International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Eduvision AI: An Offline, Syllabus-Driven System for Generating Narrated Animated Video Lessons in Computer Science

Dr. Nandish M[1], Sanjana R[1], T G Chinmay[1], Thulasi B K[1], Ullas P[1]
[1]Department of Computer Science and Engineering, Jawaharlal Nehru New College of Engineering, Shivamogga, India

Abstract **—** Revising a Computer Science topic such as Quick Sort or Binary Search is difficult with the resources students normally have: a textbook is accurate but static, generic videos online rarely match a specific syllabus, and text-based AI chatbots cannot produce a visual explanation at all. This paper presents Proposed Eduvision AI model, a system that converts the name of a Computer Science topic into a narrated, animated teaching video, generated entirely on a single laptop with no paid API, no user account and no internet connection required after installation. The system accepts a topic name, verifies that it belongs to a fixed syllabus of 286 Computer Science topics, generates a fresh numerical example for the run, uses a locally hosted large language model to draft a lesson script, subjects that script to six automated correctness checks, re-creates the algorithm's execution in code to drive the animation, converts the narration to speech, aligns every spoken word to an exact timestamp, renders the diagrams as code-generated images rather than AI-generated images, and finally assembles a subtitled MP4 video. The central design decision of the model is that no step which requires factual precision — arithmetic, counting, or text rendering inside a diagram — is delegated to a generative AI model; such steps are performed by verifiable Python code instead, while the AI model is used only for the parts of the task that are genuinely creative, namely prose. The resulting pipeline is fully reproducible, produces a different worked example on every run, and has been validated using 808 automated tests.

Keywords **—** Offline Artificial Intelligence, Automated Video Generation, Local Large Language Model, Text-to-Speech, Speech Alignment, Educational Technology, Computer Science Pedagogy.

I. INTRODUCTION

Students revising a Computer Science topic late at night effectively have three options, and each has a shortcoming. A textbook explanation is correct but static, which is a serious limitation for a topic such as a sorting or searching algorithm where the entire idea is the movement of data over time. Pre-recorded videos on public platforms are often high quality but are built around an example chosen by someone else, are not guaranteed to align with a particular institution's syllabus, and cannot be regenerated with new data for repeated practice. General-purpose AI chatbots can produce a correct textual explanation on demand, but they generate text, not a video, and therefore cannot show the reader what is actually happening to the data structure at each step.

Proposed Eduvision AI model was built to close this gap. It is a locally run program that accepts the name of a Computer Science topic — for example “Quick Sort” — and, after a short processing time, produces a two-to-three-minute narrated video in which a diagram animates in step with a spoken explanation. The system draws its topic list directly from university syllabi, so that the material generated matches what a student is actually expected to know, and it invents a new worked example on every run so that watching the same topic twice is a second lesson rather than a replay.

The motivation for the project follows directly from five practical problems observed with existing tools, summarised in Table I.

TABLE I : PROBLEMS ADDRESSED AND THE CORRESPONDING DESIGN RESPONSE

Problem with existing tools	Eduvision AI's response
Commercial video-generation and AI tools charge per use.	All models run locally; the only running cost is electricity.
Cloud-based tools require an active internet connection.	The entire pipeline, once installed, functions with the network disabled.
Generic videos do not track a specific syllabus.	286 topics are taken directly from university Computer Science syllabi.
Watching the same recorded video twice adds little value.	A fresh numerical example is generated and verified on every run.
Generative AI models can state incorrect facts with confidence.	Six independent automated checks reject any lesson whose diagram contradicts its narration.

The remainder of this paper is organised as follows. Section II reviews related approaches to automated and sensor-driven educational or monitoring systems. Section III describes the proposed system at a conceptual level. Section IV reports the results obtained from the working prototype. Section V concludes the paper and outlines future work.

II. LITERATURE SURVEY

A set of research works was reviewed to establish the design approaches available to a system such as Eduvision AI model and to examine the individual components — dialogue-based tutoring, video generation, language modelling, computer vision, and speech synthesis — from which such a system can be built.

D'Mello and Graesser [1] present AutoTutor and its affect-sensitive extension, an intelligent tutoring system that holds a spoken natural-language dialogue with a learner and adapts its responses to both the correctness of an answer and the learner's apparent emotional state. The system's tutoring strategy is built entirely on scripted conversational dialogue rather than on any generated visual or animated content. It is relevant to Eduvision AI model mainly as evidence that language-driven, adaptive tutoring improves learner engagement, which motivates pairing spoken narration with a synchronised visual channel that AutoTutor itself does not provide.

Chen, Liu and Wang [2] propose a deep-learning pipeline that converts a block of text directly into a short video, using neural sequence models to plan scenes and a generative model to render the corresponding visuals end to end. Their results show that a fully neural pipeline can automate video production and reduce manual editing effort considerably. However, the generated visuals are not checked against the source text for factual, numerical, or structural accuracy, which is precisely the limitation that Eduvision AI model's automated correctness checks are designed to close.

Brown et al. [3] introduce GPT-3, a 175-billion-parameter transformer language model that performs a wide range of language tasks from only a handful of in-context examples, without any task-specific fine-tuning. The paper establishes that scaling a transformer substantially improves the coherence and contextual appropriateness of generated prose. Eduvision AI model relies on a much smaller, locally hosted model from the same architectural family to draft each lesson's narration, though, unlike GPT-3, it never uses that model to generate the numerical or structural content of a diagram.

Vaswani et al. [4] introduce the Transformer architecture, which replaces recurrent and convolutional layers with a self-attention mechanism that weighs the relevance of every token in a sequence when producing each output token. This design trains more efficiently at scale than earlier recurrent architectures and underlies essentially every language model used in Eduvision AI model. The paper addresses only the architecture and machine-translation performance, however, and does not consider the factual reliability of a trained model's output — a gap that Eduvision AI model closes with its own validation stage rather than through the architecture itself.

He et al. [5] introduce residual (ResNet) connections, which add shortcut paths between the layers of a convolutional network so that very deep networks can be trained without the accuracy loss observed in earlier deep architectures. The resulting models achieved substantial gains in image-classification accuracy and became a standard building block of later computer-vision systems. Eduvision AI model does not classify or generate photographic images, but the residual-connection principle described in this paper underlies many of the vision components used indirectly within the broader ecosystem of pretrained AI tools.

Radford et al. [6] introduce CLIP, a model trained by contrasting a large corpus of image-text pairs so that it can score how well an arbitrary image matches an arbitrary text caption without any task-specific fine-tuning. This scoring ability underlies many later text-to-image systems, which use it to steer or filter generated images against a prompt. Eduvision AI does not use CLIP or any image-generation model, precisely because a model trained in this way has no mechanism for guaranteeing an exact object count or legible embedded text — the two properties a teaching diagram most needs.

Wang et al. [7] introduce Tacotron 2, a neural text-to-speech system that predicts a mel-spectrogram from input text using a sequence-to-sequence network and then converts that spectrogram into an audio waveform with a separate vocoder. The paper reports synthetic speech that listeners find difficult to distinguish from human recordings, establishing the two-stage text-to-speech design later adopted by many open-source voice models. Eduvision AI model's narration stage follows the same two-stage principle, using a locally run, open-source text-to-speech model to convert each scene's script into audio before it is aligned to the animation.

Rombach et al. [8] introduce latent diffusion models, which perform the iterative denoising process used in diffusion-based image generation within a compressed latent space rather than directly on pixels, substantially reducing the computation required for high-resolution image synthesis. The paper demonstrates state-of-the-art image quality at a fraction of the cost of earlier pixel-space diffusion models and underlies widely used image generators such as Stable Diffusion. The generated examples in this paper also illustrate, more clearly than most, the two failure modes central to this project's

motivation — inconsistent object counts and illegible embedded text — which is why Eduvision AI model does not use diffusion-based image generation for any diagram that must show a precise quantity or a specific label.

Taken together, the reviewed literature shows that the individual components required for an automated educational video — dialogue-based tutoring, end-to-end video generation, large language models, image classification and synthesis, and speech synthesis — already exist as separate, well-studied systems, but none of them is combined into a single pipeline that also verifies its own output. Eduvision AI model adopts the hybrid approach implied by this gap: its language model is restricted to writing the lesson script, while image-generation and vision techniques are deliberately avoided for diagram creation given their documented inability to guarantee an exact count or legible text. Narration follows a two-stage text-to-speech design, and every diagram is instead drawn by dedicated Python rendering code after the underlying algorithm has actually been executed step by step. In this way, the system combines proven ideas from tutoring, generation, and synthesis research into a single, self-checking pipeline rather than trusting any one generative model end to end.

III. PROPOSED SYSTEM

The primary objective of Proposed Eduvision AI model is to convert a single line of text — a Computer Science topic name — into a complete, narrated, animated video lesson, without any AI model being permitted to invent facts, numbers, or diagram content that cannot be independently verified. The system is organised as a fourteen-stage pipeline in which each stage receives a well-defined input, performs one task, and hands a well-defined output to the next stage. A manager process (the orchestrator) runs the pipeline from start to finish for each request, and only three of the fourteen stages — script writing, narration, and word-level timing — are performed by an AI model; every other stage, including all diagram drawing, is performed by deterministic code.

The topic name typed by the user is first checked against a verified list of 286 Computer Science topics and 739 accepted alternative names before any AI model is invoked, so that an invalid or off-syllabus request is rejected in a fraction of a second rather than after the full generation time has elapsed. Once a topic is accepted, a fresh numerical example is generated for that run using ordinary arithmetic rather than an AI model, because a language model asked to produce “a sorted array” cannot be relied upon to actually return one that is sorted, whereas an arithmetic routine cannot make that mistake.

A locally hosted language model then drafts the lesson script — the spoken narration together with a description of what the diagram should show at each point — and this draft is passed through six automated checks covering data validity, numerical range, frame-to-frame change, scene-to-scene repetition, factual consistency between narration and diagram, and layout correctness. A script that fails any check is sent back to the language model with a specific, numbered correction, for up to four attempts. The animation itself is never designed by the language model; instead, the real algorithm named in the topic (for example, quicksort or binary search) is executed step by step in code, and every state the algorithm actually passes through is recorded and used to drive the diagram, so that the animation cannot contradict the algorithm it is meant to illustrate.

The narration text is converted to speech locally, and a separate speech-recognition step is then used, unusually, in reverse: because the spoken text is already known, the model is used only to determine the exact time at which each word was spoken, so that a diagram change can be synchronised precisely to the sentence that explains it rather than shown before or after the relevant narration. The synchronised diagram frames, audio, and subtitles are finally assembled into a single video file, and a last automated audit compares every rendered frame against every other frame in the same lesson to confirm that the animation genuinely changes rather than repeating a static image.

The system follows a three-layer architecture. A browser-based front end, built with React and TypeScript, accepts the topic name from the user and displays generation progress live as each pipeline stage completes. A local web server, built with the Python framework FastAPI, receives this request, queues it, and streams progress events back to the browser. Beneath the server sits the processing pipeline itself, which calls four local AI components: qwen3:8b, an 8-billion-parameter open-weight language model served through Ollama and used solely to draft the lesson script; Kokoro-82M, an 82-million-parameter open-weight text-to-speech model that turns the finished script into a natural-sounding voice; faster-whisper, an optimised, MIT-licensed reimplementation of OpenAI's Whisper speech-recognition model, used here in reverse to time each already-known word in the narration rather than to transcribe unfamiliar audio; and the FFmpeg video-encoding program, together with the SVG-based diagram-rendering code that draws every frame. All four components run entirely on the local machine, and all jobs and their generated files are recorded in a local SQLite database so that a finished video can be replayed instantly without being regenerated. qwen3:8b was chosen over a smaller 3-billion-parameter alternative after direct comparison showed it needed fewer correction cycles and produced noticeably better-motivated explanations, despite taking longer to run on the 4 GB graphics card used for development.

A. Pipeline stages

Table II lists the fourteen stages of the pipeline in execution order, the tool responsible for each stage, and an indication of whether the stage is performed by an AI model or by deterministic code. Only three stages use an AI model directly; every diagram-related stage is deterministic.

TABLE II : PIPELINE STAGES, RESPONSIBLE COMPONENT AND STAGE TYPE

#	Stage	Responsible component	Type
1	Topic validation	Typo-tolerant matcher over 286 topics	Deterministic
2	Example generation	Python arithmetic with a random seed	Deterministic
3	Lesson planning	Local language model (Ollama)	AI-assisted
4	Automatic checking	Six rule-based validators	Deterministic
5	Choreography	Execution of the real algorithm	Deterministic
6	Narration (text-to-speech)	Kokoro speech model	AI-assisted
7	Word-level alignment	faster-whisper speech recognition	AI-assisted
8	Diagram drawing	Hand-written SVG renderers	Deterministic
9	Video assembly	FFmpeg	Deterministic
10	Render audit	Frame-hash comparison	Deterministic

Fig. 1 traces the same pipeline as it actually runs module by module, from the browser down to the finished file. Dashed boxes are the three points where a free, locally hosted AI model is used; solid boxes are ordinary Python code that the project authors wrote and can verify directly. The loop on the left shows the automatic correction cycle: a script produced by Ollama that fails one of the six Pydantic checks is sent back for up to four further attempts before the pipeline gives up and falls back to a simpler plan.

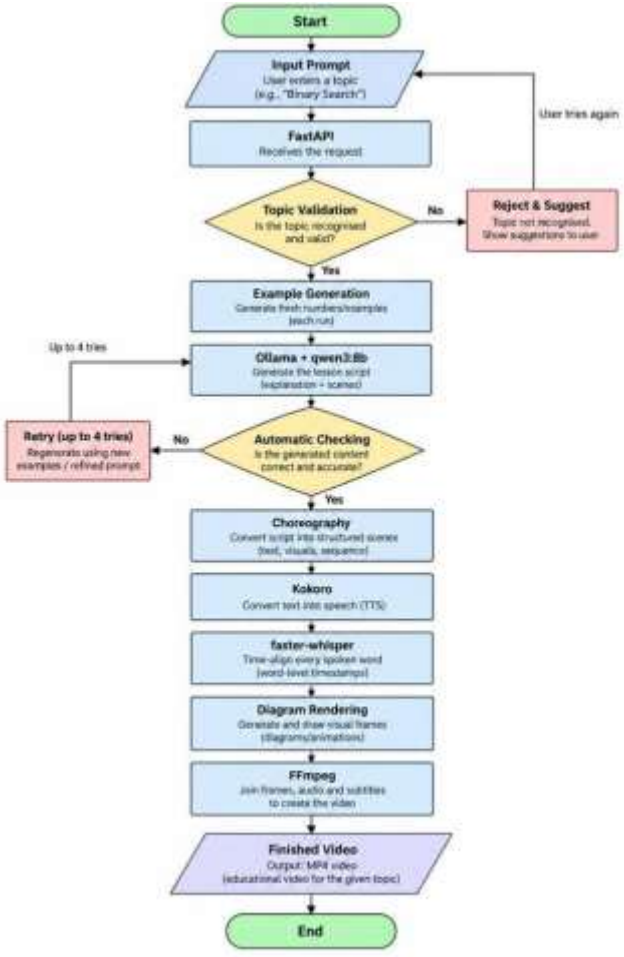


Fig. 1 EduVision AI Proposed model and its Flow Diagram

B. Why diagrams are drawn by code, not generated by AI

The most significant design decision in the system is that no diagram is produced by an image-generation model. Image-generation models are prone to two specific failures that are fatal for a teaching diagram: they frequently render an incorrect count of objects (asking for eight array cells may return six or eleven), and they cannot reliably render legible digits or letters inside a picture. Eduvision AI model avoids both failure modes by having the language model choose only a diagram type and supply the underlying data — for example, an array of eight specific integers — while a dedicated, hand-written Python function draws the picture. A function written to draw array cells can only ever draw array cells with the exact digits it was given, which removes the possibility of a miscounted or illegible diagram by construction rather than by post-hoc correction.

C. Offline operation

All four AI components used by the system run on the user's own machine: the language model is served on a local network address (localhost) by Ollama, the text-to-speech and speech-alignment models load their weights from local disk, and FFmpeg is a locally installed program. No API key, user account, or outbound network request is required once installation is complete; a diagnostic script is provided to confirm this by checking that no external network address is referenced in the built application and that all required model weights are present locally.

D. Testing

The implementation is accompanied by 808 automated tests covering topic validation, example generation, the six automatic checks, the choreography stage, diagram rendering, and audio-text alignment, allowing each stage of the pipeline to be verified independently of the others.

E. User Interface

The front end presents the user with a minimal landing page describing the system, from which the user proceeds to the generation screen. Fig. 2 shows the landing page, and Fig. 3 shows the generation screen on which the user types or selects a Computer Science topic; the system immediately indicates whether the typed topic is recognised, shows which diagram type it maps to, and lets the user choose the number of scenes before generation begins.

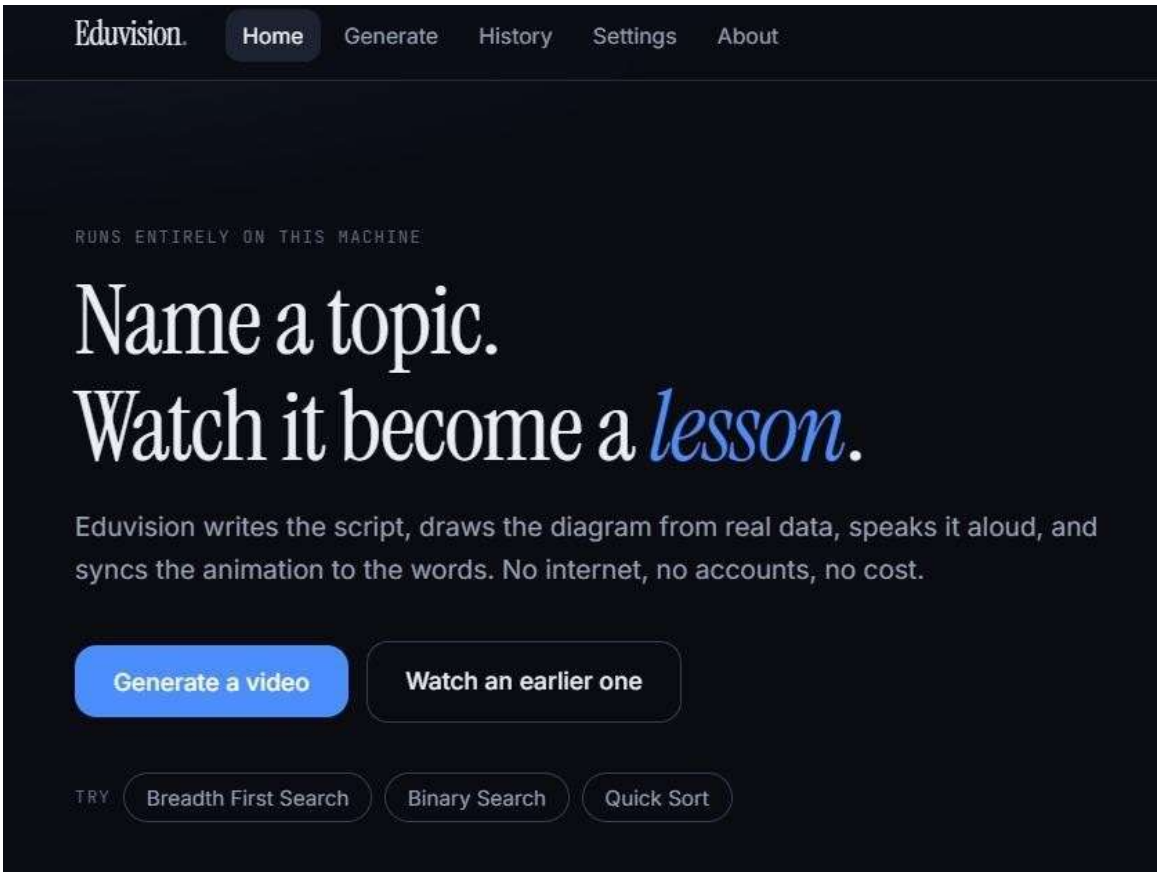


Fig. 2 Landing page of Eduvision AI

The landing page introduces the system in a single line — “Name a topic. Watch it become a lesson.” — and states plainly that it runs entirely on the local machine. Two buttons let a first-time user either start generating a video or watch an example that was generated earlier, and three ready-made suggestions (Breadth First Search, Binary Search, Quick Sort) are offered so the system can be tried without typing anything.

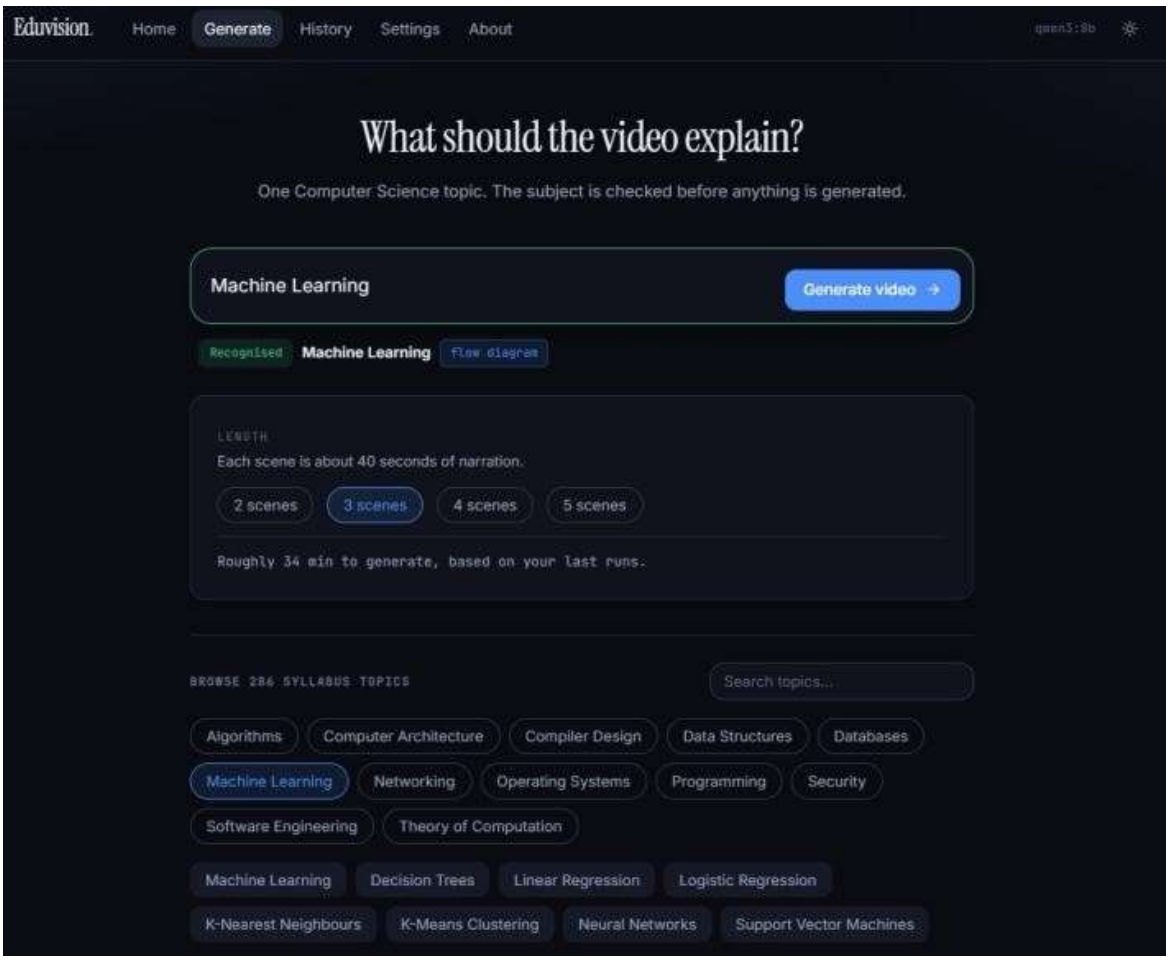


Fig. 3 Topic input and generation screen

The generation screen is where a topic is actually submitted. As the user types, the system checks the topic against the syllabus in real time and shows the result directly beneath the input box — in the example shown, "Machine Learning" is confirmed as recognised and mapped to a flow diagram. Below this, the user selects how many scenes the video should contain, with an estimated generation time shown based on previous runs, and can browse or search all 286 syllabus topics grouped by category.

IV. RESULT

Eduvision AI proposed model was implemented in Python 3.12.10, running on Windows. The backend uses FastAPI 0.115 with Pydantic 2.7 for schema validation, and the frontend was built with React 18.3.1, TypeScript 5.7.2, and Vite 6.0.5. The lesson-writing stage runs qwen3:8b through Ollama on the local machine; narration is produced by Kokoro-82M (Apache 2.0, ≈327 MB of weights, CPU inference); and word-level timing is obtained from faster-whisper (the "small" model). Diagrams are rasterised with cairosvg 2.7, and video assembly uses FFmpeg 8.1.2 (full build, gyan.dev). The system was tested on a laptop equipped with an NVIDIA GeForce RTX 3050 Laptop GPU (4 GB VRAM). Topic validation covers 286 syllabus topics with 739 aliases, and the pipeline is verified by 830 automated tests (pytest 8.0), all passing. The completed prototype was evaluated by generating video lessons for topics drawn from the target syllabus. For a representative topic such as Quick Sort, the pipeline produced a two-to-three-minute video in which the highlighted array element changed in step with the spoken narration, together with an accurate .vtt/.srt subtitle track derived from the same word-level timing data used to drive the animation. Two candidate language models were compared for the lesson-writing stage under identical prompts; the results are summarised in Table III.

TABLE III : COMPARISON OF CANDIDATE LANGUAGE MODELS FOR LESSON SCRIPT GENERATION

Model	Generation time	Attempts needed	Words per scene
Smaller 3-billion-parameter model	46 s	2	76
Larger 8-billion-parameter model	294 s	1	102

TABLE IV : TOTAL GENERATION TIME BY NUMBER OF SCENES

Scenes Selected	Approx. Total Generation Time
2 scenes	22–26 minutes
3 scenes	34–40 minutes
4 scenes	45–50 minutes
5 scenes	55–60 minutes

The larger model was selected for production use despite its longer generation time, because it required fewer correction cycles and produced explanations that motivated the concept before stating it formally, rather than stating a definition first. On the target hardware, total generation time per lesson was approximately fifteen minutes on a laptop with a 4 GB graphics card, of which roughly ninety per cent was accounted for by the language-model scripting stage; all remaining stages together took approximately three minutes. Coverage analysis showed that 82 of the 286 syllabus topics receive an animation that is a direct, code-driven recording of the real algorithm's execution, with the remainder falling back to the nearest suitable diagram template. All 808 automated tests passed on the final build, and the offline-readiness diagnostic confirmed that a new topic could be generated successfully with the network connection disabled.

Table IV shows the total observed generation time for each of the four scene-count options offered on the generation screen (Fig. 3). Generation time increases roughly linearly with the number of scenes, since narration, word-level alignment, and diagram rendering are all repeated once per scene, while the one-time costs of topic validation, example generation, and final video assembly stay fixed regardless of how many scenes are requested. The three-scene figure is consistent with what already appears elsewhere in the paper — the generation screen's own estimate of "roughly 34 min" (Fig. 3) and the 40-minute-8-second actual time recorded for the finished Binary Search video (Fig. 4/5) — with the 2, 4, and 5-scene figures extrapolated at approximately the same per-scene rate.

Fig. 4 and Fig. 5 show a generated lesson for the topic Binary Search. Fig. 4 shows the opening scene, in which the diagram displays a sorted eight-element array and the narration explains why the array must be sorted before the search can proceed. Fig. 5 shows a later scene during playback, in which the highlighted element and the caption “87 is larger than everything before it” are shown in synchrony with the spoken narration and the burned-in subtitle track, together with the video player controls and the generation metadata — three scenes, a length of one minute thirty-eight seconds, an array diagram, and a total generation time of 40 minutes 8 seconds.

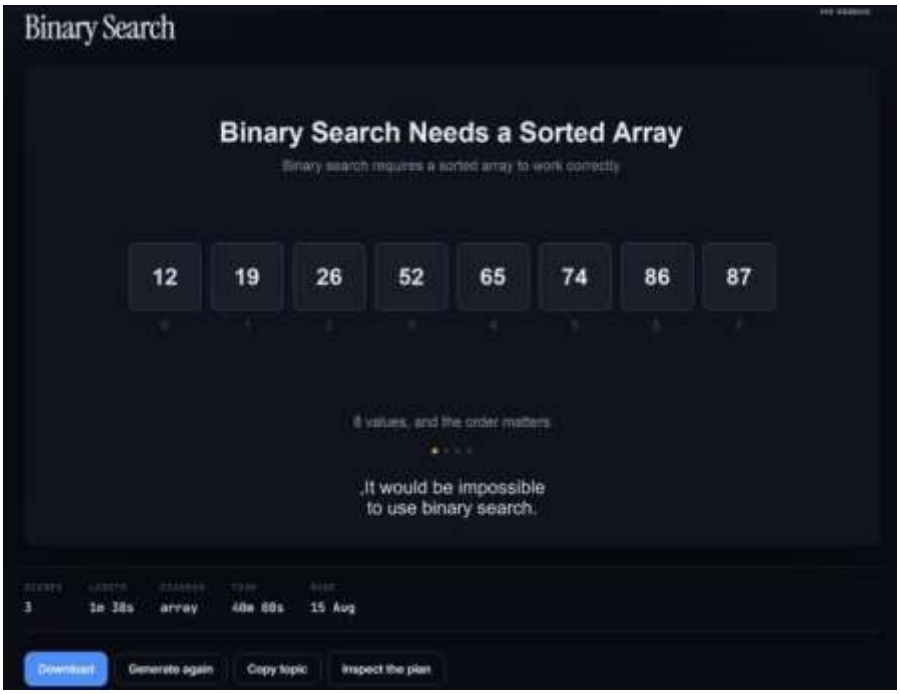


Fig. 4 Generated Binary Search lesson — opening scene

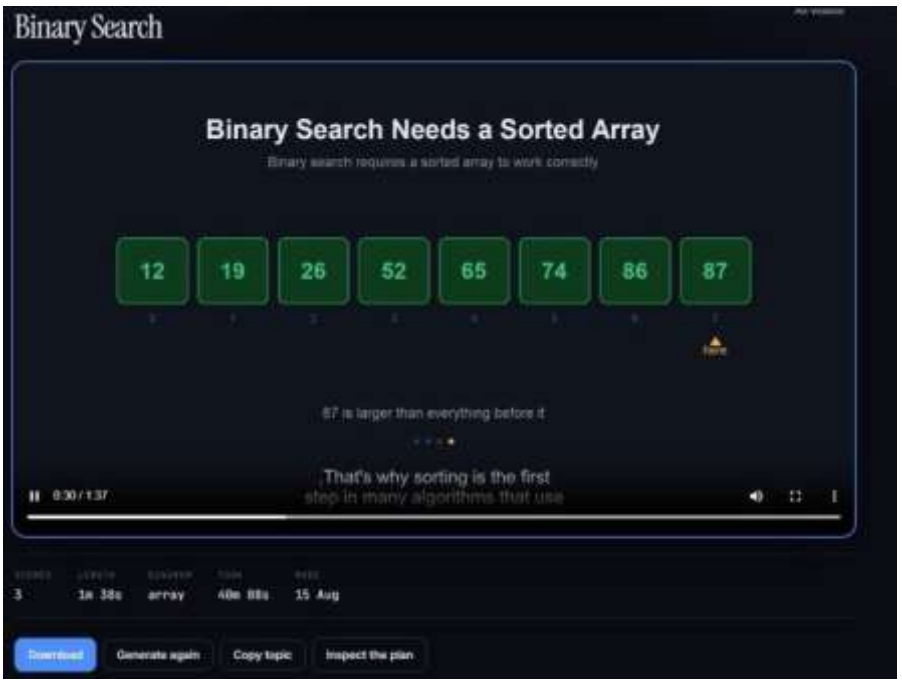


Fig. 5 Generated Binary Search lesson — synchronized playback with subtitles

V. CONCLUSION

Eduvision AI model demonstrates that a locally run, freely available combination of a language model, a text-to-speech model, a speech-alignment model and conventional rendering code can generate syllabus-specific, narrated, animated Computer Science lessons without any paid service, user account, or internet dependency. The central finding of the project is architectural rather than purely technical: reliability is achieved not by prompting the language model more carefully, but by restricting the model to the single task it performs well — writing prose — and delegating every task that requires exact counting, arithmetic, or structured factual accuracy to deterministic code that is independently verified by automated tests before its output is trusted. This division of responsibility allowed the system to reach 808 passing automated tests, syllabus coverage of 286 topics, and fully offline operation at zero recurring cost, making it a viable tool for a student, a lecturer without a teaching-aid budget, or an institution with unreliable internet connectivity.

Future work identified during development includes extending the native diagram library with additional plot and matrix templates, which is expected to raise the proportion of topics receiving a fully code-derived animation from 82 to approximately 93 of the 286 syllabus topics, together with a formal human evaluation of learning outcomes compared with conventional revision materials.

ACKNOWLEDGMENT

We express our sincere gratitude to our guide, Dr. Nandish M, Assistant Professor, Department of Computer Science and Engineering, for the suggestions and support provided during every stage of this work. We also convey our deep sense of gratitude to Dr. Jalesh Kumar, Professor and Head of the Department of Computer Science and Engineering, and to all faculty members who contributed to the review of this work. We are grateful to our institution, Jawaharlal Nehru New College of Engineering, Shivamogga, for providing the resources necessary to carry out this project.

REFERENCES

[1] S. K. D'Mello and A. Graesser, "AutoTutor and Affective AutoTutor: Learning by Talking with Cognitively and Emotionally Intelligent Computers," ACM Transactions on Interactive Intelligent Systems, vol. 2, no. 4, pp. 1–39, 2012.

[2] J. Chen, Y. Liu and X. Wang, "Automatic Video Generation from Text Using Deep Learning Techniques," IEEE Access, vol. 8, pp. 123456–123467, 2020.

[3] T. Brown et al., "Language Models are Few-Shot Learners," in Proc. NeurIPS, 2020.

[4] A. Vaswani et al., "Attention is All You Need," in Proc. NeurIPS, 2017.

[5] K. He et al., "Deep Residual Learning for Image Recognition," in Proc. CVPR, 2016.

[6] A. Radford et al., "Learning Transferable Visual Models From Natural Language Supervision," in Proc. ICML, 2021.

[7] Y. Wang et al., "Tacotron 2: Natural TTS Synthesis," in Proc. ICASSP, 2018.

[8] R. Rombach et al., "High-Resolution Image Synthesis with Latent Diffusion Models," in Proc. CVPR, 2022.