
LECTUREMIND AI – AI-POWERED LEARNING COMPANION

Presented By:

UTTAM RAWAT

THDC-IHET ,TEHRI , UTTARAKHAND

B. Tech CSE (Ongoing 2023-2027).

OUTLINE

- Problem Statement
- Proposed System/Solution
- System Development Approach
- Algorithm & Deployment
- Result
- Conclusion
- Future Scope
- References

PROBLEM STATEMENT

- Students struggle to convert long lecture audio into structured study material.
- Manual note-taking is time-consuming and error-prone.
- Existing tools are : Cloud-dependent.
- Costly (paid APIs)Limited to transcription only.
- Lack of interactive learning from lecture content.
- No unified system for notes, flashcards, quizzes, and Q&A.

PROPOSED SOLUTION

- End-to-end local-first web application powered by AI
- Automatic transcription, cleaning, and structured content generation
- RAG-based chatbot for intelligent Q&A with lectures
- No paid API required – fully open-source and self-hosted

SYSTEM DEVELOPMENT APPROACH

- Modular architecture: separate concerns (transcription, cleaning, generation, RAG)
- Streamlit for UI — rapid prototyping and user-friendly dashboard
- Local vector database (FAISS) for efficient retrieval
- Fallback heuristics for LLM functions when API unavailable

ALGORITHM & DEPLOYMENT

- **Whisper STT:** Convert audio to raw transcript
- **Text Cleaning:** Remove fillers, fix punctuation, split paragraphs
- **Content Generation:** Notes, flashcards, quizzes via LLM or heuristic

ALGORITHM & DEPLOYMENT

- **Embedding & Indexing:** SentenceTransformers + FAISS vector store
- **RAG Chat:** Retrieve chunks → Generate contextual answers
- **Deployment:** Run via Streamlit; no external service required

RESULTS

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists project files and folders. The main area contains two code editors: 'app.py' and 'chat.py'. The 'chat.py' editor is active and displays the following Python code:

```
1 import os
2 from typing import List
3
4 from .embeddings import EmbeddingModel
5 from .vector_store import LectureVectorStore
6
7
8 def _format_context(ctx_chunks: List[str]) -> str:
9     ctx = "\n\n".join([f"- {c}" for c in ctx_chunks])
10    return ctx
11
12
13 def _llm_answer_fallback(question: str, context: str) -> str:
14     # Simple heuristic answer generator combining context and question
15     answer = (
16         "Based on the retrieved lecture context, here is a synthesized answer:\n\n"
17         f"Question: {question}\n\n"
18         f"Context:\n{context}\n\n"
19         "Summary: The context suggests key points relevant to your question. "
20         "Please review the bullet points above; they reflect the most relevant excerpts."
21     )
22     return answer
23
24
25 def answer_question(question: str, store: LectureVectorStore, top_k: int = 5) -> str:
26     embedder = EmbeddingModel()
27     q_emb = embedder.embed_query(question)
```

The Explorer sidebar shows the following file structure:

- OPEN EDITORS:
 - app.py
 - chat.py
- LECTUREMIND AI:
 - .git
 - ai-learning-companion
 - audio
 - llm
 - rag
 - _pycache_
 - faiss_index
 - lecture_20251218_214427
 - lecture_20251226_201204
 - lecture_20251229_175609
 - .gitkeep
 - chat.py
 - embeddings.py
 - vector_store.py
 - transcripts
 - utils
 - app.py
 - README.md
 - requirements.txt
 - LectureMindAI
 - sampleaudio
 - venv
 - OUTLINE

RESULTS

The screenshot shows a code editor interface with a dark theme. The top bar includes a file menu (File, Edit, Selection, View, Go, Run, ...), a search bar (Q LectureMind AI), and various window control icons. The left sidebar is an 'EXPLORER' view showing a project structure under 'LECTUREMIND AI'. Files listed include app.py, embeddings.py, chat.py, vector_store.py, transcripts, utils, README.md, requirements.txt, LectureMindAI, sampleaudio, and venv. The 'embeddings.py' file is currently selected and highlighted with a blue border. The main editor area displays the following Python code:

```
1 import os
2 from typing import List
3
4 import numpy as np
5 from sentence_transformers import SentenceTransformer
6
7
8 class EmbeddingModel:
9     """Wrapper around SentenceTransformers for embedding texts and queries."""
10
11     def __init__(self, model_name: str = None):
12         self.model_name = model_name or os.environ.get(
13             "EMBEDDING_MODEL", "sentence-transformers/all-MiniLM-L6-v2"
14         )
15         self.model = SentenceTransformer(self.model_name)
16
17     def embed_texts(self, texts: List[str]) -> np.ndarray:
18         embeddings = self.model.encode(texts, convert_to_numpy=True, normalize_embeddings=True)
19         return embeddings.astype(np.float32)
20
21     def embed_query(self, query: str) -> np.ndarray:
22         emb = self.model.encode([query], convert_to_numpy=True, normalize_embeddings=True)[0]
23         return emb.astype(np.float32)
```

The bottom navigation bar includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined in yellow), PORTS, and AUGMENT NEXT EDIT. A terminal input field shows 'powershell ▲ + ⌘ T'.

RESULTS

The screenshot shows a code editor interface with two open files: `app.py` and `embeddings.py`. The `app.py` file is the active editor, displaying Python code for a lecture companion application. The code imports various modules like os, json, shutil, datetime, streamlit, whisper, and several custom utility functions from the project's `utils` module. It also imports classes from `llm.summarizer`, `llm.flashcards`, `llm.quiz_generator`, and `llm.concept_identifier`. The `DATA_DIRS` dictionary defines paths for audio, raw transcripts, and cleaned transcripts relative to the project root. The `embeddings.py` file is visible in the background.

```
import os
import json
import shutil
from datetime import datetime
import streamlit as st
# Whisper STT
try:
    import whisper
except Exception:
    whisper = None
from utils.text_cleaner import clean_transcript
from llm.summarizer import generate_notes
from llm.flashcards import generate_flashcards
from llm.quiz_generator import generate_quiz
from llm.concept_identifier import identify_concepts, filter_concepts
from rag.embeddings import EmbeddingModel
from rag.vector_store import LectureVectorStore
from rag.chat import answer_question

PROJECT_ROOT = os.path.dirname(os.path.abspath(__file__))
DATA_DIRS = {
    "audio": os.path.join(PROJECT_ROOT, "audio"),
    "raw": os.path.join(PROJECT_ROOT, "transcripts", "raw"),
    "cleaned": os.path.join(PROJECT_ROOT, "transcripts", "cleaned"),
```

The left sidebar shows the project structure under the `LECTUREMIND AI` folder, including subfolders `llm`, `rag`, `utils`, and files `chat.py`, `embeddings.py`, `vector_store.py`, `requirements.txt`, and `README.md`.

RESULTS

The screenshot shows the LectureMind AI web application running at `localhost:8501`. The interface has a dark theme.

Lecture Session

Lecture ID (auto if empty): `lecture_20260111_221028`

Set Lecture ID

Storage Paths:

- `E:\LectureMind AI\ai-learning-compan`
- `E:\LectureMind AI\ai-learning-compan`
- `E:\LectureMind AI\ai-learning-compan`

AI Settings

AI Source: Local AI (ComfyB)

LectureMind AI – AI-powered Learning Companion

Upload lectures, auto-generate study materials, and chat with your content.

1 Audio Upload

Upload lecture audio (.mp3/.wav)

Drag and drop file here
Limit 200MB per file • WAV, MP3

Browse files

1 Introduction to Algorithms.mp3 10.00MB

X

Saved: E:\LectureMind AI\ai-learning-companion\audio\lecture_20260111_221028.mp3

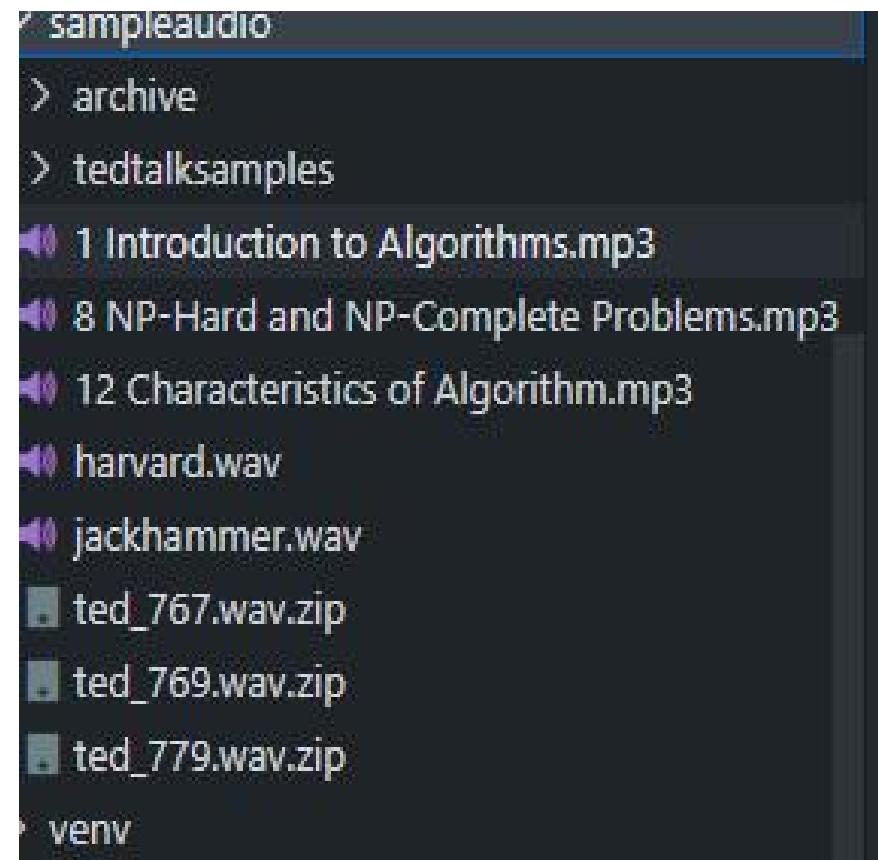
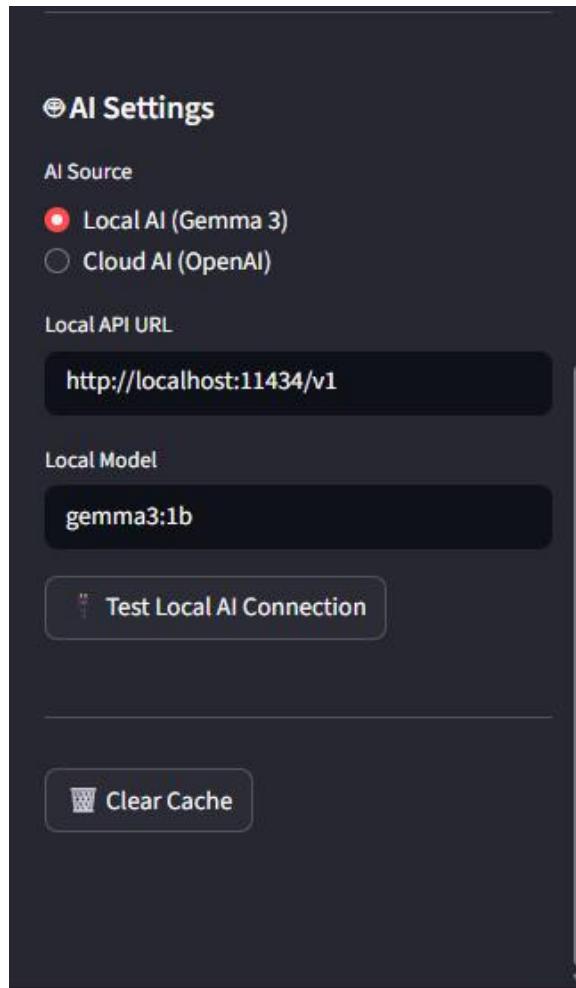
2 Speech-to-Text (Whisper)

Transcription language: Auto

Transcribe Audio

Transcript Cleaning

RESULTS



RESULTS

3 Transcript Cleaning

Raw Transcript

Hello friends, I am going to start our course on algorithms. Algorithms as a subject. Algorithms are a common subject for computer science engineering students. Most of the universities offer this course as a part of syllabus. And this is a very core subject and very important subject. And students face some difficulties in some of the topics in this one. They could not understand them very clearly. So here I am going to make the subject more and more simple so that you can easily understand at the UN you can practice that and also remember it easily. The importance of the subject is that apart from theoretical examination it is also important for competitive exams. And even programming contest most of them are designed from this subject only. If any programming challenges there they are mostly difficult to questions from this subject. So this is one of the challenges something for the student because they get a chance to solve different type of problems and they get the strategy or approach for solving the problem. So some students fail to understand it properly so they could not get the approach for the strategy for solving the problems. And they feel that they are lacking in the logic development or strategy development of a program. So I will be covering in-depth each and everything right from the basics to the depth. So that you will be able to answer any type of question from this one. I will cover the topic such that just you look at the question you can get asked. That I guarantee you right. So the thing is I will cover those points which are useful for solving the questions. Some job interviews or even for any other exam or some entrance exams like gate exam the questions are framed from algorithms also but there is also one of the key subject there. So you will be able to answer any type of question from there. To ask the lectures in the order I will be giving the numbers for each topic so that you follow that order you follow the sequence because I may be breaking the major topic in close more topics. I got more important thing this you feedback is very important for me because already I have made few prepared few videos using presentations and other tools but now I am giving a lecture on whiteboard so maybe I am new for the video quality and audio quality all these things. So technically once I get set up so I need your feedback so I can make my

RESULTS

Notes (Markdown)

Okay, here's a revised and expanded study note based on the provided lecture analysis, aiming for clarity, conciseness, and a professional tone. It builds upon the existing framework while adding more detail and structure.

Understanding Algorithms: A Comprehensive Overview

I. Introduction - What is an Algorithm?

- **Definition:** An algorithm is a precise, step-by-step procedure for solving a computational problem. It's a sequence of instructions designed to achieve a specific goal.
- **Why is it important?**
 - **Core Subject:** Fundamental to computer science, engineering, and programming students.
 - **Competitive Exams:** Crucial for success in algorithm-based exams (e.g., Algorithm Design & Analysis).
 - **Programming Contests:** Essential for problem-solving in programming contests and challenges.
- **The Challenge:** Students often struggle to understand algorithms because they are complex and require careful thought.
- **The Goal:** To provide a clear, step-by-step approach to solving problems.

II. Key Definitions

- **Program:** A set of instructions for a computer to execute. It's the complete, executable piece of code.
- **Difference between Program and Algorithm:**
 - **Program:** A complete, executable piece of code. Think of it as the *result* of the process.
 - **Algorithm:** A step-by-step procedure for solving a problem. It's the *process* itself.
- **Algorithm's Role:**
 - **Design Phase:** The *initial* stage where the algorithm is conceived and designed.
 - **Implementation Phase:** The stage where the algorithm is translated into code.
 - **Design Time:** The time spent on designing the algorithm.

III. The Algorithm Development Process

1. Design Phase:

- **Purpose:** Define the problem and the solution.
- **Steps:**
 - Understand the problem thoroughly.
 - Create a design that is perfect and thorough.
 - Ensure the design is clear and easy to understand.

2. Implementation Phase:

- **Purpose:** Translate the design into code.
- **Steps:**
 - Write the code.
 - Change the mind and delete the program.
 - Start writing the program.

3. Why is it important to understand the design?

- **Waste of time:** If you don't understand the problem, you can waste a lot of time.
- **Understanding the problem:** You need to understand the problem to create a solution.

IV. The Role of the Programmer

• Programmers are crucial:

- **Domain Knowledge:** Programmers need to have domain knowledge.
- **Problem Knowledge:** They need to know the problem and its solution.
- **Knowledge of the Problem:** They can help the student who understands the problem.
- **Knowledge of the Solution:** They can write the solution.

• Programmer's Role:

- **Write Programs:** Programmers write programs.

REFERENCES

- **OpenAI Whisper: Speech-to-text models**
- **FAISS: Efficient similarity search and clustering**
- **Streamlit: Rapid web application development**
- **SentenceTransformers: Semantic text embeddings**
- **LangChain: RAG pipeline inspiration and chunking strategies**

FUTURE SCOPE

- Multilingual transcription and content generation
- Real-time speaker diarization for multi-speaker lectures
- Collaborative RAG for shared lecture repositories
- Mobile app for on-the-go access to study materials

CONCLUSION

LectureMind AI democratizes learning by transforming raw lecture content into actionable study materials. With a modular, local-first architecture, it empowers students to learn smarter without infrastructure or cost barriers. The project is production-ready and scalable for classrooms and institutions.



THANK YOU

GITHUB LINK

- Github Link : <https://github.com/the-introvert20/LectureMind.AI.git>
- Deployment link : <https://lecturemindai-najuzbv6topc4zgedyafxh.streamlit.app/>