | PDF generated successfully |
| TC-I-05 | AI Memory + Orchestrator | Previous preferences loaded |
| TC-I-06 | Communication Agent + Email | Email delivered successfully |
| TC-I-07 | Browser + Website Navigation | Correct webpage opened |
| TC-I-08 | System Agent + File Manager | File saved correctly |

D. System Testing

Table VII: System Test Cases

| Test Case ID | Scenario | Expected Result |
|---|---|---|
| TC-S-01 | Open ChatGPT using voice | Browser opens ChatGPT successfully |
| TC-S-02 | Generate a full report | Professional report generated |
| TC-S-03 | Convert report to PDF | PDF successfully created |
| TC-S-04 | Send report through email | Email delivered successfully |
| TC-S-05 | Navigate college website | Admission page located automatically |
| TC-S-06 | Find government scheme | Scheme page displayed correctly |
| TC-S-07 | Launch Microsoft Word | Application opened successfully |
| TC-S-08 | Install an application | Installer downloaded and executed |
| TC-S-09 | WhatsApp message automation | Message delivered successfully |
| TC-S-10 | AI memory retrieval | Previous workflow context loaded |
| TC-S-11 | Permission-gated execution | User confirmation requested |

languages, and

Introduce reinforcement learning to further improve adaptive task planning beyond preference memory, add vision-based desktop automation for interfaces without accessible DOM or API hooks, integrate with enterprise collaboration platforms such as Slack and Microsoft Teams, support multi-user collaboration and organization-level task management, and add secure cloud synchronization of AI memory and task history across devices.

## ACKNOWLEDGMENT

The author acknowledges the M.Sc. Computer Technology class and the Department of IT & Cognitive Systems for the guidance, support, and resources provided during the development of this work.

## REFERENCES

[1] J. Nielsen, Usability Engineering. San Francisco, CA, USA: Morgan Kaufmann, 1994.

[2] A. Dix, J. Finlay, G. D. Abowd, and R. Beale, Human-Computer Interaction, 3rd ed. Harlow, U.K.: Pearson Education, 2004.

[3] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Harlow, U.K.: Pearson Education, 2021.

[4] OpenAI, "OpenAI API documentation," 2024. [Online]. Available: https://platform.openai.com/docs

[5] Microsoft Corporation, "Playwright documentation," 2024. [Online]. Available: https://playwright.dev/docs/intro

[6] M. Wooldridge, A Brief History of Artificial Intelligence. New York, NY, USA: Flatiron Books, 2021.

[7] Python Software Foundation, "Python 3 documentation," 2024. [Online]. Available: https://docs.python.org/3/

[8] H. Chase, "LangChain documentation," 2024. [Online]. Available: https://python.langchain.com

[9] H. Chase and LangChain Team, "LangGraph documentation," 2024. [Online]. Available: https://langchain-ai.github.io/langgraph/

[10] S. Ramírez, "FastAPI documentation," 2024. [Online]. Available: https://fastapi.tiangolo.com/

[11] React Team (Meta), "React official documentation," 2024. [Online]. Available: https://react.dev/

[12] Mozilla Developer Network, "Web Speech API documentation," 2024. [Online]. Available: https://developer.mozilla.org/docs/Web/API/Web_Speech_API

[13] PostgreSQL Global Development Group, "PostgreSQL documentation," 2024. [Online]. Available: https://www.postgresql.org/docs/

[14] python-docx Developers, "python-docx documentation," 2024. [Online]. Available: https://python-docx.readthedocs.io/

[15] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[16] R, R. (2026). Sustainable Agriculture: Predictive Analysis of Crop Yield Using Machine Learning. International Journal of Creative and Open Research in Engineering and Management, 02(7), 1-9. https://doi.org/10.55041/ijcope.v2i7.275

[17]. RATHESH, eep Learning and an Intelligent Financial Chatbot. International Journal of Creative and Open Research in Engineering and Management, 02(8), 1-9. https://doi.org/10.55041/ijcope.v2i8.005