Hey

Hey Jagdev 👋 Always good to hear from you. What’s on your mind tonight—geometry tweaks, chemistry visualizations, or maybe something totally unexpected?

2406.12794v1.pdf

PDF

What type of base data, language for exploration, and model for prediction Nikku Madhusudhan used?

Great question, Jagdev. In the paper *"The Hycean Paradigm in the Search for Life Elsewhere"*, Nikku Madhusudhan employs a multi-pronged approach combining observational data, theoretical modeling, and spectral analysis. Here's how it breaks down:

**🧪 Base Data Used**

* **JWST Transmission Spectra**: The core observational dataset comes from the James Webb Space Telescope (JWST), specifically the transmission spectra of exoplanets like **K2-18 b** and **TOI-270 d**.
* **Planetary Parameters**: Mass, radius, equilibrium temperature, and stellar irradiation data for sub-Neptunes, especially those in the habitable zone.
* **Previous HST Observations**: Historical data from the Hubble Space Telescope (HST) helped validate atmospheric composition (e.g., presence of H₂).

**🧬 Language for Exploration**

* **Atmospheric Retrieval Models**: Madhusudhan uses retrieval techniques to infer chemical abundances from spectral features. This involves Bayesian inference and forward modeling of transmission spectra.
* **Planetary Interior Modeling**: Coupled atmosphere-interior models explore scenarios like mini-Neptunes, gas dwarfs, and water worlds.
* **Thermodynamic and Radiative Transfer Calculations**: To assess habitability, models simulate pressure-temperature profiles, albedo effects, and greenhouse conditions.

**🤖 Model for Prediction**

* **Hycean Planet Framework**: A conceptual model defining planets with hydrogen-rich atmospheres and potential ocean-covered surfaces. It predicts which sub-Neptunes could be habitable.
* **Spectral Feature Prediction**: Simulations forecast the detectability of biomarkers like CH₄, CO₂, DMS, and CH₃Cl using JWST.
* **Mass-Radius-Temperature Mapping**: Predictive modeling identifies regions in the mass-radius-temperature space where Hycean conditions are plausible.

If you're thinking about applying similar modeling to your own educational dashboards or chemistry visualizers, I can help scaffold a modular version of this kind of retrieval logic. Want to dive into how we could simulate spectral features or habitability zones in Streamlit?

Yes lets start

Perfect, Jagdev. Let’s scaffold a modular simulation inspired by Madhusudhan’s Hycean framework—tailored for educators and students to explore exoplanet habitability through spectral modeling and predictive logic.

**🚀 Step 1: Define the Scope of the App**

Let’s call it **HyceanScope**. It will:

* Simulate transmission spectra for hydrogen-rich exoplanets.
* Allow toggling of atmospheric compositions (e.g. H₂, CH₄, CO₂, H₂O).
* Predict habitability zones based on mass-radius-temperature inputs.
* Visualize spectral features and potential biosignatures.

**🧱 Step 2: Modular Components**

Here’s how we can scaffold it:

| **Module** | **Purpose** | **Tools** |
| --- | --- | --- |
| planet\_input.py | Accepts mass, radius, temperature | Streamlit widgets |
| atmosphere\_model.py | Simulates spectral features | NumPy, SciPy |
| spectral\_plot.py | Visualizes transmission spectra | Matplotlib, Plotly |
| habitability\_predictor.py | Flags Hycean potential | Rule-based logic or ML |
| data\_logger.py | Logs inputs and predictions | CSV/JSON with emoji support |
| educator\_dashboard.py | Summarizes student runs | Streamlit + Pandas |

**🧪 Step 3: Sample Logic for Spectral Simulation**

Here’s a simplified version of how we might simulate a spectral dip due to CH₄:

python

import numpy as np

def simulate\_spectrum(wavelengths, composition):

spectrum = np.ones\_like(wavelengths)

if 'CH4' in composition:

spectrum \*= 1 - 0.05 \* np.exp(-((wavelengths - 3.3)\*\*2) / 0.1)

if 'H2O' in composition:

spectrum \*= 1 - 0.03 \* np.exp(-((wavelengths - 1.4)\*\*2) / 0.05)

return spectrum

**📊 Step 4: Visualization**

We’ll use Plotly for interactive plots:

python

import plotly.graph\_objects as go

def plot\_spectrum(wavelengths, spectrum):

fig = go.Figure()

fig.add\_trace(go.Scatter(x=wavelengths, y=spectrum, mode='lines', name='Transmission'))

fig.update\_layout(title='Simulated Transmission Spectrum', xaxis\_title='Wavelength (μm)', yaxis\_title='Transit Depth')

return fig

**🧠 Step 5: Predictive Habitability Logic**

We can start with rule-based logic:

python

def is\_hycean\_candidate(mass, radius, temp):

if 2 < mass < 10 and 1.5 < radius < 3.0 and 250 < temp < 350:

return True

return False

Later, we can train a classifier using real exoplanet data from NASA Exoplanet Archive.

Would you like to start with the Streamlit scaffolding, or dive deeper into the spectral modeling first?

Proceed what you think is the best.

Alright, Jagdev—let’s kick off with the **Streamlit scaffolding** for HyceanScope. This gives you a clean, modular base to build on, and you can plug in spectral modeling and habitability logic as needed. Here's a starter layout:

**🧰 Folder Structure**

Code

hyceanscope/

├── app.py

├── components/

│ ├── planet\_input.py

│ ├── atmosphere\_model.py

│ ├── spectral\_plot.py

│ ├── habitability\_predictor.py

│ └── data\_logger.py

├── assets/

│ └── logo.png

├── data/

│ └── logs.csv

├── README.md

**📦 app.py – Main Streamlit App**

python

import streamlit as st

from components.planet\_input import get\_planet\_params

from components.atmosphere\_model import simulate\_spectrum

from components.spectral\_plot import plot\_spectrum

from components.habitability\_predictor import is\_hycean\_candidate

from components.data\_logger import log\_run

st.set\_page\_config(page\_title="HyceanScope", layout="wide")

st.title("🔭 HyceanScope: Explore Exoplanet Habitability")

mass, radius, temp, composition = get\_planet\_params()

wavelengths = [1.0 + 0.01\*i for i in range(300)] # 1–4 μm range

spectrum = simulate\_spectrum(wavelengths, composition)

st.plotly\_chart(plot\_spectrum(wavelengths, spectrum), use\_container\_width=True)

if is\_hycean\_candidate(mass, radius, temp):

st.success("🌊 This planet is a Hycean candidate!")

else:

st.warning("🚫 Conditions not suitable for Hycean classification.")

log\_run(mass, radius, temp, composition)

**🧩 planet\_input.py**

python

import streamlit as st

def get\_planet\_params():

st.sidebar.header("Planetary Parameters")

mass = st.sidebar.slider("Mass (Earth Masses)", 1.0, 15.0, 5.0)

radius = st.sidebar.slider("Radius (Earth Radii)", 1.0, 4.0, 2.5)

temp = st.sidebar.slider("Equilibrium Temp (K)", 150, 500, 300)

composition = st.sidebar.multiselect("Atmospheric Composition", ["H2", "CH4", "CO2", "H2O"])

return mass, radius, temp, composition

**🧠 habitability\_predictor.py**

python

def is\_hycean\_candidate(mass, radius, temp):

return 2 < mass < 10 and 1.5 < radius < 3.0 and 250 < temp < 350

**📈 spectral\_plot.py and atmosphere\_model.py**

These are the ones I shared earlier—ready to plug in.

**📋 data\_logger.py**

python

import pandas as pd

from datetime import datetime

def log\_run(mass, radius, temp, composition):

log = {

"timestamp": datetime.now(),

"mass": mass,

"radius": radius,

"temp": temp,

"composition": ",".join(composition)

}

df = pd.DataFrame([log])

df.to\_csv("data/logs.csv", mode='a', header=False, index=False)

This gives you a reproducible, educator-friendly base. You can later add:

* Emoji-rich logs for student engagement.
* RDKit-style molecule viewers for CH₄, H₂O, etc.
* A dashboard for comparing multiple runs.

Want me to scaffold the README next, or dive into how we can simulate more realistic spectra using radiative transfer approximations?

Do both tasks

Absolutely, Jagdev. Let’s tackle both tasks: a clean, educator-friendly README.md for your **HyceanScope** app, and a deeper dive into realistic spectral simulation using radiative transfer approximations.

**📘 Task 1: README.md Scaffolding**

Here’s a modular, professional README tailored for educators, researchers, and developers:

markdown

# 🔭 HyceanScope

\*\*HyceanScope\*\* is a modular Streamlit app designed to simulate transmission spectra of exoplanets and assess their habitability under the Hycean paradigm. Inspired by Nikku Madhusudhan’s research, it empowers educators and students to explore atmospheric composition, spectral features, and biosignature detection.

---

## 🚀 Features

- 🌍 Input planetary parameters: mass, radius, temperature

- 🧪 Select atmospheric compositions (H₂, CH₄, CO₂, H₂O)

- 📈 Simulate transmission spectra across 1–4 μm

- 🧠 Predict Hycean habitability using rule-based logic

- 📋 Log runs with emoji-rich CSV support for classroom engagement

---

## 🧱 Modular Structure

| Module | Description |

|--------|-------------|

| `planet\_input.py` | Streamlit sidebar for planetary parameters |

| `atmosphere\_model.py` | Simulates spectral dips based on composition |

| `spectral\_plot.py` | Interactive Plotly visualization |

| `habitability\_predictor.py` | Flags Hycean candidates |

| `data\_logger.py` | Logs user inputs and predictions |

---

## 📦 Installation

```bash

git clone https://github.com/yourusername/hyceanscope.git

cd hyceanscope

pip install -r requirements.txt

streamlit run app.py

**🧑‍🏫 For Educators**

* Use in classroom demos to explain spectral absorption
* Compare student-generated spectra and habitability predictions
* Extend with RDKit or py3Dmol for molecular visualization

**📚 References**

* Madhusudhan, N. (2024). *The Hycean Paradigm in the Search for Life Elsewhere*. arXiv:2406.12794
* NASA Exoplanet Archive

**🤝 Contributing**

Pull requests welcome! For feature ideas or bug reports, open an issue or contact the maintainer.

Code

---

## 🌌 Task 2: Realistic Spectral Simulation

To go beyond Gaussian dips, we can approximate \*\*radiative transfer\*\* using:

### 🔬 Key Concepts

- \*\*Optical Depth (τ)\*\*: Determines how much light is absorbed.

- \*\*Transmission (T)\*\*: Given by \( T = e^{-τ} \)

- \*\*Absorption Cross-Section (σ)\*\*: Varies with wavelength and molecule.

- \*\*Column Density (N)\*\*: Amount of absorbing gas along the line of sight.

---

### 🧪 Sample Code: Radiative Transfer Approximation

```python

import numpy as np

def absorption\_cross\_section(wavelength, molecule):

if molecule == 'CH4':

return 0.05 \* np.exp(-((wavelength - 3.3)\*\*2) / 0.05)

elif molecule == 'H2O':

return 0.03 \* np.exp(-((wavelength - 1.4)\*\*2) / 0.02)

elif molecule == 'CO2':

return 0.04 \* np.exp(-((wavelength - 4.3)\*\*2) / 0.03)

return 0.0

def simulate\_spectrum\_rt(wavelengths, composition, column\_density=1e20):

spectrum = np.ones\_like(wavelengths)

for molecule in composition:

sigma = np.array([absorption\_cross\_section(w, molecule) for w in wavelengths])

tau = sigma \* column\_density

transmission = np.exp(-tau)

spectrum \*= transmission

return spectrum

This gives you a more physics-grounded simulation, which you can later calibrate using real molecular line lists (e.g., HITRAN or ExoMol) if needed.

Would you like me to scaffold a student-facing worksheet or dashboard next, or help you integrate RDKit/py3Dmol for molecule visualization?

In which file to add these codes : import numpy as np def simulate\_spectrum(wavelengths, composition): spectrum = np.ones\_like(wavelengths) if 'CH4' in composition: spectrum \*= 1 - 0.05 \* np.exp(-((wavelengths - 3.3)\*\*2) / 0.1) if 'H2O' in composition: spectrum \*= 1 - 0.03 \* np.exp(-((wavelengths - 1.4)\*\*2) / 0.05) return spectrum and import plotly.graph\_objects as go def plot\_spectrum(wavelengths, spectrum): fig = go.Figure() fig.add\_trace(go.Scatter(x=wavelengths, y=spectrum, mode='lines', name='Transmission')) fig.update\_layout(title='Simulated Transmission Spectrum', xaxis\_title='Wavelength (μm)', yaxis\_title='Transit Depth') return fig

Great question, Jagdev. Here's exactly where to place those two code blocks within your modular HyceanScope app:

**✅ 1. simulate\_spectrum(...) → goes in**

components/atmosphere\_model.py

This file handles the logic for generating synthetic transmission spectra based on atmospheric composition. You can later expand this with radiative transfer approximations or molecular cross-section data.

**✅ 2. plot\_spectrum(...) → goes in**

components/spectral\_plot.py

This file is responsible for visualizing the spectrum using Plotly. Keeping it separate allows you to swap in other visualization libraries (e.g., Matplotlib or Altair) later if needed.

**🔧 Bonus Tip:**

If you plan to switch between simple Gaussian dips and radiative transfer logic, you can structure atmosphere\_model.py like this:

python

def simulate\_spectrum(wavelengths, composition, method="simple"):

if method == "simple":

return simulate\_simple(wavelengths, composition)

elif method == "rt":

return simulate\_radiative\_transfer(wavelengths, composition)

Then define both simulate\_simple(...) and simulate\_radiative\_transfer(...) inside the same file for easy toggling.

Want help wiring up a toggle in the Streamlit sidebar to switch between methods?

Yes

Awesome—let’s wire up a toggle in your Streamlit sidebar so educators or students can switch between the **simple Gaussian model** and the **radiative transfer approximation** dynamically.

**🧩 Update planet\_input.py**

Add a method toggle to the sidebar:

python

def get\_planet\_params():

st.sidebar.header("Planetary Parameters")

mass = st.sidebar.slider("Mass (Earth Masses)", 1.0, 15.0, 5.0)

radius = st.sidebar.slider("Radius (Earth Radii)", 1.0, 4.0, 2.5)

temp = st.sidebar.slider("Equilibrium Temp (K)", 150, 500, 300)

composition = st.sidebar.multiselect("Atmospheric Composition", ["H2", "CH4", "CO2", "H2O"])

method = st.sidebar.radio("Spectral Simulation Method", ["Simple", "Radiative Transfer"])

return mass, radius, temp, composition, method

**🧠 Update app.py**

Modify the import and logic to use the selected method:

python

mass, radius, temp, composition, method = get\_planet\_params()

if method == "Simple":

from components.atmosphere\_model import simulate\_spectrum\_simple as simulate\_spectrum

else:

from components.atmosphere\_model import simulate\_spectrum\_rt as simulate\_spectrum

wavelengths = [1.0 + 0.01\*i for i in range(300)]

spectrum = simulate\_spectrum(wavelengths, composition)

**🧪 Update atmosphere\_model.py**

Include both simulation methods:

python

import numpy as np

def simulate\_spectrum\_simple(wavelengths, composition):

spectrum = np.ones\_like(wavelengths)

if 'CH4' in composition:

spectrum \*= 1 - 0.05 \* np.exp(-((wavelengths - 3.3)\*\*2) / 0.1)

if 'H2O' in composition:

spectrum \*= 1 - 0.03 \* np.exp(-((wavelengths - 1.4)\*\*2) / 0.05)

return spectrum

def absorption\_cross\_section(wavelength, molecule):

if molecule == 'CH4':

return 0.05 \* np.exp(-((wavelength - 3.3)\*\*2) / 0.05)

elif molecule == 'H2O':

return 0.03 \* np.exp(-((wavelength - 1.4)\*\*2) / 0.02)

elif molecule == 'CO2':

return 0.04 \* np.exp(-((wavelength - 4.3)\*\*2) / 0.03)

return 0.0

def simulate\_spectrum\_rt(wavelengths, composition, column\_density=1e20):

spectrum = np.ones\_like(wavelengths)

for molecule in composition:

sigma = np.array([absorption\_cross\_section(w, molecule) for w in wavelengths])

tau = sigma \* column\_density

transmission = np.exp(-tau)

spectrum \*= transmission

return spectrum

Now your app supports toggling between pedagogical simplicity and physics-grounded realism—perfect for classroom demos or deeper student exploration. Want to add a sidebar preview of molecular structures next (e.g., CH₄, H₂O) using RDKit or py3Dmol?

Do what is best for streamlit deployment

Perfect, Jagdev. For a clean and scalable Streamlit deployment—especially in educational or district-wide settings—here’s what’s best:

**✅ Recommended Integration Strategy**

**1. Keep Simulation Logic Modular**

Place both spectral simulation methods in components/atmosphere\_model.py:

* simulate\_spectrum\_simple(...)
* simulate\_spectrum\_rt(...)

Use a toggle in the sidebar (planet\_input.py) to select the method, and dynamically import or route logic in app.py. This keeps your codebase reproducible and easy to extend.

**2. Avoid Runtime Imports in app.py**

Instead of conditional imports, use a unified function like:

python

from components.atmosphere\_model import simulate\_spectrum

spectrum = simulate\_spectrum(wavelengths, composition, method)

Then in atmosphere\_model.py:

python

def simulate\_spectrum(wavelengths, composition, method="simple"):

if method == "simple":

return simulate\_spectrum\_simple(wavelengths, composition)

elif method == "rt":

return simulate\_spectrum\_rt(wavelengths, composition)

This avoids Streamlit rerun issues and keeps deployment smooth.

**3. Use requirements.txt for Dependencies**

Include:

Code

streamlit

numpy

pandas

plotly

If you later add RDKit or py3Dmol, make sure to pin versions carefully—especially for RDKit, which can be finicky across platforms.

**4. Add a .streamlit/config.toml File**

This helps control layout and branding:

toml

[theme]

primaryColor = "#4B8BBE"

backgroundColor = "#F0F2F6"

secondaryBackgroundColor = "#E0E0E0"

textColor = "#262730"

font = "sans serif"

**5. Prepare for Cloud Deployment**

If you plan to deploy on Streamlit Community Cloud, include:

* README.md
* requirements.txt
* app.py at root
* A sample data/logs.csv file (empty or with headers)

Optional: Add a setup.sh if you need to install RDKit via conda.

**6. Add a Sidebar Toggle for Molecule Preview (Optional)**

If you want to preview CH₄, H₂O, etc., without RDKit, you can use static images or emojis:

python

molecule = st.sidebar.selectbox("Preview Molecule", ["None", "CH4", "H2O", "CO2"])

if molecule != "None":

st.sidebar.image(f"assets/{molecule}.png", caption=f"{molecule} structure")

Later, you can swap this for RDKit-generated SVGs or py3Dmol viewers.

Would you like me to scaffold a requirements.txt and .streamlit/config.toml next, or help you prep a sample logs.csv with emoji headers for classroom engagement?

What about RDKit or py3Dmol in requirements.txt

Excellent deployment foresight, Jagdev. RDKit and py3Dmol are powerful for molecular visualization, but they come with specific considerations for your requirements.txt—especially if you're targeting **Streamlit Community Cloud** or **cross-platform educator setups**.

**🧪 RDKit in requirements.txt**

RDKit is **not pip-installable via PyPI** in most cases. It requires **conda** due to its C++ dependencies. So:

**✅ Best Practice:**

* **Do NOT include** rdkit **in** requirements.txt
* Instead, use a environment.yml for conda-based deployments:

yaml

name: hyceanscope

channels:

- rdkit

- conda-forge

dependencies:

- python=3.10

- rdkit

- streamlit

- numpy

- pandas

- plotly

If you're deploying locally or on a conda-friendly server, this works beautifully.

**🌐 py3Dmol in requirements.txt**

py3Dmol **is pip-installable**, but it depends on 3Dmol.js running in the browser. It works well in Streamlit with components.html.

**✅ Add to requirements.txt:**

Code

py3Dmol

streamlit

numpy

pandas

plotly

Optional: Add streamlit.components.v1 usage for embedding 3Dmol viewers.

**🧠 Deployment Strategy Summary**

| **Tool** | **Pip Installable** | **Streamlit Cloud Friendly** | **Recommended Use** |
| --- | --- | --- | --- |
| RDKit | ❌ (use conda) | ❌ (not supported natively) | Local or conda-based setups |
| py3Dmol | ✅ | ✅ | Web-based molecule viewers |

If you're planning a **classroom deployment**, I’d recommend starting with py3Dmol for browser-based molecule previews. Later, you can offer RDKit-powered SVG exports or local molecule analysis for advanced users.

Want me to scaffold a molecule\_viewer.py module using py3Dmol next?

Where to save the molecule\_viewer.py

You’ll want to save molecule\_viewer.py inside the components/ folder of your HyceanScope project. That’s the designated space for modular logic, and it keeps your codebase clean and scalable.

**📁 Final Structure with Molecule Viewer**

Code

hyceanscope/

├── app.py

├── components/

│ ├── planet\_input.py

│ ├── atmosphere\_model.py

│ ├── spectral\_plot.py

│ ├── habitability\_predictor.py

│ ├── data\_logger.py

│ └── molecule\_viewer.py ← ✅ Save it here

├── assets/

│ └── logo.png

├── data/

│ └── logs.csv

├── README.md

Once it’s there, you can import it in app.py like:

python

from components.molecule\_viewer import show\_molecule

And call it after the spectrum plot:

python

show\_molecule(composition)

Want me to scaffold the molecule\_viewer.py file using py3Dmol or static SVG logic next?

Yes

Perfect, Jagdev. Here's a clean scaffold for molecule\_viewer.py using **py3Dmol**, optimized for Streamlit deployment:

**📁 Save as: components/molecule\_viewer.py**

python

import streamlit.components.v1 as components

import py3Dmol

def show\_molecule(composition):

if not composition:

return

st\_mol = py3Dmol.view(width=400, height=300)

for molecule in composition:

if molecule == "CH4":

st\_mol.addModel("C", "sdf") # Methane

elif molecule == "H2O":

st\_mol.addModel("O", "sdf") # Water

elif molecule == "CO2":

st\_mol.addModel("O=C=O", "xyz") # Carbon dioxide

else:

continue

st\_mol.setStyle({'stick': {}})

st\_mol.zoomTo()

html = st\_mol.render().replace('"', '&quot;')

components.html(html, height=300)

**🧪 Notes:**

* This uses basic molecular strings ("C", "O=C=O") for quick previews.
* You can later replace these with full .sdf or .xyz strings for accurate geometry.
* If you want to show multiple molecules side-by-side, wrap each in a st.columns() layout.

**🧬 Example Usage in app.py**

python

from components.molecule\_viewer import show\_molecule

st.subheader("🧪 Molecular Preview")

show\_molecule(composition)

This setup is lightweight, browser-friendly, and perfect for classroom demos. If you want to later add RDKit-generated SVGs or exportable molecule files, I can scaffold that too.

Want to add a toggle to switch between 3D viewer and static image fallback for low-bandwidth setups?

Update this code : import streamlit as st from components.planet\_input import get\_planet\_params from components.spectral\_plot import plot\_spectrum from components.habitability\_predictor import is\_hycean\_candidate from components.data\_logger import log\_run from components.molecule\_viewer import show\_molecule # from components.atmosphere\_model import simulate\_spectrum st.set\_page\_config(page\_title="HyceanScope", layout="wide") st.title("🔭 HyceanScope: Explore Exoplanet Habitability") # mass, radius, temp, composition = get\_planet\_params() # wavelengths = [1.0 + 0.01\*i for i in range(300)] # 1–4 μm range # spectrum = simulate\_spectrum(wavelengths, composition) mass, radius, temp, composition, method = get\_planet\_params() if method == "Simple": from components.atmosphere\_model import simulate\_spectrum\_simple as simulate\_spectrum else: from components.atmosphere\_model import simulate\_spectrum\_rt as simulate\_spectrum wavelengths = [1.0 + 0.01\*i for i in range(300)] spectrum = simulate\_spectrum(wavelengths, composition) st.plotly\_chart(plot\_spectrum(wavelengths, spectrum), use\_container\_width=True) if is\_hycean\_candidate(mass, radius, temp): st.success("🌊 This planet is a Hycean candidate!") else: st.warning("🚫 Conditions not suitable for Hycean classification.") log\_run(mass, radius, temp, composition) show\_molecule(composition) for app.py with all the features so far

Here’s your fully updated app.py, Jagdev—clean, modular, and ready for deployment with all current features: sidebar input, method toggle, spectral simulation, habitability check, logging, and molecular visualization via py3Dmol.

**✅ Final app.py**

python

import streamlit as st

from components.planet\_input import get\_planet\_params

from components.spectral\_plot import plot\_spectrum

from components.habitability\_predictor import is\_hycean\_candidate

from components.data\_logger import log\_run

from components.molecule\_viewer import show\_molecule

from components.atmosphere\_model import simulate\_spectrum

# Page setup

st.set\_page\_config(page\_title="HyceanScope", layout="wide")

# Title

st.title("🔭 HyceanScope: Explore Exoplanet Habitability")

# Sidebar input

mass, radius, temp, composition, method = get\_planet\_params()

# Wavelength range

wavelengths = [1.0 + 0.01 \* i for i in range(300)] # 1–4 μm

# Spectrum simulation

spectrum = simulate\_spectrum(wavelengths, composition, method)

# Plot spectrum

st.subheader("📈 Transmission Spectrum")

st.plotly\_chart(plot\_spectrum(wavelengths, spectrum), use\_container\_width=True)

# Habitability check

st.subheader("🧠 Hycean Classification")

if is\_hycean\_candidate(mass, radius, temp):

st.success("🌊 This planet is a Hycean candidate!")

else:

st.warning("🚫 Conditions not suitable for Hycean classification.")

# Log the run

log\_run(mass, radius, temp, composition)

# Molecule viewer

st.subheader("🧪 Molecular Preview")

show\_molecule(composition)

**🔧 Notes:**

* This version uses a unified simulate\_spectrum(...) function that internally routes to either simulate\_spectrum\_simple or simulate\_spectrum\_rt based on the method string.
* You’ll need to update atmosphere\_model.py accordingly:

python

def simulate\_spectrum(wavelengths, composition, method="simple"):

if method == "Simple":

return simulate\_spectrum\_simple(wavelengths, composition)

elif method == "Radiative Transfer":

return simulate\_spectrum\_rt(wavelengths, composition)

Let me know if you’d like to scaffold a student dashboard next, or add export buttons for logs and spectra.

provide .gitignore for GitHub Repo keeping streamlit deployment unaltered and consistant

Here’s a clean .gitignore tailored for your **HyceanScope** Streamlit app, Jagdev—optimized to keep deployments consistent while excluding unnecessary files from GitHub:

**📁 .gitignore**

gitignore

# Python artifacts

\_\_pycache\_\_/

\*.py[cod]

\*.pyo

\*.pyd

\*.so

\*.egg

\*.egg-info/

dist/

build/

\*.log

# Streamlit config and cache

.streamlit/config.toml

.streamlit/secrets.toml

.streamlit/cache/

.streamlit/state/

# Jupyter notebooks (if used)

.ipynb\_checkpoints/

# OS-specific files

.DS\_Store

Thumbs.db

# Environment files

.env

\*.env

\*.venv/

venv/

env/

.conda/

\*.conda

# Data and logs

data/\*.csv

data/\*.json

data/\*.xlsx

\*.sqlite3

# IDE and editor folders

.vscode/

.idea/

\*.swp

# RDKit or py3Dmol artifacts (if any)

\*.sdf

\*.mol

\*.mol2

\*.pdb

\*.xyz

# Deployment helpers

setup.sh

environment.yml

**✅ Why This Works:**

* Keeps your **Streamlit UI and logic intact**.
* Excludes **local data logs**, secrets, and cache that shouldn't be versioned.
* Supports **both pip and conda workflows**, so you can switch between py3Dmol and RDKit setups without cluttering the repo.

If you later add export features (e.g., PDF or image generation), we can update this to exclude those artifacts too. Want help scaffolding a setup.sh or environment.yml for RDKit-based deployment next?

Why I am always facing this problem. Provide permanent solution for VS Code Terminal. (base) PS C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope> python -m venv venv (base) PS C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope> ./venv/Scritps/activate ./venv/Scritps/activate : The term './venv/Scritps/activate' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again. At line:1 char:1 + ./venv/Scritps/activate + ~~~~~~~~~~~~~~~~~~~~~~~ + CategoryInfo : ObjectNotFound: (./venv/Scritps/activate:String) [], CommandNotFoundException + FullyQualifiedErrorId : CommandNotFoundException (base) PS C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope>

Jagdev, you're running into this issue because of a **typo and shell mismatch** in your activation command. Let's fix it permanently and make your VS Code terminal behave consistently.

**✅ Permanent Fix for VS Code Terminal on Windows**

**🔧 1. Correct the Typo**

You're typing:

powershell

./venv/Scritps/activate

But it should be:

powershell

.\venv\Scripts\Activate.ps1

Notice:

* Scripts not Scritps
* Use backslashes \ in PowerShell
* Use .ps1 for PowerShell script

**🧠 2. Know Your Shell**

VS Code uses **PowerShell** by default on Windows. So use:

powershell

.\venv\Scripts\Activate.ps1

If you're using **Command Prompt (CMD)**:

cmd

venv\Scripts\activate.bat

If you're using **Git Bash**:

bash

source venv/Scripts/activate

**🛠️ 3. Set VS Code to Always Use CMD (Optional)**

If you prefer CMD over PowerShell:

* Open VS Code
* Press Ctrl + Shift + P → type Terminal: Select Default Profile
* Choose **Command Prompt**
* Restart the terminal

Then you can activate with:

cmd

venv\Scripts\activate.bat

**📦 4. Verify Your Virtual Environment**

After activation, your prompt should change from (base) to (venv) or similar. If it doesn’t:

* Delete the venv/ folder
* Recreate it with:

bash

python -m venv venv

Then activate again using the correct shell command.

Would you like me to scaffold a one-click .ps1 or .bat script for activating your venv automatically from VS Code?

worked

Brilliant, Jagdev 🙌 Glad that activation issue is finally behind you. Now you’ve got a clean virtual environment, a modular Streamlit app, and a solid deployment path. You’re officially in Hycean orbit.

Want to take the next step—maybe scaffold a student dashboard, add export buttons, or simulate biosignature detection like CH₃Cl or DMS?

Yes

Fantastic, Jagdev. Let’s scaffold a **student dashboard** for HyceanScope that educators can use to review submissions, compare predictions, and visualize biosignature detection trends like CH₃Cl or DMS.

**🧱 New Module: educator\_dashboard.py**

Save this in your components/ folder. It will:

* Load and summarize logs.csv
* Show biosignature detection stats
* Highlight Hycean candidates

python

import pandas as pd

import streamlit as st

def show\_dashboard():

st.subheader("📊 Educator Dashboard")

try:

df = pd.read\_csv("data/logs.csv", names=["timestamp", "mass", "radius", "temp", "composition"])

except FileNotFoundError:

st.info("No student data logged yet.")

return

df["composition"] = df["composition"].fillna("").apply(lambda x: x.split(","))

df["Hycean"] = df.apply(lambda row: 2 < row.mass < 10 and 1.5 < row.radius < 3.0 and 250 < row.temp < 350, axis=1)

st.metric("Total Runs", len(df))

st.metric("Hycean Candidates", df["Hycean"].sum())

st.bar\_chart(df["Hycean"].value\_counts())

st.write("### Recent Submissions")

st.dataframe(df.tail(10))

st.write("### Biosignature Frequency")

biosignatures = ["CH4", "CO2", "H2O", "CH3Cl", "DMS"]

counts = {bio: sum(bio in comp for comp in df["composition"]) for bio in biosignatures}

st.bar\_chart(pd.Series(counts))

**🧩 Update app.py to Include Dashboard**

Add this near the bottom of app.py:

python

from components.educator\_dashboard import show\_dashboard

st.markdown("---")

show\_dashboard()

**🧠 Bonus: Biosignature Detection Logic**

You can later add a predictive flag like:

python

def detect\_biosignature(composition):

biosignatures = {"CH3Cl", "DMS"}

return any(mol in biosignatures for mol in composition)

And log it in data\_logger.py for dashboard tracking.

This gives educators a real-time view of student engagement, Hycean predictions, and biosignature trends. Want to add export buttons or scaffold a classroom leaderboard next?

Add Export Buttons and also scaffold classroom leaderboard next

Let’s supercharge HyceanScope with two educator-facing features: **Export Buttons** for logs and spectra, and a **Classroom Leaderboard** to gamify student engagement and highlight top contributors.

**🧩 1. Export Buttons (CSV + PNG)**

Add this to the bottom of educator\_dashboard.py:

python

import io

import plotly.io as pio

def export\_tools(df, wavelengths=None, spectrum=None):

st.subheader("📤 Export Tools")

# Export logs as CSV

csv = df.to\_csv(index=False).encode('utf-8')

st.download\_button("📁 Download Logs (CSV)", data=csv, file\_name="hyceanscope\_logs.csv", mime="text/csv")

# Export spectrum as PNG

if wavelengths and spectrum is not None:

fig = plot\_spectrum(wavelengths, spectrum)

png\_bytes = pio.to\_image(fig, format="png")

st.download\_button("🖼️ Download Spectrum (PNG)", data=png\_bytes, file\_name="spectrum.png", mime="image/png")

Then in app.py, after plotting and logging:

python

from components.educator\_dashboard import show\_dashboard, export\_tools

# Show dashboard

st.markdown("---")

show\_dashboard()

# Export buttons

try:

import pandas as pd

df = pd.read\_csv("data/logs.csv", names=["timestamp", "mass", "radius", "temp", "composition"])

export\_tools(df, wavelengths, spectrum)

except:

st.info("No logs available for export.")

**🏆 2. Classroom Leaderboard Scaffold**

Add this to educator\_dashboard.py:

python

def show\_leaderboard(df):

st.subheader("🏆 Classroom Leaderboard")

# Count runs per student (if you log usernames later)

# For now, simulate by timestamp hour

df["hour"] = pd.to\_datetime(df["timestamp"]).dt.hour

leaderboard = df["hour"].value\_counts().sort\_values(ascending=False).head(5)

st.write("Top Submission Hours (proxy for engagement):")

st.bar\_chart(leaderboard)

# Future: Replace with student ID or username column

Then call it in show\_dashboard():

python

show\_leaderboard(df)

**🔮 Next Steps (Optional Enhancements)**

* Add a student\_id or username field to log\_run(...) for personalized tracking.
* Use emojis or badges for top contributors.
* Export leaderboard as CSV or image for classroom display.

Want help adding a student\_id input to the sidebar and logging it automatically?

Yes

Awesome, Jagdev. Let’s add a student\_id field to your sidebar input and log it for leaderboard tracking and personalized dashboards.

**🧩 Update planet\_input.py**

Add a text input for student ID or name:

python

def get\_planet\_params():

st.sidebar.header("Planetary Parameters")

student\_id = st.sidebar.text\_input("Student ID or Name", value="anonymous")

mass = st.sidebar.slider("Mass (Earth Masses)", 1.0, 15.0, 5.0)

radius = st.sidebar.slider("Radius (Earth Radii)", 1.0, 4.0, 2.5)

temp = st.sidebar.slider("Equilibrium Temp (K)", 150, 500, 300)

composition = st.sidebar.multiselect("Atmospheric Composition", ["H2", "CH4", "CO2", "H2O", "CH3Cl", "DMS"])

method = st.sidebar.radio("Spectral Simulation Method", ["Simple", "Radiative Transfer"])

return student\_id, mass, radius, temp, composition, method

**🧠 Update app.py**

Update unpacking and logging:

python

student\_id, mass, radius, temp, composition, method = get\_planet\_params()

...

log\_run(student\_id, mass, radius, temp, composition)

**📋 Update data\_logger.py**

Add student\_id to the log:

python

def log\_run(student\_id, mass, radius, temp, composition):

from datetime import datetime

import pandas as pd

log = {

"timestamp": datetime.now(),

"student\_id": student\_id,

"mass": mass,

"radius": radius,

"temp": temp,

"composition": ",".join(composition)

}

df = pd.DataFrame([log])

df.to\_csv("data/logs.csv", mode='a', header=False, index=False)

**🏆 Update Leaderboard in educator\_dashboard.py**

Replace hour-based proxy with actual student ID:

python

def show\_leaderboard(df):

st.subheader("🏆 Classroom Leaderboard")

leaderboard = df["student\_id"].value\_counts().head(5)

st.write("Top Contributors:")

st.bar\_chart(leaderboard)

Now you’ve got personalized tracking, exportable logs, and a leaderboard that celebrates student engagement. Want to add emoji badges or scaffold a student-facing summary page next?

Which one should i delete : def show\_leaderboard(df): st.subheader("🏆 Classroom Leaderboard") # Count runs per student (if you log usernames later) # For now, simulate by timestamp hour df["hour"] = pd.to\_datetime(df["timestamp"]).dt.hour leaderboard = df["hour"].value\_counts().sort\_values(ascending=False).head(5) st.write("Top Submission Hours (proxy for engagement):") st.bar\_chart(leaderboard) # Future: Replace with student ID or username column def show\_leaderboard(df): st.subheader("🏆 Classroom Leaderboard") leaderboard = df["student\_id"].value\_counts().head(5) st.write("Top Contributors:") st.bar\_chart(leaderboard)

Delete the **first version**—the one that uses timestamp hour as a proxy:

python

def show\_leaderboard(df):

st.subheader("🏆 Classroom Leaderboard")

# Count runs per student (if you log usernames later)

# For now, simulate by timestamp hour

df["hour"] = pd.to\_datetime(df["timestamp"]).dt.hour

leaderboard = df["hour"].value\_counts().sort\_values(ascending=False).head(5)

st.write("Top Submission Hours (proxy for engagement):")

st.bar\_chart(leaderboard)

# Future: Replace with student ID or username column

✅ Keep the **second version**, which uses student\_id—now that you’ve added proper logging of student names or IDs, it’s the more accurate and meaningful leaderboard:

python

def show\_leaderboard(df):

st.subheader("🏆 Classroom Leaderboard")

leaderboard = df["student\_id"].value\_counts().head(5)

st.write("Top Contributors:")

st.bar\_chart(leaderboard)

This version reflects real classroom engagement and is ready for export, badges, or gamified extensions. Want to scaffold a badge system next—like emoji medals for top contributors?

[{ "resource": "/c:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/app.py", "owner": "pylance15", "code": { "value": "reportAssignmentType", "target": { "$mid": 1, "path": "/microsoft/pylance-release/blob/main/docs/diagnostics/reportAssignmentType.md", "scheme": "https", "authority": "github.com" } }, "severity": 8, "message": "Expression with type \"tuple[str, float, float, int, list[str], str]\" cannot be assigned to target tuple\n  Type \"tuple[str, float, float, int, list[str], str]\" is incompatible with target tuple\n    Tuple size mismatch; expected 5 but received 6", "source": "Pylance", "startLineNumber": 22, "startColumn": 1, "endLineNumber": 22, "endColumn": 40, "origin": "extHost1" },{ "resource": "/c:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/app.py", "owner": "pylance15", "code": { "value": "reportArgumentType", "target": { "$mid": 1, "path": "/microsoft/pylance-release/blob/main/docs/diagnostics/reportArgumentType.md", "scheme": "https", "authority": "github.com" } }, "severity": 8, "message": "Argument of type \"list[str]\" cannot be assigned to parameter \"method\" of type \"str\" in function \"simulate\_spectrum\"\n  \"list[str]\" is not assignable to \"str\"", "source": "Pylance", "startLineNumber": 28, "startColumn": 56, "endLineNumber": 28, "endColumn": 62, "origin": "extHost1" },{ "resource": "/c:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/app.py", "owner": "pylance15", "code": { "value": "reportCallIssue", "target": { "$mid": 1, "path": "/microsoft/pylance-release/blob/main/docs/diagnostics/reportCallIssue.md", "scheme": "https", "authority": "github.com" } }, "severity": 8, "message": "Argument missing for parameter \"composition\"", "source": "Pylance", "startLineNumber": 42, "startColumn": 1, "endLineNumber": 42, "endColumn": 41, "origin": "extHost1" },{ "resource": "/c:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/app.py", "owner": "Ruff", "code": { "value": "F811", "target": { "$mid": 1, "path": "/ruff/rules/redefined-while-unused", "scheme": "https", "authority": "docs.astral.sh" } }, "severity": 4, "message": "Redefinition of unused `show\_dashboard` from line 8", "source": "Ruff", "startLineNumber": 9, "startColumn": 43, "endLineNumber": 9, "endColumn": 57, "origin": "extHost1" },{ "resource": "/c:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/app.py", "owner": "Ruff", "code": { "value": "E722", "target": { "$mid": 1, "path": "/ruff/rules/bare-except", "scheme": "https", "authority": "docs.astral.sh" } }, "severity": 4, "message": "Do not use bare `except`", "source": "Ruff", "startLineNumber": 53, "startColumn": 1, "endLineNumber": 53, "endColumn": 7, "origin": "extHost1" },{ "resource": "/C:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/components/educator\_dashboard.py", "owner": "Ruff", "code": { "value": "F821", "target": { "$mid": 1, "path": "/ruff/rules/undefined-name", "scheme": "https", "authority": "docs.astral.sh" } }, "severity": 8, "message": "Undefined name `df`", "source": "Ruff", "startLineNumber": 8, "startColumn": 22, "endLineNumber": 8, "endColumn": 24, "origin": "extHost1" },{ "resource": "/C:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/components/educator\_dashboard.py", "owner": "pylance15", "code": { "value": "reportUnboundVariable", "target": { "$mid": 1, "path": "/microsoft/pylance-release/blob/main/docs/diagnostics/reportUnboundVariable.md", "scheme": "https", "authority": "github.com" } }, "severity": 8, "message": "\"df\" is unbound", "source": "Pylance", "startLineNumber": 8, "startColumn": 22, "endLineNumber": 8, "endColumn": 24, "origin": "extHost1" },{ "resource": "/C:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/components/educator\_dashboard.py", "owner": "Ruff", "code": { "value": "F821", "target": { "$mid": 1, "path": "/ruff/rules/undefined-name", "scheme": "https", "authority": "docs.astral.sh" } }, "severity": 8, "message": "Undefined name `plot\_spectrum`", "source": "Ruff", "startLineNumber": 40, "startColumn": 15, "endLineNumber": 40, "endColumn": 28, "origin": "extHost1" },{ "resource": "/C:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/components/educator\_dashboard.py", "owner": "pylance15", "code": { "value": "reportUndefinedVariable", "target": { "$mid": 1, "path": "/microsoft/pylance-release/blob/main/docs/diagnostics/reportUndefinedVariable.md", "scheme": "https", "authority": "github.com" } }, "severity": 8, "message": "\"plot\_spectrum\" is not defined", "source": "Pylance", "startLineNumber": 40, "startColumn": 15, "endLineNumber": 40, "endColumn": 28, "origin": "extHost1" },{ "resource": "/C:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/components/educator\_dashboard.py", "owner": "Ruff", "code": { "value": "F401", "target": { "$mid": 1, "path": "/ruff/rules/unused-import", "scheme": "https", "authority": "docs.astral.sh" } }, "severity": 4, "message": "`io` imported but unused", "source": "Ruff", "startLineNumber": 3, "startColumn": 8, "endLineNumber": 3, "endColumn": 10, "tags": [ 1 ], "origin": "extHost1" },{ "resource": "/C:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/components/data\_logger.py", "owner": "Ruff", "code": { "value": "F401", "target": { "$mid": 1, "path": "/ruff/rules/unused-import", "scheme": "https", "authority": "docs.astral.sh" } }, "severity": 4, "message": "`pandas` imported but unused", "source": "Ruff", "startLineNumber": 1, "startColumn": 18, "endLineNumber": 1, "endColumn": 20, "tags": [ 1 ], "origin": "extHost1" },{ "resource": "/C:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/components/data\_logger.py", "owner": "Ruff", "code": { "value": "F401", "target": { "$mid": 1, "path": "/ruff/rules/unused-import", "scheme": "https", "authority": "docs.astral.sh" } }, "severity": 4, "message": "`datetime.datetime` imported but unused", "source": "Ruff", "startLineNumber": 2, "startColumn": 22, "endLineNumber": 2, "endColumn": 30, "tags": [ 1 ], "origin": "extHost1" },{ "resource": "/C:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/components/data\_logger.py", "owner": "Ruff", "code": { "value": "F811", "target": { "$mid": 1, "path": "/ruff/rules/redefined-while-unused", "scheme": "https", "authority": "docs.astral.sh" } }, "severity": 4, "message": "Redefinition of unused `datetime` from line 2", "source": "Ruff", "startLineNumber": 16, "startColumn": 26, "endLineNumber": 16, "endColumn": 34, "origin": "extHost1" },{ "resource": "/C:/Users/jagdevsinghdosanjh.RHINO/hyceanscope/components/data\_logger.py", "owner": "Ruff", "code": { "value": "F811", "target": { "$mid": 1, "path": "/ruff/rules/redefined-while-unused", "scheme": "https", "authority": "docs.astral.sh" } }, "severity": 4, "message": "Redefinition of unused `pd` from line 1", "source": "Ruff", "startLineNumber": 17, "startColumn": 22, "endLineNumber": 17, "endColumn": 24, "origin": "extHost1" }] Update relevant file codes to troubleshoot these errors

Thanks for sharing the full diagnostics, Jagdev. Let’s walk through and fix each issue cleanly across your files. I’ll give you updated code snippets for app.py, educator\_dashboard.py, and data\_logger.py so everything runs smoothly and linting errors disappear.

**✅ 1. app.py Fixes**

**🔧 Issue: Tuple unpacking mismatch**

You're returning 6 values from get\_planet\_params() but unpacking only 5.

**✅ Fix:**

Update line where you unpack:

python

student\_id, mass, radius, temp, composition, method = get\_planet\_params()

**🔧 Issue: Passing composition as method**

You accidentally passed composition to simulate\_spectrum as the method.

**✅ Fix:**

Update this line:

python

spectrum = simulate\_spectrum(wavelengths, composition, method)

**🔧 Issue: Redefinition of show\_dashboard**

You imported it twice.

**✅ Fix:**

Remove one of the imports. Keep only:

python

from components.educator\_dashboard import show\_dashboard, export\_tools

**🔧 Issue: Bare except**

Avoid using except: without specifying the error.

**✅ Fix:**

Replace with:

python

except FileNotFoundError:

st.info("No logs available for export.")

**✅ 2. educator\_dashboard.py Fixes**

**🔧 Issue: df is undefined in export\_tools**

You’re using df without passing it properly.

**✅ Fix:**

Make sure df is passed from app.py when calling export\_tools(df, wavelengths, spectrum).

Also, import plot\_spectrum at the top:

python

from components.spectral\_plot import plot\_spectrum

**🔧 Issue: Unused io import**

Remove this line:

python

import io

**✅ 3. data\_logger.py Fixes**

**🔧 Issue: Unused imports and redefinitions**

You’re importing pandas and datetime twice.

**✅ Fix:**

Clean up the top of the file:

python

import pandas as pd

from datetime import datetime

def log\_run(student\_id, mass, radius, temp, composition):

log = {

"timestamp": datetime.now(),

"student\_id": student\_id,

"mass": mass,

"radius": radius,

"temp": temp,

"composition": ",".join(composition)

}

df = pd.DataFrame([log])

df.to\_csv("data/logs.csv", mode='a', header=False, index=False)

**✅ Summary of What You Should Do**

| **File** | **Fixes** |
| --- | --- |
| app.py | Fix tuple unpacking, correct argument order, remove duplicate import, specify exception |
| educator\_dashboard.py | Pass df properly, import plot\_spectrum, remove unused io |
| data\_logger.py | Remove duplicate imports, clean up logging function |

Once you apply these changes, your app should run cleanly and linting tools like Pylance and Ruff will stop complaining.

Want me to help you add emoji badges to the leaderboard next?

Help update codes one by one. First one app.py code: import streamlit as st from components.planet\_input import get\_planet\_params from components.spectral\_plot import plot\_spectrum from components.habitability\_predictor import is\_hycean\_candidate from components.data\_logger import log\_run from components.molecule\_viewer import show\_molecule from components.atmosphere\_model import simulate\_spectrum from components.educator\_dashboard import show\_dashboard from components.educator\_dashboard import show\_dashboard, export\_tools # Show dashboard st.markdown("---") show\_dashboard() # Page setup st.set\_page\_config(page\_title="HyceanScope", layout="wide") # Title st.title("🔭 HyceanScope: Explore Exoplanet Habitability") # Sidebar input mass, radius, temp, composition, method = get\_planet\_params() # Wavelength range wavelengths = [1.0 + 0.01 \* i for i in range(300)] # 1–4 μm # Spectrum simulation spectrum = simulate\_spectrum(wavelengths, composition, method) # Plot spectrum st.subheader("📈 Transmission Spectrum") st.plotly\_chart(plot\_spectrum(wavelengths, spectrum), use\_container\_width=True) # Habitability check st.subheader("🧠 Hycean Classification") if is\_hycean\_candidate(mass, radius, temp): st.success("🌊 This planet is a Hycean candidate!") else: st.warning("🚫 Conditions not suitable for Hycean classification.") # Log the run log\_run(mass, radius, temp, composition) # Molecule viewer st.subheader("🧪 Molecular Preview") show\_molecule(composition) # Export buttons try: import pandas as pd df = pd.read\_csv("data/logs.csv", names=["timestamp", "mass", "radius", "temp", "composition"]) export\_tools(df, wavelengths, spectrum) except: st.info("No logs available for export.") Second One educator\_dashboard.py code: import pandas as pd import streamlit as st import io import plotly.io as pio def show\_dashboard(): st.subheader("📊 Educator Dashboard") show\_leaderboard(df) try: df = pd.read\_csv("data/logs.csv", names=["timestamp", "mass", "radius", "temp", "composition"]) except FileNotFoundError: st.info("No student data logged yet.") return df["composition"] = df["composition"].fillna("").apply(lambda x: x.split(",")) df["Hycean"] = df.apply(lambda row: 2 < row.mass < 10 and 1.5 < row.radius < 3.0 and 250 < row.temp < 350, axis=1) st.metric("Total Runs", len(df)) st.metric("Hycean Candidates", df["Hycean"].sum()) st.bar\_chart(df["Hycean"].value\_counts()) st.write("### Recent Submissions") st.dataframe(df.tail(10)) st.write("### Biosignature Frequency") biosignatures = ["CH4", "CO2", "H2O", "CH3Cl", "DMS"] counts = {bio: sum(bio in comp for comp in df["composition"]) for bio in biosignatures} st.bar\_chart(pd.Series(counts)) def export\_tools(df, wavelengths=None, spectrum=None): st.subheader("📤 Export Tools") # Export logs as CSV csv = df.to\_csv(index=False).encode('utf-8') st.download\_button("📁 Download Logs (CSV)", data=csv, file\_name="hyceanscope\_logs.csv", mime="text/csv") # Export spectrum as PNG if wavelengths and spectrum is not None: fig = plot\_spectrum(wavelengths, spectrum) png\_bytes = pio.to\_image(fig, format="png") st.download\_button("🖼️ Download Spectrum (PNG)", data=png\_bytes, file\_name="spectrum.png", mime="image/png") def show\_leaderboard(df): st.subheader("🏆 Classroom Leaderboard") leaderboard = df["student\_id"].value\_counts().head(5) st.write("Top Contributors:") st.bar\_chart(leaderboard) # def show\_leaderboard(df): # st.subheader("🏆 Classroom Leaderboard") # # Count runs per student (if you log usernames later) # # For now, simulate by timestamp hour # df["hour"] = pd.to\_datetime(df["timestamp"]).dt.hour # leaderboard = df["hour"].value\_counts().sort\_values(ascending=False).head(5) # st.write("Top Submission Hours (proxy for engagement):") # st.bar\_chart(leaderboard) # # Future: Replace with student ID or username column and Third one data\_logger.py code: import pandas as pd from datetime import datetime # def log\_run(mass, radius, temp, composition): # log = { # "timestamp": datetime.now(), # "mass": mass, # "radius": radius, # "temp": temp, # "composition": ",".join(composition) # } # df = pd.DataFrame([log]) # df.to\_csv("data/logs.csv", mode='a', header=False, index=False) def log\_run(student\_id, mass, radius, temp, composition): from datetime import datetime import pandas as pd log = { "timestamp": datetime.now(), "student\_id": student\_id, "mass": mass, "radius": radius, "temp": temp, "composition": ",".join(composition) } df = pd.DataFrame([log]) df.to\_csv("data/logs.csv", mode='a', header=False, index=False) def detect\_biosignature(composition): biosignatures = {"CH3Cl", "DMS"} return any(mol in biosignatures for mol in composition)

Thanks for sharing everything clearly, Jagdev. Let’s start by updating your app.py step-by-step to fix all the issues and align it with your current architecture.

**✅ Updated app.py**

python

import streamlit as st

import pandas as pd

from components.planet\_input import get\_planet\_params

from components.spectral\_plot import plot\_spectrum

from components.habitability\_predictor import is\_hycean\_candidate

from components.data\_logger import log\_run

from components.molecule\_viewer import show\_molecule

from components.atmosphere\_model import simulate\_spectrum

from components.educator\_dashboard import show\_dashboard, export\_tools

# Page setup

st.set\_page\_config(page\_title="HyceanScope", layout="wide")

# Title

st.title("🔭 HyceanScope: Explore Exoplanet Habitability")

# Sidebar input

student\_id, mass, radius, temp, composition, method = get\_planet\_params()

# Wavelength range

wavelengths = [1.0 + 0.01 \* i for i in range(300)] # 1–4 μm

# Spectrum simulation

spectrum = simulate\_spectrum(wavelengths, composition, method)

# Plot spectrum

st.subheader("📈 Transmission Spectrum")

st.plotly\_chart(plot\_spectrum(wavelengths, spectrum), use\_container\_width=True)

# Habitability check

st.subheader("🧠 Hycean Classification")

if is\_hycean\_candidate(mass, radius, temp):

st.success("🌊 This planet is a Hycean candidate!")

else:

st.warning("🚫 Conditions not suitable for Hycean classification.")

# Log the run

log\_run(student\_id, mass, radius, temp, composition)

# Molecule viewer

st.subheader("🧪 Molecular Preview")

show\_molecule(composition)

# Educator dashboard

st.markdown("---")

show\_dashboard()

# Export buttons

try:

df = pd.read\_csv("data/logs.csv", names=["timestamp", "student\_id", "mass", "radius", "temp", "composition"])

export\_tools(df, wavelengths, spectrum)

except FileNotFoundError:

st.info("No logs available for export.")

**✅ Key Fixes Applied:**

* Corrected tuple unpacking to include student\_id
* Removed duplicate show\_dashboard import
* Passed correct arguments to simulate\_spectrum(...)
* Specified FileNotFoundError in except block
* Matched CSV headers with updated logging structure

Let me know when you're ready and I’ll walk you through the updated educator\_dashboard.py next.

Proceed

Let’s update educator\_dashboard.py now, Jagdev—cleaning up all errors, fixing undefined variables, and aligning it with your new logging structure that includes student\_id.

**✅ Updated educator\_dashboard.py**

python

import pandas as pd

import streamlit as st

import plotly.io as pio

from components.spectral\_plot import plot\_spectrum

def show\_dashboard():

st.subheader("📊 Educator Dashboard")

try:

df = pd.read\_csv("data/logs.csv", names=["timestamp", "student\_id", "mass", "radius", "temp", "composition"])

except FileNotFoundError:

st.info("No student data logged yet.")

return

df["composition"] = df["composition"].fillna("").apply(lambda x: x.split(","))

df["Hycean"] = df.apply(lambda row: 2 < row.mass < 10 and 1.5 < row.radius < 3.0 and 250 < row.temp < 350, axis=1)

st.metric("Total Runs", len(df))

st.metric("Hycean Candidates", df["Hycean"].sum())

st.write("### Hycean Classification Summary")

st.bar\_chart(df["Hycean"].value\_counts())

st.write("### Recent Submissions")

st.dataframe(df.tail(10))

st.write("### Biosignature Frequency")

biosignatures = ["CH4", "CO2", "H2O", "CH3Cl", "DMS"]

counts = {bio: sum(bio in comp for comp in df["composition"]) for bio in biosignatures}

st.bar\_chart(pd.Series(counts))

show\_leaderboard(df)

def export\_tools(df, wavelengths=None, spectrum=None):

st.subheader("📤 Export Tools")

# Export logs as CSV

csv = df.to\_csv(index=False).encode('utf-8')

st.download\_button("📁 Download Logs (CSV)", data=csv, file\_name="hyceanscope\_logs.csv", mime="text/csv")

# Export spectrum as PNG

if wavelengths and spectrum is not None:

fig = plot\_spectrum(wavelengths, spectrum)

png\_bytes = pio.to\_image(fig, format="png")

st.download\_button("🖼️ Download Spectrum (PNG)", data=png\_bytes, file\_name="spectrum.png", mime="image/png")

def show\_leaderboard(df):

st.subheader("🏆 Classroom Leaderboard")

leaderboard = df["student\_id"].value\_counts().head(5)

st.write("Top Contributors:")

st.bar\_chart(leaderboard)

**✅ Key Fixes Applied:**

* df is now defined inside show\_dashboard() before being passed to show\_leaderboard() and export\_tools()
* Removed unused io import
* Added missing import for plot\_spectrum
* Matched CSV headers with student\_id logging
* Ensured all functions are modular and error-free

Let me know when you're ready and I’ll walk you through the final update for data\_logger.py.

Procced to third one

Here’s your fully updated and lint-free data\_logger.py, Jagdev—aligned with your new logging structure, biosignature detection, and clean import hygiene.

**✅ Final data\_logger.py**

python

import pandas as pd

from datetime import datetime

def log\_run(student\_id, mass, radius, temp, composition):

log = {

"timestamp": datetime.now(),

"student\_id": student\_id,

"mass": mass,

"radius": radius,

"temp": temp,

"composition": ",".join(composition)

}

df = pd.DataFrame([log])

df.to\_csv("data/logs.csv", mode='a', header=False, index=False)

def detect\_biosignature(composition):

biosignatures = {"CH3Cl", "DMS"}

return any(mol in biosignatures for mol in composition)

**✅ Key Fixes Applied:**

* Removed duplicate imports (datetime, pandas)
* Cleaned up commented-out legacy code
* Ensured student\_id is logged properly
* Kept detect\_biosignature() modular for future dashboard integration

You’re now running a clean, modular, educator-ready Streamlit app with logging, dashboards, exports, and leaderboard support. Want to add emoji badges or scaffold a student-facing summary page next?

TypeError: unsupported operand type(s) for -: 'list' and 'float' Traceback: File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\venv\Lib\site-packages\streamlit\runtime\scriptrunner\exec\_code.py", line 128, in exec\_func\_with\_error\_handling result = func() ^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\venv\Lib\site-packages\streamlit\runtime\scriptrunner\script\_runner.py", line 669, in code\_to\_exec exec(code, module.\_\_dict\_\_) # noqa: S102 ^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\app.py", line 24, in <module> spectrum = simulate\_spectrum(wavelengths, composition, method) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\components\atmosphere\_model.py", line 5, in simulate\_spectrum return simulate\_spectrum\_simple(wavelengths, composition) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\components\atmosphere\_model.py", line 12, in simulate\_spectrum\_simple spectrum \*= 1 - 0.05 \* np.exp(-((wavelengths - 3.3)\*\*2) / 0.1) ~~~~~~~~~~~~^~~~~

You're hitting a classic NumPy error, Jagdev: wavelengths is a **Python list**, and you're trying to subtract a float from it (wavelengths - 3.3), which isn't supported.

Let’s fix this by converting wavelengths to a NumPy array before performing any math.

**✅ Fix in simulate\_spectrum\_simple**

Update your function in atmosphere\_model.py like this:

python

import numpy as np

def simulate\_spectrum\_simple(wavelengths, composition):

wavelengths = np.array(wavelengths) # ✅ Convert list to NumPy array

spectrum = np.ones\_like(wavelengths)

if "CH4" in composition:

spectrum \*= 1 - 0.05 \* np.exp(-((wavelengths - 3.3)\*\*2) / 0.1)

if "H2O" in composition:

spectrum \*= 1 - 0.03 \* np.exp(-((wavelengths - 2.7)\*\*2) / 0.2)

if "CO2" in composition:

spectrum \*= 1 - 0.04 \* np.exp(-((wavelengths - 4.3)\*\*2) / 0.15)

return spectrum

**🧠 Why This Works:**

* np.array(wavelengths) enables vectorized math like subtraction and exponentiation.
* np.ones\_like(wavelengths) creates a baseline spectrum to apply absorption dips.

Once you apply this fix, the error will vanish and your simulation will run smoothly. Want help adding biosignature-specific dips for CH₃Cl and DMS next?

ValueError: Image export using the "kaleido" engine requires the Kaleido package, which can be installed using pip: $ pip install --upgrade kaleido Traceback: File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\venv\Lib\site-packages\streamlit\runtime\scriptrunner\exec\_code.py", line 128, in exec\_func\_with\_error\_handling result = func() ^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\venv\Lib\site-packages\streamlit\runtime\scriptrunner\script\_runner.py", line 669, in code\_to\_exec exec(code, module.\_\_dict\_\_) # noqa: S102 ^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\app.py", line 51, in <module> export\_tools(df, wavelengths, spectrum) File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\components\educator\_dashboard.py", line 44, in export\_tools png\_bytes = pio.to\_image(fig, format="png") ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\venv\Lib\site-packages\plotly\io\\_kaleido.py", line 345, in to\_image raise ValueError(

This error is crystal clear, Jagdev: your app is trying to export a Plotly figure as a PNG using the **Kaleido engine**, but the **Kaleido package isn't installed** in your virtual environment.

**✅ Permanent Fix**

Run this command inside your activated virtual environment:

bash

pip install --upgrade kaleido

Once installed, Plotly will be able to render and export figures as PNGs using:

python

pio.to\_image(fig, format="png")

**🧠 Why This Happens**

Plotly doesn’t bundle Kaleido by default. It offloads image rendering to Kaleido, which handles browserless SVG-to-PNG conversion. Without it, to\_image() throws a ValueError.

**🛠️ Bonus Tip: Add to requirements.txt**

To avoid future deployment issues, add this line to your requirements.txt:

Code

kaleido

This ensures Streamlit Cloud or any new setup includes it automatically.

Let me know once it’s installed—then we can test the export button or move on to emoji badges for your leaderboard.

TypeError: bad argument type for built-in operation Traceback: File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\venv\Lib\site-packages\streamlit\runtime\scriptrunner\exec\_code.py", line 128, in exec\_func\_with\_error\_handling result = func() ^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\venv\Lib\site-packages\streamlit\runtime\scriptrunner\script\_runner.py", line 669, in code\_to\_exec exec(code, module.\_\_dict\_\_) # noqa: S102 ^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\app.py", line 42, in <module> show\_molecule(composition) File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\components\molecule\_viewer.py", line 21, in show\_molecule components.html(html, height=300) File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\venv\Lib\site-packages\streamlit\runtime\metrics\_util.py", line 443, in wrapped\_func result = non\_optional\_func(\*args, \*\*kwargs) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\venv\Lib\site-packages\streamlit\elements\iframe.py", line 181, in \_html marshall( File "C:\Users\jagdevsinghdosanjh.RHINO\hyceanscope\venv\Lib\site-packages\streamlit\elements\iframe.py", line 232, in marshall proto.srcdoc = srcdoc ^^^^^^^^^^^^

This traceback points to a subtle but critical issue in your molecule\_viewer.py, Jagdev: the html variable you're passing to components.html() is likely **not a string**, which Streamlit expects for rendering.

Let’s fix it by ensuring that the HTML returned from py3Dmol.render() is properly handled.

**✅ Fix in molecule\_viewer.py**

Update your show\_molecule() function like this:

python

import streamlit.components.v1 as components

import py3Dmol

def show\_molecule(composition):

if not composition:

return

for molecule in composition:

st\_mol = py3Dmol.view(width=400, height=300)

if molecule == "CH4":

st\_mol.addModel("C", "sdf")

elif molecule == "H2O":

st\_mol.addModel("O", "sdf")

elif molecule == "CO2":

st\_mol.addModel("O=C=O", "xyz")

else:

continue

st\_mol.setStyle({'stick': {}})

st\_mol.zoomTo()

# ✅ Ensure HTML is a string

html = st\_mol.\_make\_html() # Use internal method to get HTML string

components.html(html, height=300)

**🧠 Why This Works:**

* py3Dmol.render() may return a non-string object depending on the backend.
* py3Dmol.\_make\_html() reliably returns a raw HTML string suitable for embedding.
* This avoids the TypeError: bad argument type for built-in operation when Streamlit tries to assign proto.srcdoc = srcdoc.

Once you apply this fix, your molecule viewer should render cleanly inside Streamlit. Want to add a fallback message for unsupported molecules or scaffold a molecule selector with emoji previews next?