Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Intelligent Context-Aware Screen Assistance System for Real-Time Productivity Enhancement

Nandeesh M Doggalli[1], Gausiya Riffath Khan[2], Mahek Kadak[2], Nitish Kumar T[2], Parvathi M[2]
[1]Assistant Professor, Department of Information Science and Engineering
[2]UG Scholars, Department of Information Science and Engineering
[1,2]GM Institute of Technology, Davangere

***Abstract*—Fixing a problem on a computer usually means leaving the task. The user copies an error message, searches the web, and then tries to apply what they find. This paper describes a screen assistance system that removes most of that detour. It watches the screen, reads the visible text with optical character recognition (OCR), identifies interface elements, and builds a structured description of what the user is doing. When it finds a problem, it rates the severity and asks a large language model (LLM) for a fix. The answer appears as an issue card with the root cause, numbered steps, a command that can be copied, and a confidence value. The assistant runs in a React dashboard and in a floating Windows overlay, and it also provides a prompt library, a knowledge base, voice mode, and a searchable history of past sessions. A Node.js and MongoDB back end handles login and storage, and capture starts only after the user consents. The prototype was tested at unit, integration, functional, and system levels. In a worked example, it recognized a missing-module error in a terminal and proposed the correct repair, which suggests that proactive, screen-aware help is practical on an ordinary desktop.**

***Index Terms*—Desktop assistant, screen understanding, context-aware computing, optical character recognition, large language models, proactive assistance, productivity**

I. INTRODUCTION

Much of today's knowledge work happens on a desktop screen. Developers, students, and office staff move all day between editors, terminals, browsers, and documents. When an error appears, most people follow the same routine: copy the message, search the web or ask a chatbot, read a few answers, and then try to remember where they left off. Each detour is short, but together they add up and interrupt concentration.

Desktop assistants have done little to change this. Voice assistants can send emails, schedule tasks, and control devices, which makes computers easier to use [1], [2]. However, they wait for a spoken or typed request and know nothing about what is on the screen. Reviews of voice interfaces still list speech accuracy and context handling as open problems [3]. Debugging tools can show that an error exists, yet they rarely explain what it means for the person looking at it. The user has to describe the problem to a tool that could have seen it directly.

We built a system that fills this gap. It captures the screen, extracts the text and interface elements, summarizes the situation in a structured form, and offers a step-by-step suggestion without being asked. It is available through a React dashboard and a floating Windows overlay, and it also includes a prompt library, a local knowledge base, voice mode, and a searchable history of watched sessions. Suggestions come from a large language model (LLM), and the user's feedback on each one is stored so that later answers can improve.

What sets this work apart is the combination: continuous screen understanding, automatic detection of problems with severity levels, and an LLM explanation of each problem, all in one desktop tool. The multi-utility assistant that inspired this project [4] concentrates on commands and utilities rather than on watching what the user is doing. We also treat privacy as a starting point. Nothing is captured until the user agrees, capture can be paused at any time, and stored data is encrypted.

The rest of the paper is organized as follows. Section II reviews related work, Section III describes the proposed solution, Section IV presents the testing and results, and Section V concludes with future work.

II. RELATED WORK

The multi-utility assistant mentioned above bundles useful utilities, such as OCR, reminders, and developer tools, into one desktop application that stays light on CPU and memory. A survey of desktop automation frameworks, including COSMO, stresses lightweight and privacy-aware designs but also reports that varied user interfaces remain hard to handle [5]. Voice assistants in the style of JARVIS, discussed earlier, combine speech recognition, language processing, and device control, yet they rely on spoken commands and depend heavily on the host system. Browser-based LLM assistants expose weaknesses in privacy and scalability [6]. Task-specific assistants for cooking guidance [7], document interaction [8], and sign-language conversion [9] show that assistants can be tailored to a purpose, but each is built around one domain rather than the user's general desktop activity.

Work on context-aware assistance goes beyond fixed commands. A framework for context-aware assistive systems at work combines gamification and projection to support the user in the task at hand [10], and collaborative intent prediction uses real-time contextual data to anticipate what a user wants [11]. An assistant built on OpenAI models can keep track of the long-running context of a conversation, although it does not scale easily [12]. Human–machine collaboration can

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

improve contextual understanding in personalized systems, but it needs manual feedback [13]. Surveys of intent detection and commonsense reasoning note how hard it is to run these methods in real time [14], and LitSense shows how a tool can support the workflows of academic literature review [15].

A taxonomy of vision-based multimodal interfaces offers a useful design framework for context-aware systems, but it has not been validated in real time [16]. A review of multimodal interaction highlights missing human-factors work and a shortage of user studies [17].

Context-aware recommender systems form a related line of work. Reviews of recent developments and future directions [18] and of challenges in user modeling and privacy [19] stress that context improves suggestions but raises privacy concerns. Context-aware recommenders for knowledge workers supply datasets and frameworks, but they have seen little adoption [20], and using context to recommend apps improves predictions, although the data behind it is limited [21]. Studies of human–AI decision making map the design space of empirical human-subject studies [22], and literature reviews on AI support for UX work [23] and on the design of interactive assistants [24] both find a gap between what the tools offer and what users actually need. Reviews of intelligent automation point to the effort needed to put these ideas into practice [25].

Among the works we surveyed, none combines continuous screen understanding, proactive detection of problems, and an LLM explanation of the problem in a single desktop tool. That is the gap this work addresses.

III. PROPOSED SOLUTION

The proposed system watches the screen, works out what the user is trying to do, and offers help before the user has to ask. It combines screen capture, OCR, interface-element detection, issue classification, and LLM-based solution generation in one application, so the user no longer has to switch between a terminal, a browser, and a chat tool.

A. System Architecture

The system is built in layers, and each layer has one job. The user and input layer holds the React dashboard and collects frames, input events, and the active window. The processing layer cleans each frame, extracts its text and finds interface elements. The AI layer analyzes the context, classifies the issue, and produces a solution. The response layer shows suggestions and alerts, while a set of core services manages sessions, feedback, and the knowledge base. A Node.js and MongoDB back end stores sessions, captures, issues, solutions, and logs. The front end uses React; the server uses Node.js and Express.js; OCR uses Tesseract.js; alerts travel over WebSocket; and Docker handles containerization. The assistant works in four modes: Standard, Coding Help, Writing Assistant, and Screen Awareness.

B. Key Features and Functionalities

The system brings several functions into one platform.

- **Screen Awareness:** captures the screen while the user works and skips frames that show no meaningful change.
- **Issue Detection:** places each problem into a category (error, warning, idle, code bug, or interface confusion) with a severity level.
- **Solution Generation:** returns the root cause, numbered steps, a copyable command, and a confidence value for each detected issue.
- **Floating Windows Overlay:** offers a compact assistant with Diagnose, Chat, and Settings tabs and a keyboard shortcut for diagnosing the active screen.
- **Prompt Library:** provides 16 built-in prompt templates with input placeholders, plus the user's own saved templates.
- **Knowledge Base and Voice Mode:** lets the user ask questions about local documents and talk to the assistant by voice.
- **History and Analytics:** stores every session and supports full-text search, pinning, archiving, export, and usage statistics.

C. Processing Pipeline

The system obtains frames through the screen-capture API, either at a fixed interval or when an event occurs. Privacy filters run before anything is captured. Each frame is converted to grayscale, thresholded, denoised, and resized. A Tesseract-based engine returns the recognized text with bounding boxes, and a vision-capable LLM can act as an alternative route for text extraction. A computer-vision model, together with window metadata, finds buttons, input fields, dialogs, and error widgets.

Next, the OCR text, interface metadata, and active-application information are merged into one structured context object. It records the text regions, the detected elements, the focused application, a timestamp, and recent history. A rule-based filter and a machine-learning classifier then assign the context to an issue category and give it a severity score, which is mapped to low, medium, or high.

When an issue is found, the system first searches the knowledge base for related entries. It then sends the context and the issue type to the LLM through its API and turns the reply into a suggestion card. Alerts reach the dashboard over WebSocket and follow the frequency and severity settings that the user has chosen. Whenever the user accepts, ignores, or rates a suggestion, the action is written to the activity log and fed back into the knowledge base and the classifier retraining process.

D. Data Model

A user owns many screen sessions; each session holds many captures; a capture can produce several detected issues; and each issue can have several suggested solutions. User actions are recorded separately in an activity log. Table I lists the main entities.

E. Security and Data Persistence

Data security and persistence are handled as follows:

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

TABLE I
MAIN DATA ENTITIES

Entity	Key fields	Relation
Users	id, name, email, password hash, role	1:N sessions, logs
ScreenSessions	id, user, start/end time, active app	1:N captures
ScreenCaptures	id, session, path, OCR text, time	1:N issues
DetectedIssues	id, capture, type, severity, confidence	1:N solutions
SuggestedSolutions	id, issue, text, model, helpful, feedback	–
ActivityLogs	id, user, action, time, details, IP	–

- Login relies on JSON Web Tokens kept in an HttpOnly cookie, and an account is locked after five failed attempts.
- Passwords must be at least eight characters long, with a strength indicator to guide the user.
- Stored user data is encrypted, and sessions, captures, issues, and solutions are saved in MongoDB.
- Nothing is captured until the user has given explicit consent, and capture can be paused at any time.

F. Implementation Strategy

The code is split into nine modules in three groups. The first group handles capture and extraction: screen monitoring, OCR, and interface detection. The second handles analysis: the context analyzer, the issue classifier, and the solution generator. The third handles delivery and memory: the notification manager, the feedback and knowledge-base module, and the history and reporting module. The prototype runs on Windows 10 and 11 with Node.js v18 and a local MongoDB instance.

The login form checks required fields, email format, and the password policy, and moves through idle, submitting, success, and error states. The registration form adds name, role, and password confirmation, checks for duplicate email addresses as the user types, and shows a privacy notice. Both forms are part of a React front end that also contains the dashboard, session history, and prompt library shown in Section IV.

IV. RESULTS AND DISCUSSION

The developed system was tested across multiple scenarios to evaluate its functionality, performance, and user experience. The evaluation (see Table II) focuses on accuracy, response time, resource use, and security of the main features. Testing used synthetic screen captures, mock OCR output, and dummy user accounts, and no real personal data was involved.

Figure 1 gives an overview of the main result screens. The Screen Sessions page (top left) lists every live-watch run with its captures and detected issues. The dashboard (top right) is the home screen, with shortcuts to a new conversation, voice mode, screen capture, and the knowledge base. The Screen Awareness view (bottom) shows the live capture of a terminal on the left and, on the right, the issue the system detected,

TABLE II
PERFORMANCE METRICS OF THE SCREEN ASSISTANCE SYSTEM

Metric	Idle State	Normal Usage	Heavy Load
CPU Usage (%)	–	–	–
Memory Consumption (MB)	–	–	–
End-to-End Response Time (s)	–	–	–
OCR Accuracy (%)	–	–	–
Issue-Classification F1	–	–	–

together with its root cause, numbered steps, and a fix that can be copied. Each screen is described in more detail below.

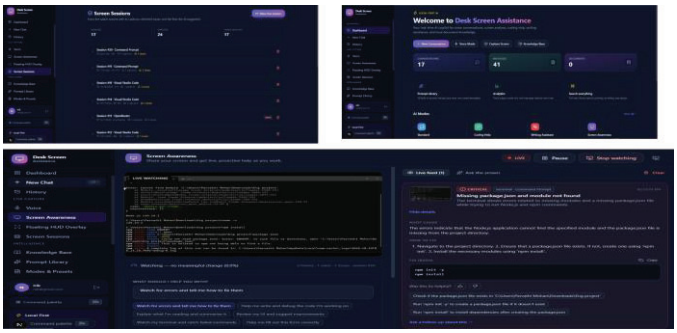


Fig. 1. Overview of the result screens of the Desk Screen Assistance application: the Screen Sessions page listing live-watch runs with their captures and detected issues (top left), the dashboard with quick actions, usage cards, and AI modes (top right), and the Screen Awareness view with the live terminal capture and the detected critical issue with its fix (bottom).

Unit tests checked individual functions, listed in Table III. *Integration tests* followed the full chain from screen capture through the OCR engine, context analyzer, issue classifier, solution generator, and notification manager to the interface. They checked how data and errors pass between modules and how the system behaves when an earlier module returns nothing or only part of the expected output. *Functional tests* covered login and registration, the display of issues, accepting or ignoring a solution, history, notifications, and feedback. *System tests* validated the whole workflow, including performance, stability, security, compatibility (Chrome, Edge, and Windows 10/11), and usability. Boundary cases included empty OCR output, an unsupported screen resolution, a network timeout, an invalid login, a duplicate session, and an oversized capture file.

A. Feature Performance and Accuracy

Dashboard. Figure 2 shows the home screen. A sidebar on the left groups the tools into Workspace, Live Capture, and Intelligence sections. The main area greets the user and offers quick actions for a new conversation, voice mode, screen capture, and the knowledge base. Three cards report the number of conversations, messages, and stored documents, and a row of AI modes (standard, coding help, writing assistant, and screen awareness) lets the user choose how the assistant behaves. The screen also carries a local-first label and a command palette.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

TABLE III
UNIT TEST CASES

Function	Input	Expected output
`validateUserInput`	Empty email	Validation error
`parseOCRResult`	Noisy OCR text	Cleaned, structured text
`extractUIElements`	Screenshot bitmap	Elements with coordinates
`parseContextData`	Raw session object	Structured context model
`classifyIssue`	Context and elements	Issue category
`mapSeverityLevel`	Issue type	LOW / MED / HIGH
`formatSolution`	Raw AI response	Solution card object
`applyNotificationRule`	Severity and preferences	Correct trigger
`saveSessionRecord`	Completed session	Persisted, ID returned

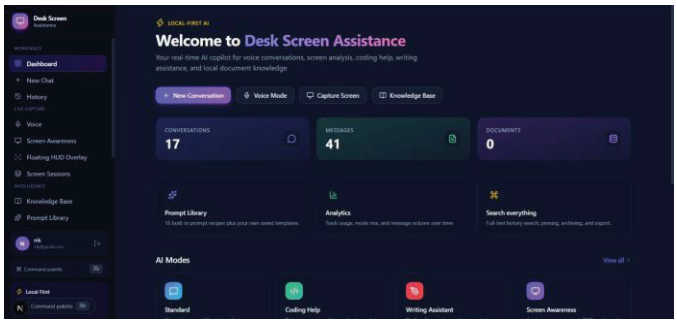


Fig. 2. Dashboard of the Desk Screen Assistance application: the sidebar with Workspace, Live Capture, and Intelligence sections, the quick actions, the usage cards, and the AI modes.

Screen Awareness. In a representative case, the user was working in a terminal that printed a `Cannot find module` error because `package.json` was missing. The bottom of Fig. 1 shows the live capture on the left, with a status line reporting that frames with no meaningful change are skipped. On the right, the live feed lists the detected issue, which the system classified as critical in a command-prompt context. It titled the issue "Missing package.json and module not found," explained the root cause, and gave numbered steps and a ready-to-copy fix, `npm init -y` followed by `npm install`. The user can mark the fix as helpful or not helpful, or pick a suggested follow-up question.

Issue Card and Windows Overlay. Figure 3 shows the same diagnosis in two forms. The issue card (top) names the error type (FILENOTFOUNDERROR), carries a confidence value of 90 %, describes the problem in one line, and shows the recommended fix with a button that copies the command. The floating overlay (bottom) presents the same result in a compact window with Diagnose, Chat, and Settings tabs. Its "Diagnose Active Screen" button is bound to `Alt+Shift+S`, and a live screen monitor toggle sits below it.

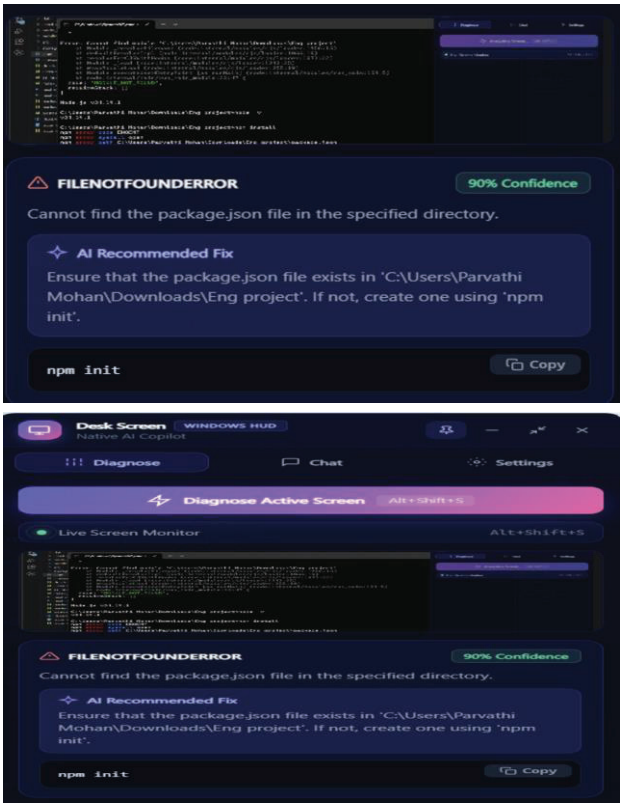


Fig. 3. Diagnosis of the missing `package.json` error: (top) the issue card with the error type, the 90 % confidence value, a one-line description, and the copyable fix, and (bottom) the floating Windows overlay with the Diagnose, Chat, and Settings tabs.

Session History. Every live-watch run is stored as a session. The Screen Sessions page (top left of Fig. 1) shows the totals (17 sessions, 24 captures, and 17 detected issues) and lists each session with the application it watched (for example Command Prompt, Visual Studio Code, and OpenRouter), its duration, its captures, and its issues. A session that is still running is marked as open.

Prompt Library. Figure 4 shows the prompt library. A search box and category filters (all, coding, general, interview, meetings, screen, and writing) help the user find a template, and a button creates a new one. Each card, such as explaining an error, code review, writing unit tests, refactoring, building a regular expression, or summarizing a meeting, has a Use button that runs the template with the user's input and a star for marking favorites.

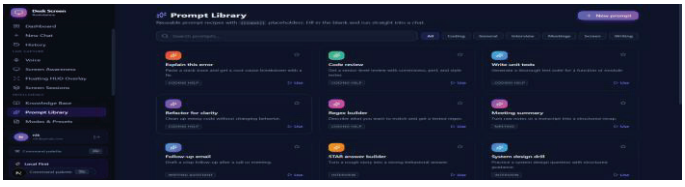


Fig. 4. Prompt Library: the search box, the category filters, and reusable prompt cards, each with a Use button.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

B. Performance Evaluation

The evaluation covers six aspects. OCR quality is measured as character and word error rate on terminal, editor, and web-page screenshots whose text is known in advance. Issue detection is scored with precision, recall, and F1 over the five issue classes. Solution quality is the share of generated suggestions that actually fix the problem. Latency is timed for each stage and end to end, and reported as a mean and a 95th percentile against the target of a response in under two seconds. Resource use covers CPU, memory, and network at idle and under load. Usability compares task time and user satisfaction with a manual-search baseline. Table II reports the measured values.

Two factors matter most for quality: how well OCR copes with cluttered screens, and how long the LLM call takes, since that largely determines how responsive the system feels. The results should be read with some caution. The test data were synthetic, the screens were in English only, and the system depends on an external LLM.

C. Security and Privacy Considerations

- Capture starts only after explicit user consent, privacy filters run before capture, and the user can pause capture at any time.
- Stored user data is encrypted, and login uses token-based authentication with account lockout after repeated failures.
- Because the captured context is sent to an external LLM service for analysis, users should be told clearly what is transmitted.

D. Limitations

- **OCR noise:** low-resolution or cluttered screens reduce text quality. Pre-processing helps, but it does not remove the problem.
- **Ambiguous context:** when several applications are open, it is hard to tell which task the user is actually working on.
- **Response delay:** calls to the LLM can slow down a system that is meant to feel immediate.
- **Domain limits:** a general-purpose LLM can be wrong on specialized topics, and the commands it suggests are not tested in a sandbox before the user runs them.
- **Platform scope:** we tested on Windows 10 and 11 only.

V. CONCLUSION AND FUTURE SCOPE

We presented a screen assistance system that captures the screen, extracts text and interface elements, classifies the issues it finds, and generates practical suggestions through an LLM. It is backed by a knowledge base that learns from user feedback. With screen awareness, severity-rated issue detection, a floating overlay, a prompt library, and a searchable session history, the assistant gives both general users and developers proactive help. The prototype shows that screen-aware assistance is feasible on an ordinary Windows desktop.

The system works as designed, but several improvements are planned. Users should be able to switch capture on and off and to exclude specific applications. Alert thresholds need to be calibrated on real data, and support should extend to more applications, operating systems, and multi-monitor setups. On-device processing with a small local model would improve privacy, and deep-learning-based OCR could raise accuracy on handwritten and low-resolution content. We also plan adaptive context learning, personalized workflow learning, and a safe way to verify suggested actions before they run.

ACKNOWLEDGMENT

The authors thank their project guide, the project coordinators, the Head of the Department, and the institution for their support.

REFERENCES

[1] A. Kumar, “Review paper on JARVIS: A desktop assistant,” *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 13, no. 2, pp. 1–8, 2025.
[2] A. Bhange, A. Virutkar, S. Bijagare, T. Nemad, and S. Nandekar, “An approach towards to real time AI desktop voice assistant,” *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 4, no. 2, pp. 156–162, 2024.
[3] F. Rakotomalala, H. N. Randriatsarafara, A. R. Hajalalaina, and N. M. V. Ravonimanantsoa, “Voice user interface: Literature review, challenges and future directions,” *Syst. Theory, Control Comput. J.*, vol. 1, no. 2, pp. 1–15, 2021.
[4] S. Rahul, P. Santhosh, B. S. Rajvignesh, and R. S. Ponmagal, “Design and development of smart AI-powered desktop assistant for productivity enhancement,” in *Proc. 2025 Int. Conf. Adv. Power, Signal, Inf. Technol. (APSIT)*, 2025, pp. 1–8, doi: 10.1109/APSIT63993.2025.11086150.
[5] R. Talmale, I. Jaiswal, A. Bamrah, J. Saggu, and T. Patle, “A comprehensive survey on AI-powered desktop automation (COSMO),” *i-manager’s J. Augmented Virtual Reality*, vol. 4, no. 1, pp. 1–12, 2026.
[6] S. O. Singh and K. Yadav, “Browser-based AI assistants,” *Int. J. Sci. Res. Eng. Manag.*, vol. 9, no. 2, pp. 1–10, 2025.
[7] V. Kirkire, “Smart chef: A comprehensive survey on AI-powered kitchen assistant systems for recipe management and intelligent cooking guidance,” *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 13, no. 1, pp. 1–12, 2025.
[8] S. N. Landge, “ChatMyDoc: AI powered document interaction assistant,” *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 13, no. 3, pp. 1–10, 2025.
[9] K. Malarvizhi, R. Vijaykumar, B. K. Pavishnu, P. Hariharan, and A. Kavin, “Smart sign language interpreter for text and speech conversion,” *Int. J. Sci. Res. Eng. Manag.*, vol. 9, no. 1, pp. 1–8, 2025.
[10] O. Korn, “Context-aware assistive systems for augmented work: A framework using gamification and projection,” Univ. Stuttgart, 2014.
[11] Y. Sun, N. J. Yuan, X. Xie, K. McDonald, and R. Zhang, “Collaborative intent prediction with real-time contextual data,” *ACM Trans. Inf. Syst.*, vol. 35, no. 4, pp. 1–33, 2017.
[12] K. D, A. Raj, S. Singh, V. Khandelwal, and M. Jindal, “Context-aware AI assistant using Open AI,” in *Proc. 2025 Int. Conf. Innovative Trends Inf. Technol. (ICITIIT)*, 2025, pp. 1–6.
[13] W. Chen, C. Yu, H. Wang, Z. Wang, L. Yang, Y. Wang, W. Shi, and Y. Shi, “From gap to synergy: Enhancing contextual understanding through human-machine collaboration in personalized systems,” in *Proc. 36th Annu. ACM Symp. User Interface Softw. Technol. (UIST)*, 2023, pp. 1–14.
[14] M. N. Sakib, “An interdisciplinary review of commonsense reasoning and intent detection,” arXiv:2501.02156, 2025.
[15] N. Sultanum, C. Murad, and D. J. Wigdor, “Understanding and supporting academic literature review workflows with LitSense,” in *Proc. 2020 Int. Working Conf. Adv. Vis. Interfaces*, 2020, pp. 1–9.
[16] Y. Hu, J. Tang, X. Gong, Z. Zhou, S. Zhang, D. S. Elvitigala, F. Mueller, W. Hu, and A. J. Quigley, “Vision-based multimodal interfaces: A survey and taxonomy for enhanced context-aware system design,” in *Proc. 2025 CHI Conf. Human Factors Comput. Syst.*, 2025, pp. 1–25.
[17] M. J. S. Lazaro, S. H. Kim, J. Y. Lee, J. Chun, G. Kim, E. Yang, A. Bilyalova, and M. Yun, “A review of multimodal interaction in intelligent systems,” *Interaccio´n*, vol. 2, no. 1, pp. 1–15, 2021.
[18] K. Haruna, M. Ismail, S. Suhendroyono, D. Damiasih, A. Pierewan, H. Chiroma, and T. Herawan, “Context-aware recommender system: A review of recent developmental process and future research direction,” *Appl. Sci.*, vol. 7, no. 10, Art. no. 1016, 2017.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

[19] H. U. R. Khan, C. Lim, and S. L. Wang, "Contextual suggestion and recommendation systems: A review on challenges in user modeling and privacy concern," *J. ICT Educ.*, vol. 8, no. 1, pp. 1–20, 2021.
[20] M. Bakhshizadeh, "Supporting knowledge workers through personal information assistance with context-aware recommender systems," in *Proc. 18th ACM Conf. Recommender Syst.*, 2024, pp. 1–6.
[21] M. Aliannejadi, H. Zamani, F. Crestani, and W. B. Croft, "Context-aware target apps selection and recommendation for enhancing personal mobile assistants," *ACM Trans. Inf. Syst.*, vol. 39, no. 4, pp. 1–30, 2021.
[22] V. Lai, C. Chen, A. Smith-Renner, Q. Liao, and C. Tan, "Towards a science of human-AI decision making: An overview of design space in empirical human-subject studies," in *Proc. 2023 Conf. Fairness, Accountability, Transparency*, 2023, pp. 1369–1388.
[23] Y. Lu, Y. Yang, Q. Zhao, C. Zhang, and T. J. Li, "AI assistance for UX: A literature review through human-centered AI," arXiv:2404.09799, 2024.
[24] M. Lee *et al.*, "A design space for intelligent and interactive writing assistants," in *Proc. 2024 CHI Conf. Human Factors Comput. Syst.*, 2024, pp. 1–26.
[25] K. Ng, C. H. Chen, C. K. M. Lee, R. Jiao, and Z. X. Yang, "A systematic literature review on intelligent automation: Aligning concepts from theory, practice, and future perspectives," *Adv. Eng. Informat.*, vol. 47, Art. no. 101246, 2021.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)