Think of the project as a story with four main characters: **The Physicist**, **The Student**, **The Lab Assistant**, and **The Observer**.

Part 1: The Physicist (generate_data.py)

Goal: To create a perfect "textbook" of orbital examples.

This script plays the role of a physicist who knows the laws of gravity perfectly.

1. **It Knows the Rules:** It uses the mathematical formulas for gravity (ode_system function) that have been known for centuries.
2. **It Runs Simulations:** It runs hundreds of different simulations (N_SIMULATIONS = 200). In each one, it randomly places a planet somewhere with a random velocity.
3. **It Records Everything:** For each simulation, it uses a powerful calculator (scipy.solve_ivp) to predict the planet's exact path, step by step.
4. **It Creates the "Textbook":** It saves all of this information into a single, large file: orbital_data.csv. Each row in this file is like a sentence saying: "If a planet is *here* (x, y) and moving *this fast* (vx, vy), then in the next moment, it will be *over there* (next_x, next_y) and moving *that fast* (next_vx, next_vy)."

Outcome: A giant CSV file full of perfect, ground-truth examples of how planets move.

Part 2: The Student (train_model.py)

Goal: To learn the rules of gravity just by reading the textbook, without ever being told the formulas.

This script is the Machine Learning model, our "student."

1. **It Reads the Textbook:** It opens the orbital_data.csv file created by the Physicist.
2. **It Studies:** The student's only task is to find a pattern. It looks at thousands of (current_state -> next_state) examples and tries to build its own internal "intuition" (a complex mathematical function) to predict the next_state given any current_state.
3. **It Never Cheats:** Crucially, this script **never sees the original physics formulas**. It only learns from the data. It's like learning a language by listening to native speakers instead of reading a grammar book.
4. **It Takes a Test:** We use a portion of the data (the test set) to see how well it learned. The R^2 score tells us how accurate its predictions are.
5. **It Remembers What It Learned:** The script saves the student's trained "brain" into the file orbital_model.pkl. It also saves the data_scaler.pkl, which is a tool to make sure any future data is measured in the same way the student learned it.

Outcome: A trained ML model that can make an educated guess about the next step in an orbit.

Part 3: The Lab Assistant (app.py)

Goal: To take a request from the user and manage the experiment.

This is your backend server. It's the hard worker in the lab that connects everything.

1. **It's Always Ready:** When you run this script, it loads the trained model (the student's brain) into memory and waits for instructions from the user's web browser.
2. **It Receives an Order:** Your web browser sends it a request with the user's chosen starting conditions (e.g., "start the planet at x=1.5, y=0, ...").
3. **It Runs Two Experiments Simultaneously:**
    - **The AI's Prediction:** It gives the starting conditions to the ML model. The model predicts the next step. The server then takes that prediction and feeds it *back into the model* to get the *next* step, and so on. It chains hundreds of these small predictions together to form a full orbit. This is the **"learned" path**.
    - **The Physicist's Calculation:** It takes the *exact same* starting conditions and gives them to the original, perfect physics calculator from Part 1. This is the **"ground truth" path**.
4. **It Reports the Results:** It packages both complete paths (the AI's path and the Physicist's path) into a single JSON object and sends it back to the web browser.

Outcome: A web server that can generate two different orbital paths for any starting point.

Part 4: The Observer (frontend/ folder)

Goal: To provide a user interface to run the experiment and visualize the results.

This is the website you see in your browser.

1. **The Control Panel (index.html & style.css):** This provides the input boxes for the user to set the starting conditions and a "Simulate" button. It's the user's interface to the lab.
2. **The Action (script.js):** When you click "Simulate":
    - The JavaScript code grabs the numbers you entered.
    - It sends those numbers to the Lab Assistant (the backend server).
    - It waits for the Lab Assistant to send back the two paths (ml_path and physics_path).
3. **The Visualization:** Once it gets the data, the JavaScript code starts an animation on the <canvas> element. Frame by frame, it:
    - Clears the screen.
    - Draws the central star.

- Draws a **blue dot** at the current position from the "ground truth" path.
- Draws an **orange dot** at the current position from the "learned" path.
- It leaves a trail for both, so you can see the full orbit unfold.

The Big Picture

physics was used to create a dataset. train an **ML model** on that data to see if it can learn the underlying rules. Then build a **web application** that allows a user to input any starting conditions and visually compare the **perfectly calculated orbit** against the **orbit predicted by your AI**. The small differences between the two colored lines are the entire point of the project—they show you how well the AI learned