IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511149

AlgoVision: AI-Assisted Interactive Algorithm Visualization Platform

Sharon Drisilda[1], B Subash[2], Karthik[3]

Assistant Professor, Dept. of Information Science & Engineering, East West Institute of Technology, Bengaluru, Karnataka, India[1]

Dept. of Information Science & Engineering, East West Institute of Technology, Bengaluru, Karnataka, India[2]

Dept. of Information Science & Engineering, East West Institute of Technology, Bengaluru, Karnataka, India[3]

Abstract: This paper presents AlgoVision, an implemented web platform that combines interactive visualization, structured concept notes and retrieval-augmented tutoring across Data Structures and Algorithms (DSA), Operating Systems (OS), Computer Networks (CN) and Database Management Systems (DBMS). The platform provides 41 visualizations through a shared trace-first simulation engine: topic-specific generators precompute ordered execution states, and a common controller supports playback, reverse stepping, scrubbing and speed adjustment. Each topic includes a consistent eight-section notes panel. The AI tutor uses local all-MiniLM-L6-v2 embeddings, MongoDB Atlas Vector Search and a Groq-hosted language model, with the active topic and step supplied as context. JWT authentication, bcrypt password hashing and per-user progress records support account-based learning. Functional verification scenarios cover visualization, playback, authentication, progress tracking, tutor behaviour and deployment; controlled evaluation of learning outcomes, retrieval quality and usability has not been performed and is proposed as future work.

Keywords: Algorithm visualization, trace-first simulation, computer science education, retrieval-augmented generation, AI-assisted tutoring, vector search.

I. INTRODUCTION

Core computer science subjects describe systems that evolve through sequences of operations. In DSA, values are compared, exchanged, inserted or traversed; in OS, processes change state while scheduling, memory allocation and page replacement update shared resources; in CN, protocols exchange messages through ordered stages; and in DBMS, queries, indexes, transactions and locks govern the movement and consistency of data. In each case, understanding the final output is not sufficient; learners must understand how the system reaches it.

Slides, textbooks and recorded demonstrations typically show selected states and leave the omitted transitions to be inferred. Interactive visualization makes these transitions observable and lets learners control the pace of inspection; the educational effectiveness of algorithm visualization has been studied empirically [1]. In practice, however, a learner may use one tool for sorting, separate notes for scheduling, a video for protocols and an unrelated chatbot for questions. The representations and controls change across resources, and a general-purpose chatbot does not know which topic or execution step is currently visible.

AlgoVision combines these activities in one browser-based environment. Its contributions are engineering contributions: (1) a trace-first simulation model in which each topic precomputes a sequence of states that a shared controller navigates, used for 41 visualizations in four domains; (2) a tutor that retrieves topic knowledge using the active topic and step as context; (3) a common eight-section notes format with account-based progress records; and (4) a verification and evaluation plan that separates software verification from educational-effectiveness claims. We do not claim novelty for the individual components.

II. RELATED WORK

A. Algorithm visualization. Hundhausen et al. analysed 24 controlled experiments and concluded that how learners use a visualization matters more than how it looks, with active engagement outperforming passive viewing [1]. VisuAlgo is an established web resource for visualizing data structures and algorithms, concentrated on DSA, that also provides e-lecture material and an online quiz system [2]. VisualCodeMOOC integrates a conversational agent with dynamic algorithm and data-structure visualization in a course platform and reports an evaluation of usability and educational value [3]. AlgoVision's implemented scope is broader—OS, CN and DBMS topics are served through the same interface

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

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511149

as DSA—and its tutor receives the execution step currently shown. Other visualizer papers address single areas such as machine-learning or graph algorithms, and a separate paper also uses the title "AlgoVision" for a platform combining algorithm visualization with company-specific interview preparation; the present work was developed independently and targets a unified multi-domain platform for the core curriculum.

B. AI tutors and retrieval-augmented generation. Large language models have been reviewed as virtual tutors for generating and evaluating educational material [4]. Fluent responses do not by themselves ensure alignment with a particular course implementation or with the state shown in a visualization. Retrieval-augmented generation (RAG) combines a pre-trained generator with a non-parametric knowledge source [5]; KITE applies RAG to algorithm tracing and problem solving and evaluates it with automated metrics, expert judgement and simulated students [6]. AlgoVision uses retrieval to supply topic-specific material and step context before generation, and treats retrieval as a grounding aid, not a guarantee of correctness.

C. Sentence embeddings and vector search. Sentence-BERT introduced sentence embeddings suitable for semantic similarity and retrieval [7]. The all-MiniLM-L6-v2 model maps text to a 384-dimensional vector space [8], and MongoDB Atlas Vector Search, accessed through Mongoose, supports cosine similarity over stored vectors [9]. AlgoVision generates embeddings locally in the backend rather than calling a hosted embedding endpoint.

III. PROBLEM STATEMENT AND OBJECTIVES

Learners find it difficult to track intermediate states in algorithms, process scheduling, memory management, network communication and database operations. Existing workflows require switching between independent resources, and a question raised at a particular execution step must be manually re-described to a separate question-answering tool. A unified system is therefore needed that supports observable execution, reversible learner-controlled playback, structured explanations and context-aware doubt clarification across core subjects.

The objectives are to: (1) implement interactive visualizations for DSA, OS, CN and DBMS; (2) provide a shared trace-first engine with consistent playback controls; (3) support custom input and visible execution state where applicable; (4) provide a common eight-section notes format; (5) integrate a RAG tutor using topic and active-step context; (6) support authentication and user-specific progress records; and (7) deploy the frontend and backend as independently managed cloud services.

IV. SYSTEM ARCHITECTURE

A. Layered architecture. AlgoVision has a browser presentation layer, an application/API layer, a persistence layer and an AI layer (Fig. 1). The presentation layer uses React 19 and Vite and contains subject pages, topic pages, the visualization shell, notes, comparison mode and the tutor interface. The application layer uses Node.js and Express REST APIs for authentication, progress and tutor requests. MongoDB Atlas stores user accounts, progress records and vectorized knowledge chunks. The AI layer performs local embedding generation, vector retrieval and language-model inference through Groq.

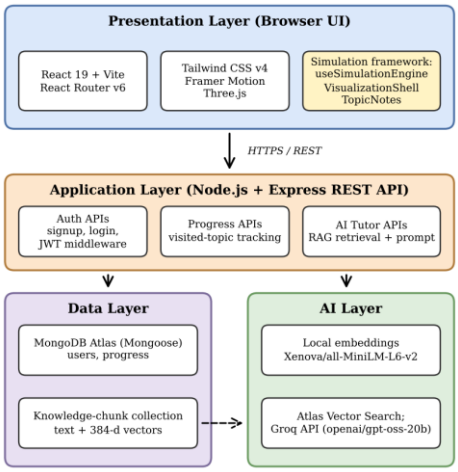


Fig. 1 Layered architecture of the AlgoVision platform.

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

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511149

The visualization layer is independent of the AI tutor: a learner can select a topic, enter input, generate a trace and use playback without a successful AI request. Tutor requests reach the backend only when the learner asks a question, and the backend holds the database credentials and model-service secrets. This separation limits the effect of an external-service failure on the deterministic simulation experience.

B. Trace-first visualization pipeline. When a learner selects a topic and supplies input, a topic-specific generator validates the input and computes an ordered sequence $S = [s_0, s_1, \ldots, s_n]$. Each snapshot s_i stores what is needed to render one state: values, active indices or pointers, relevant metrics and an explanation of the event. The playback controller maintains a current index i; rendering is determined by S[i], and playback actions change i rather than recomputing the algorithm. The shared useSimulationEngine hook manages the current step, play/pause state, speed (0.5× to 2×) and scrubbing; previous and next operations clamp i to the trace bounds. Because the trace is precomputed, forward movement, backward movement and direct selection of a step use the same state representation, and trace-generation correctness can be tested separately from the playback controller.

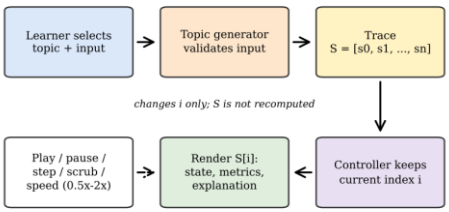


Fig. 2 Trace-first visualization pipeline.

C. AI tutor pipeline. The RAG pipeline has an offline ingestion path and an online query path. During ingestion, topic content is divided into knowledge chunks: DSA chunks are based on implementation source code, whereas OS, CN and DBMS chunks use structured teaching-note content. The backend uses the @huggingface/transformers library [10] with Xenova/all-MiniLM-L6-v2 to generate 384-dimensional embeddings, which are stored with chunk text and metadata in a dedicated Atlas collection indexed for vector search. At query time, the learner's question is combined with the active topic and step summary and embedded with the same model; Atlas Vector Search retrieves the most relevant chunks by cosine similarity, $\text{sim}(q, d) = (q \cdot d) / (\|q\| \|d\|)$; and the retrieved chunks, system instructions and step information are assembled into a grounded prompt for the Groq-hosted openai/gpt-oss-20b model [11]. If the AI service fails, a friendly failure response is returned and the visualization remains available. Because all-MiniLM-L6-v2 truncates long inputs at 256 word pieces [8], chunk length must be bounded; chunk-length statistics are not reported in this paper. Ingestion is run separately from the online flow, so updating the source notes or implementation material requires re-running ingestion for the affected chunks.

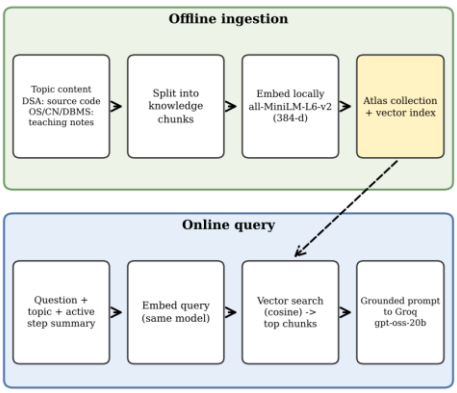


Fig. 3 Offline ingestion and online query paths of the AI tutor.

D. Data and security design. Three logical collections are used: Users, Progress and Knowledge Chunks. User records contain account information and a bcryptjs password hash (10 salt rounds). Progress records hold the topics visited by each learner and are updated with MongoDB's atomic $addToSet operator, which prevents duplicate entries when a topic is revisited. Knowledge chunks hold source text, topic/domain metadata and embedding vectors, kept separate from user-specific data. JWTs with a seven-day expiry are issued on signup and login, and protected routes verify the token in

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

DOI: 10.17148/IJARCCE.2026.1511149

middleware. Invalid-email and invalid-password conditions return identical messages so that account existence is not revealed. API keys and database credentials are held in server-side environment variables. Non-functional goals—performance through precomputed traces, security through hashing and server-side secrets, extensibility through metadata-driven routing and availability through independence from the AI service—are design provisions, not measured service-level guarantees.

V. IMPLEMENTATION

A. Learner workflow. The learner selects one of four subject domains and a topic on a common topic page, then supplies the input required by the visualizer (for example an array, graph nodes, process parameters, packet information or database input). Topic-specific validation runs before trace generation, and invalid input produces a validation message. The first snapshot is rendered and the learner can play, pause, step, change speed or scrub; the active state, metrics, legend and explanation are shown together. When the learner asks a question, the browser sends it with the topic and active-step context to the tutor API, and the response is shown beside the visualization. For authenticated learners, a visited topic is recorded in the progress collection.

B. Domain modules. Table I summarizes coverage. The DSA module contains Bubble, Selection, Insertion, Merge and Quick Sort; Linear and Binary Search; Linked List (insert, delete, traverse, reverse); Stack and Queue; Binary Search Tree (insert, search, traversals); and BFS and DFS. The OS module contains Round Robin; Process States; FCFS; SJF/SRTF; Priority Scheduling; FIFO and LRU Page Replacement; Memory Allocation (First, Best and Worst Fit); Banker's Algorithm; and Disk Scheduling (FCFS, SSTF, SCAN, C-SCAN). The CN module contains the TCP 3-Way Handshake; OSI Model; TCP/IP Model; IP Addressing; ARP; DNS Resolution; HTTP/HTTPS; Packet Routing; TCP Congestion Control; and Subnetting. The DBMS module contains SQL JOINs; SQL Queries and CRUD; ER Model; Keys and Constraints; Normalization (1NF–3NF); Indexing; B+ Tree; Transactions and ACID; Concurrency Control; and Deadlocks.

TABLE I VISUALIZATION COVERAGE BY DOMAIN

Domain	No. of topics
DSA	11
OS	10
CN	10
DBMS	10
Total	**41**

C. Structured concept notes. Each topic has a TopicNotes panel with the same eight sections: (1) what is this (definition); (2) why it is needed; (3) how it works in plain language; (4) what to observe; (5) the final outcome; (6) important points and edge cases; (7) complexity or performance behaviour; and (8) real-world relevance. The format is intended to pair the visual representation with verbal explanation, in line with dual-coding theory [12].

D. Comparison mode and themes. The sorting comparison page runs two algorithms on the same input with independent step counters and a shared play/pause control; independent counters are intentional because forcing both algorithms into lockstep would obscure differences in execution progress. A light/dark theme follows the system preference on first visit and persists the learner's choice, and the common shell provides a full-screen presentation mode. Subject and topic pages are generic components parameterized by topic metadata and a component lookup table, so a new visualizer can be added by supplying metadata and a component that follows the shared rendering contract.

E. Technology and deployment. Table III summarizes the configuration. The frontend uses React 19, Vite, Tailwind CSS v4, Framer Motion, Three.js and React Router; the backend uses Node.js and Express, with MongoDB Atlas accessed through Mongoose and embeddings generated locally. The frontend is deployed on Vercel and the backend on Render, both connected to the same Atlas cluster, with automatic rebuilds on pushes to the main branch. The whole stack runs on free tiers (Vercel, Render, MongoDB Atlas M0, local embeddings and the Groq API under its free usage), which keeps the platform reproducible without recurring cost. Deployment required adapting to filename-casing differences between Windows development and Linux builds, configuring Atlas network access for dynamic IP addresses, migrating to a recommended model after a model deprecation (the model name is kept in an environment variable), and rendering the login modal through a React portal to avoid a containing-block problem caused by backdrop blur.

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

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511149

TABLE III KEY CONFIGURATION OF THE IMPLEMENTED SYSTEM

Component	Configuration
Frontend	React 19, Vite, Tailwind CSS v4, Framer Motion, Three.js, React Router
Backend	Node.js, Express REST APIs, Mongoose
Database	MongoDB Atlas: Users, Progress, Knowledge Chunks
Embeddings	Xenova/all-MiniLM-L6-v2 via @huggingface/transformers, run locally; 384 dimensions
Retrieval	Atlas Vector Search, cosine similarity
Language model	Groq-hosted openai/gpt-oss-20b; name held in an environment variable
Authentication	JWT (seven-day expiry); bcrypt password hashes
Deployment	Frontend on Vercel; backend on Render; shared Atlas cluster

F. Illustrative examples. For the input array [5, 3, 8, 1], the Bubble Sort generator records one snapshot for every comparison and swap; Table IV lists the first six. If the learner scrubs to s_2 and asks why 5 and 3 were swapped, the browser sends the question with the topic name and step state, the nearest knowledge chunks (for example the swap condition in the implementation) are retrieved and added to the prompt, and the reply can refer to the comparison shown rather than give a generic definition of sorting. Both examples illustrate the data flow and are not measurements.

TABLE IV ILLUSTRATIVE BUBBLE SORT TRACE

Step	Array	Active	Event
s_0	[5, 3, 8, 1]	–	Initial array
s_1	[5, 3, 8, 1]	0, 1	Compare 5 and 3
s_2	[3, 5, 8, 1]	0, 1	5 > 3, swap
s_3	[3, 5, 8, 1]	1, 2	Compare 5 and 8; no swap
s_4	[3, 5, 8, 1]	2, 3	Compare 8 and 1
s_5	[3, 5, 1, 8]	2, 3	8 > 1, swap; 8 in final place

VI. FUNCTIONAL VERIFICATION

A. Strategy. Four verification concerns are distinguished: visualization correctness, retrieval relevance, response quality and learning effectiveness. Trace generation can be checked with topic-specific invariants (a sorted final state for sorting, agreement between the target-found condition and the input for search, and policy-consistent state transitions for scheduling), and playback can be tested with fixed traces by checking that the displayed state equals the selected snapshot.

B. Implementation status. The frontend core and shell, unified simulation engine, theme system, all four visualization domains, JWT authentication and progress tracking, RAG pipeline and AI tutor, and cloud deployment are implemented. The 41 topics share one simulation engine and shell, and the deployed frontend and backend run independently and share the database. Two items are incomplete: re-ingesting the DSA knowledge base from the newer concept notes, and wiring progress tracking consistently from OS, CN and DBMS topic pages.

C. Functional test scenarios. Table II lists the representative scenarios and their expected outcomes. These are verification criteria for functional checks; this paper reports no pass rate, accuracy score, latency measurement or learning gain.

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

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511149

TABLE II FUNCTIONAL TEST SCENARIOS AND EXPECTED OUTCOMES

ID	Scenario	Expected outcome	Category
T1	Sorting visualizer on a valid array	Final trace state sorted; each step matches its invariant	Visualization
T2	Empty, single-element and duplicate input	No crash; trace and final state correct	Visualization
T3	Invalid input	Validation message; no trace generated	Input
T4	Step forward and backward	Displayed state equals selected snapshot	Playback
T5	Scrub to a step; change speed	Instant jump; 0.5×–2× respected	Playback
T6	Open each of the 41 topics	Shell, controls and eight-section notes appear	Navigation
T7	Switch light/dark theme	All pages restyle; choice persists	Theme
T8	Sign up and log in	JWT issued; password stored only as a hash	Security
T9	Wrong email vs. wrong password	Identical error message	Security
T10	Visit a topic when logged in	Topic added once to progress	Progress
T11	Ask about a known topic	Terminology consistent with retrieved knowledge	AI tutor
T12	Same question at another step	Explanation reflects the active step	AI tutor
T13	Question with no matching knowledge	No unsupported implementation details presented as retrieved facts	Retrieval
T14	AI service unavailable	Visualizations and notes still work	Reliability
T15	Push to main branch	Frontend and backend rebuild and deploy automatically	Deployment

VII. DISCUSSION AND LIMITATIONS

A. Design rationale. A live animation that computes each operation as it renders makes step-back difficult, because earlier states may no longer exist and would have to be reconstructed by replaying operations or by writing reverse operations for every topic. Storing snapshots costs memory for the selected input but makes every state directly addressable by index. Heterogeneous topics (an array, a scheduler, a packet exchange, a database index) cannot share drawing logic, so topic-specific rendering is separated from the shared interaction behaviour provided by the shell and hook. Comparison mode is a deliberate exception to the unified engine because it needs two independent step counters.

B. RAG grounding. Supplying the active step gives retrieval a more specific query than the question alone, and retrieved material can keep terminology aligned with platform content. However, retrieval depends on the coverage and quality of the knowledge chunks, and the model may still produce an incorrect explanation; a retrieved passage is context, not proof of correctness.

C. Limitations. (i) The tutor depends on an external language-model service, so availability and latency may vary. (ii) DSA knowledge entries were created from source-code chunks and have not yet been re-ingested from the newer concept notes, so grounding depth differs across domains. (iii) Progress tracking from OS, CN and DBMS pages is not yet consistent. (iv) No controlled educational study, retrieval evaluation or latency measurement has been conducted. (v) The platform is a supplementary aid and does not replace textbooks, laboratories or instruction.

D. Intended use. In a classroom or laboratory, an instructor can pause at a comparison, scheduling decision, handshake stage or transaction event to discuss it, and learners can use the same controls for self-paced revision and read the

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

DOI: 10.17148/IJARCCE.2026.1511149

matching concept notes. Custom-input visualizations can also support revision of edge cases. These are intended applications of the implemented functionality, not measured educational outcomes.

VIII. EVALUATION PLAN AND FUTURE WORK

A. Technical evaluation. Automated test sets can be built for each trace generator: ordering and multiset preservation for sorting, comparison with a reference implementation for searching, and transition, waiting-time and completion checks for scheduling; networking and DBMS topics can use expected state transitions for representative inputs. Playback tests can use fixed traces to verify boundary clamping, arbitrary scrubbing and index–render consistency.

B. Retrieval and response evaluation. A labelled set of representative questions and relevant chunks is needed to compute Precision@K, Recall@K and mean reciprocal rank. Generated answers should be assessed separately for answering the question, reflecting the active step, using retrieved material appropriately and avoiding unsupported statements. Chunk-length statistics should be reported against the 256-word-piece limit of the embedding model [8].

C. Learning-effectiveness study. A controlled study could compare a baseline learning condition with an AlgoVision-supported condition using comparable pre- and post-tests plus a usability questionnaire. Sample size, participants, instruments, consent procedures and analysis methods must be specified before the study. Until it is completed, no learning-gain claim is made.

D. Planned enhancements. Re-ingesting DSA knowledge from the updated notes; completing progress tracking for OS, CN and DBMS; Big-O complexity panels and growth curves; a pseudocode panel synchronized with the current step; adaptive tutoring that adjusts depth for repeated or rephrased questions; additional topics; and automated trace-invariant and retrieval-relevance tests. These are planned and are not presented as completed features. Future experiments should record application version, topic, input, trace length, device conditions, model identifier, retrieval configuration, question, active step, retrieved chunks and generated answer so that retrieval errors can be distinguished from generation errors.

IX. CONCLUSION

AlgoVision is an implemented learning platform with 41 visualizations across DSA, OS, CN and DBMS. A shared trace-first engine provides predictable playback and step-level inspection, and a reusable shell, eight-section notes, comparison mode and themes give a consistent interface. The RAG tutor combines local sentence embeddings, Atlas Vector Search and a Groq-hosted model to produce responses informed by the learner's topic and step, and JWT authentication, bcrypt hashing, progress records and cloud deployment complete the system. The evidence presented establishes implementation and functional-verification scenarios, not educational impact; a controlled usability and learning-effectiveness study is required.

ACKNOWLEDGMENT

The authors acknowledge the support and academic facilities provided by the Department of Information Science and Engineering, East West Institute of Technology, Bengaluru.

DECLARATIONS

AI-use disclosure. This manuscript was language-edited and formatted with the assistance of an AI tool (Claude, Anthropic). The authors reviewed the content and take full responsibility for it.

Conflict of interest. The authors declare no known competing financial interests or personal relationships that could have influenced the work reported in this paper.

Ethics and consent. No human-subject study is reported; informed-consent provisions do not apply to the work described.

Data availability. No experimental datasets are reported. The project source code and selected functional-test materials are partially available from the authors on reasonable request, subject to repository access and security restrictions; secrets, credentials and private deployment configuration are excluded.

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



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026
DOI: 10.17148/IJARCCE.2026.1511149

REFERENCES

[1]. . C. D. Hundhausen, S. A. Douglas, and J. T. Stasko, "A meta-study of algorithm visualization effectiveness," *J. Vis. Lang. Comput.*, vol. 13, no. 3, pp. 259–290, 2002, doi: 10.1006/S1045-926X(02)00028-9.
[2]. . S. Halim, "VisuAlgo – Visualising data structures and algorithms through animation," *Olympiad in Informatics*, vol. 9, pp. 243–245, 2015. [Online]. Available: https://visualgo.net (accessed Nov. 17, 2026).
[3]. . M. Li, D. Wang, E. Purwanto, T. Selig, Q. Zhang, and H.-N. Liang, "VisualCodeMOOC: A course platform for algorithms and data structures integrating a conversational agent for enhanced learning through dynamic visualizations," *SoftwareX*, vol. 30, Art. no. 102072, 2025, doi: 10.1016/j.softx.2025.102072.
[4]. . S. García-Méndez, F. de Arriba-Pérez, and M. del C. Somoza-López, "A review on the use of large language models as virtual tutors," *Sci. Educ.*, 2024, doi: 10.1007/s11191-024-00530-2.
[5]. . P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 9459–9474.
[6]. . M. Jain *et al.*, "Retrieval-augmented tutoring for algorithm tracing and problem-solving in AI education," in *Proc. 21st Workshop Innov. Use NLP Build. Educ. Appl. (BEA)*, 2026, arXiv:2605.12988.
[7]. . N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proc. EMNLP-IJCNLP*, Hong Kong, China, 2019, pp. 3982–3992, doi: 10.18653/v1/D19-1410.
[8]. . Sentence Transformers, "all-MiniLM-L6-v2," Hugging Face model card. [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2 (accessed Nov. 17, 2026).
[9]. . Mongoose, "Atlas Vector Search." [Online]. Available: https://mongoosejs.com/docs/atlas-vector-search.html (accessed Nov. 17, 2026).
[10]. . Hugging Face, "Transformers.js." [Online]. Available: https://github.com/huggingface/transformers.js (accessed Nov. 17, 2026).
[11]. . Groq, "OpenAI GPT-OSS 20B," GroqDocs. [Online]. Available: https://console.groq.com/docs/model/openai/gpt-oss-20b (accessed Nov. 17, 2026).
[12]. . A. Paivio, *Mental Representations: A Dual Coding Approach*. New York, NY, USA: Oxford Univ. Press, 1986.

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