PROBLEM STATEMENT

Learning Data Structures and Algorithms (DSA) is hard—really hard. For most students, it's not just about understanding theoretical concepts, but also about applying them through coding, practicing consistently, and staying motivated over weeks or even months.

Often, learners face challenges like:

- Not knowing where to start or what to study next
- Feeling stuck while solving coding problems
- Lacking a structured practice plan
- Losing motivation when progress feels slow
- Spending too much time Googling concepts or solutions individually

This is where our project steps in. We're building a DSA Multi-Agent Companion—a personalized assistant made of intelligent AI agents that work together to support a student at every stage of their DSA journey. Whether it's understanding a tricky concept like segment trees, writing code for a tough recursion problem, or just needing a motivational nudge to keep going, the companion is always there, ready to help.

What makes this system powerful is multi-agent collaboration:
Each agent specializes in a different task—just like a team of tutors and mentors working in sync. One explains the theory, another helps with the code, one plans your schedule, and one keeps your spirits high. Together, they make the learning experience smoother, smarter, and much more engaging.

In short, we're solving the problem of scattered, lonely, and inefficient DSA preparation by bringing everything into one intelligent system—tailored for the modern learner.

Objective:

- To develop an AI-powered learning assistant that can:
- Classify and understand user queries.
- Route them to appropriate specialized modules (agents).
- Provide conceptual explanations, code-related help, study plans, and motivation as needed.

SOLUTION OVERVIEW

The solution contains 4 side agents, a central agent and an orchestrator agent working together to solve the problem.

1. **CENTRAL AGENT**
    - **Role** : Understand the user's intent. Is it a coding problem? A study plan? A motivation request?

- **Model Used:** Gemini 2.0 Flash
- **Why This Model :** Gemini Flash is optimized for speed and structured output, making it perfect for quick intent classification and data extraction. Also it can simply extract out the important information in it.

2. **CONCEPT AGENT**
   - **Role** : Teach programming concepts in a beginner-friendly way.
   - **Model Used:** DeepSeek LLaMA 70B (distill) from GROQ
   - **Why This Model :** DeepSeek's LLaMA 70B (distilled) is a strong general-purpose reasoning model trained on both code and natural language. It gives high-quality, context-aware explanations and can break down even tricky topics like recursion or dynamic programming in a simple and engaging way—perfect for students.

3. **CODE AGENT**
   - **Role** : Helps with code writing, error-finding, and debugging
   - **Model Used:** DeepSeek LLaMA 70B (distill) from GROQ
   - **Why This Model :** The same DeepSeek model performs well on code too. Since it has fine-tuned instruction-following ability and code understanding, it's ideal for explaining bugs, refactoring snippets, and even writing small functions. We didn't use a pure code model like Codex or GPT-Engineer because we also needed solid natural language explanation alongside the code.

4. **PLANNER AGENT**
   - **Role** : Creates personalized study plans based on topic and timeline
   - **Model Used:** DeepSeek LLaMA 70B (distill) from GROQ
   - **Why This Model :** The Planner Agent needs to understand input, structure a curriculum, and format it cleanly. DeepSeek shines at following instructions and generating long-form outputs like a 10-day or 30-day plan. It balances creativity (for varied plans) with discipline (structured format) quite well.

5. **MOTIVATION AGENT**
   - **Role** : Encourages users when they feel down or unmotivated.
   - **Model Used:** Gemma 2 9B IT from GROQ
   - **Why This Model :** Gemma 2 was chosen for its emotional tone and empathetic writing. It's lighter, faster, and trained to respond kindly—perfect for motivation use-cases. Since this agent doesn't need technical depth, a smaller, friendlier model like Gemma works best.

6. **ORCHESTRATOR AGENT**
   - **Role** : Routes the user prompt to the correct expert agent.
   - **Functions:**
     - process_request(prompt) → asks Central Agent to classify

    - route(query_type, payload) → sends to the right agent
- The orchestrator is lightweight but crucial. It stitches everything together like a conductor in an orchestra.

FLOW

1. **User Interaction Starts**
    - The user types a query into the Streamlit frontend chat interface
    - This message can be anything—asking for an explanation, code help, a study plan, or just a motivational boost

2. **Request Sent to Orchestrator Agent**
    - The Orchestrator acts as the central controller
    - It forwards the user's input to the Central Agent for classification

3. **Central Agent: Query Classification**
    - The Central Agent analyzes the input and identifies the type of request, such as: concept, code, plan and motivation. Also it gives topic to discuss and days to plan in JSON Format

4. **Request Sent to Orchestrator Agent**
    - Based on the predicted type, the Orchestrator routes the request to the appropriate agent

5. **Agent Processes the Request**
    - The selected agent processes the query using its specialized model, generates a response, and returns it to the orchestrator.

6. **Response Sent Back to Frontend**
    - The Orchestrator receives the agent's response and streams it back to the Streamlit frontend, where the user sees the reply in real time

LLM SELECTION

1. For **central agent** , we dont need such so Gemini will be best fit.
2. For **Concept agent**, Claude 3 Opus will be best since these days it is known for reasoning and education style explanations.
3. For **code agent**, GPT Turbo 4.5 or Code Llama 70B will be best suited since they are made specialised for code.
4. For **planner agent**, Claude 3 Sonnet will be best since they are best fit for consistency and structured output.
5. For **motivational agent**, we need some emotional or empathetic flavour so Gemini 1.5 pro will be best fit.

HOW TO RUN?

1. First of all, make a new virtual environment

```
python  -m venv new_env
```

2. Activate the virtual environment

```
new_env/Scripts/activate
```

3. Install dependencies

```
pip install -r requirements.txt
```

4. You need to write in .env file , the api keys you are using in below manner:

```
GEMINI_KEY = ‘your_gemini_api_key’
GROQ_API_KEY = ‘your_groq_api_key’
```

5. Now run the app.py file :

```
python -m streamlit run app.py ## or below one
streamlit run app.py
```

You can see the demo of my app below :
https://web-production-2376d.up.railway.app/

The app is hosted on railway.