PetalView Documentation

- ## High-Level Project Summary

We developed PetalView, a mobile application that monitors and predicts global flowering patterns using NASA Earth observation data and AI models. The app addresses the challenge by providing real-time bloom maps, AI-powered predictions, and community-driven flower observations, helping users, scientists, and policymakers track seasonal changes in vegetation.

This is important because flowering phenology is a key indicator of climate change, ecosystem health, and food security. By connecting people with nature and enabling better ecological insights, PetalView contributes to sustainability, biodiversity monitoring, and environmental awareness worldwide.

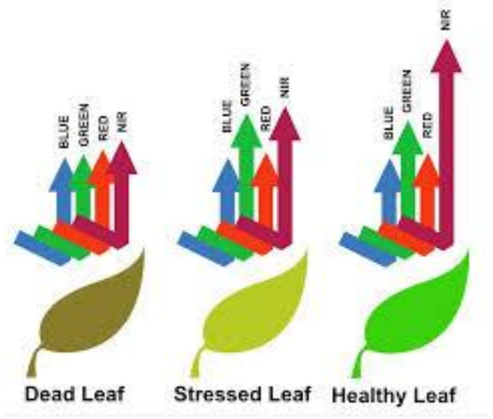
- ## Project Details

What exactly does it do?

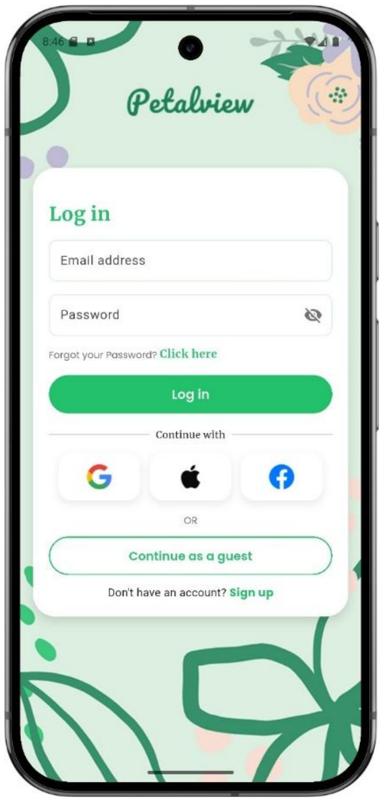
Our project, PetalView , is a monitoring and prediction tool that detects, visualizes, and forecasts plant blooming events using NASA Earth observation data. It allows users to explore current bloom conditions in their region or globally, and to receive predictions about when blooming is expected to occur in the near future. The tool is designed to serve farmers, researchers, and conservationists by providing insights into vegetation dynamics.

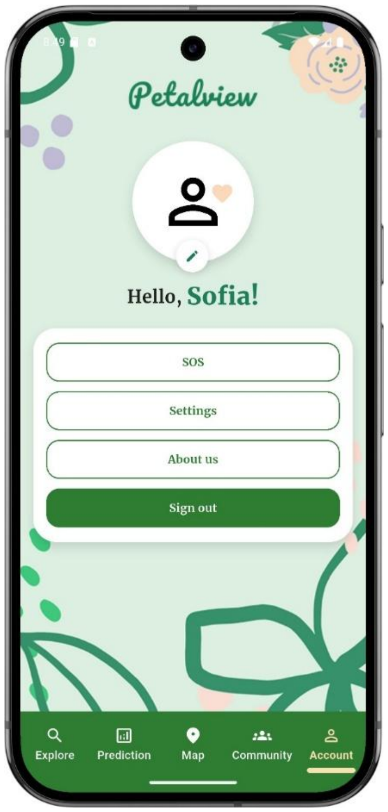
How does it work?

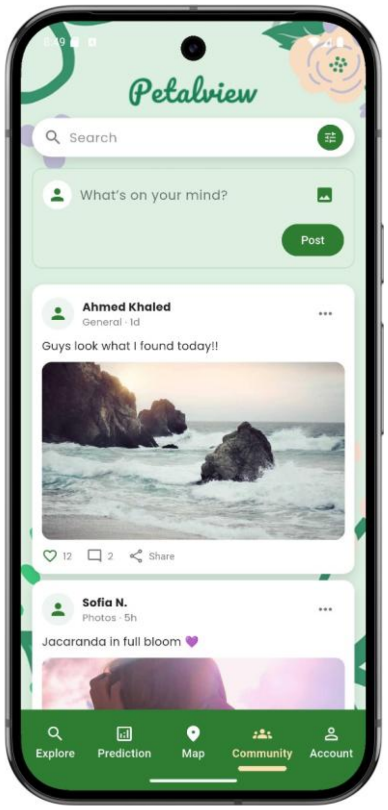
1. The system integrates NASA satellite datasets (e.g., MODIS, Landsat, GIBS) to extract vegetation indices such as NDVI.



2. An AI/ML prediction model processes location-based features to forecast upcoming bloom cycles.
3. The backend (FastAPI with a trained model) exposes prediction endpoints.
4. A Flutter-based mobile app provides users with two options:
5. Auto-detect their current location using GPS.
6. Manually enter a region of interest.
7. The app queries the backend API and displays results with clear visualizations, timelines, and explanations of bloom status.







Guys look what I found today!!

Jacaranda in full bloom 💜

What benefits does it have?

- Agriculture: Helps farmers determine optimal flowering and harvesting times, and supports disease/pest management strategies.
- Environment & Conservation: Assists in tracking ecological cycles, detecting invasive species, and monitoring biodiversity health.
- Research: Provides long-term bloom trends for phenology and climate change studies.
- Public Engagement: Raises awareness of seasonal cycles and ecological importance of flowering events through an easy-to-use visual tool.

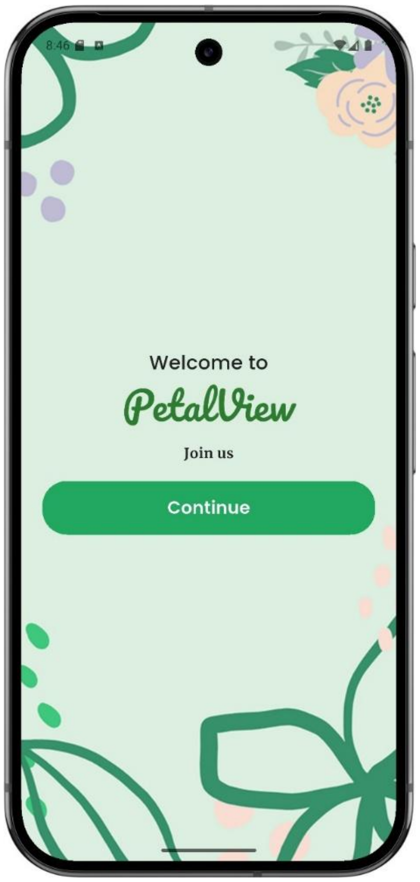
What do you hope to achieve?

- Deliver a scalable tool that works across global, regional, and local scales.
- Support sustainable agriculture and environmental management through AI-powered predictions.
- Make complex Earth observation data accessible and actionable for a wider community.
- Demonstrate how NASA datasets can provide real-world solutions to challenges like food security, climate adaptation, and biodiversity conservation.

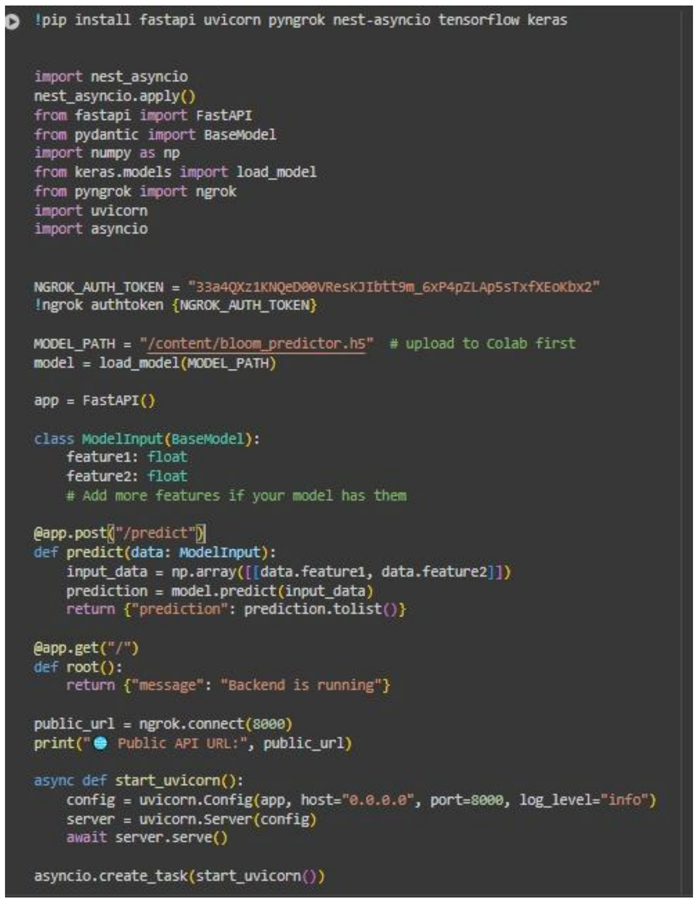
What tools, coding languages, and software did you use?

- **Languages & Frameworks:** Python (for AI/ML model training), FastAPI (backend API), Flutter/Dart (mobile application).
- **AI/ML Libraries:** TensorFlow, Keras, NumPy.
- Geospatial & NASA Tools: NDVI calculations from MODIS/Landsat, NASA GIBS for satellite imagery layers.
- **Hosting & Integration:** Google Colab (model training + hosting prototype), Ngrok (temporary tunneling), planned migration to cloud (AWS/GCP).
- **UI/UX:** Flutter animations and a green-themed design aligned with the ecological purpose of the app.

UI



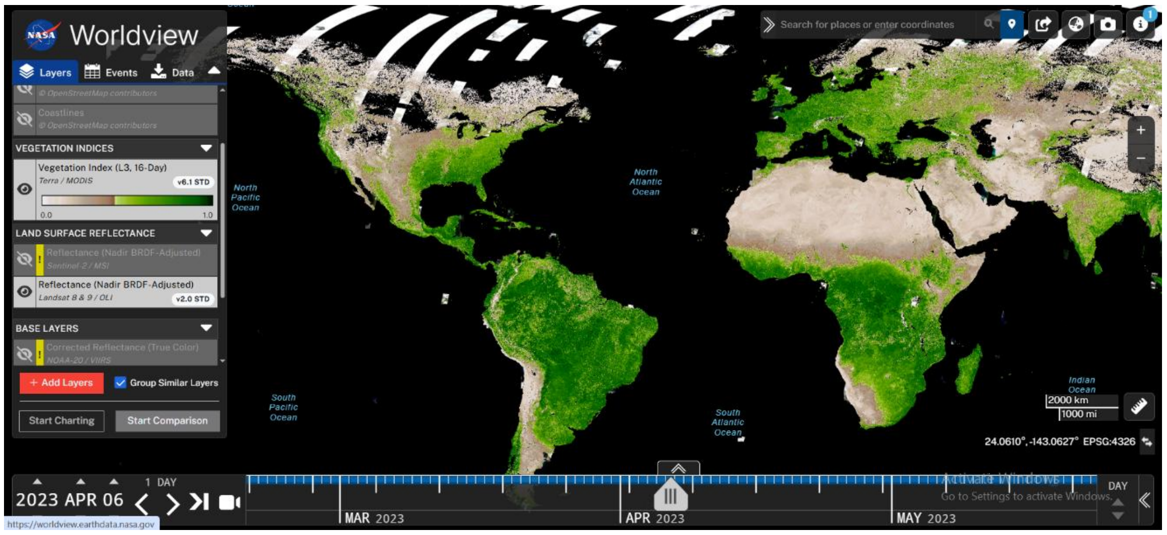
The Hosting Server

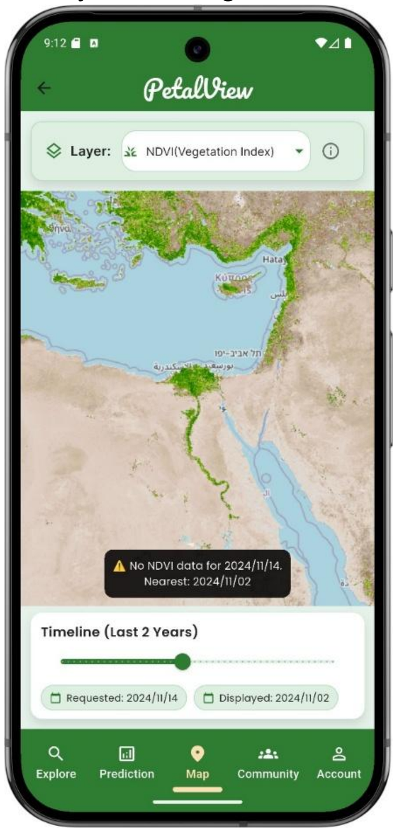


```
!pip install fastapi uvicorn pyngrok nest-asyncio tensorflow keras


import nest_asyncio
nest_asyncio.apply()
from fastapi import FastAPI
from pydantic import BaseModel
import numpy as np
from keras.models import load_model
from pyngrok import ngrok
import uvicorn
import asyncio


NGROK_AUTH_TOKEN = "33a4QXz1KNQeD00VResKJIbtt9m_6xP4pZLAp5sTxfXEoKbx2"
!ngrok authtoken {NGROK_AUTH_TOKEN}

MODEL_PATH = "/content/bloom_predictor.h5"  # upload to Colab first
model = load_model(MODEL_PATH)

app = FastAPI()

class ModelInput(BaseModel):
    feature1: float
    feature2: float
    # Add more features if your model has them

@app.post("/predict")
def predict(data: ModelInput):
    input_data = np.array([[data.feature1, data.feature2]])
    prediction = model.predict(input_data)
    return {"prediction": prediction.tolist()}

@app.get("/")
def root():
    return {"message": "Backend is running"}

public_url = ngrok.connect(8000)
print("🌐 Public API URL:", public_url)

async def start_uvicorn():
    config = uvicorn.Config(app, host="0.0.0.0", port=8000, log_level="info")
    server = uvicorn.Server(config)
    await server.serve()

asyncio.create_task(start_uvicorn())
```

- ## NASA Data Usage

The project leverages NASA's GIBS (Global Imagery Browse Services) and MODIS satellite imagery datasets to extract vegetation indices like NDVI. These indices are processed into training datasets for the AI model, allowing accurate prediction of blooming cycles by correlating climate, vegetation, and seasonal patterns.





Space Agency Partner & Other Data

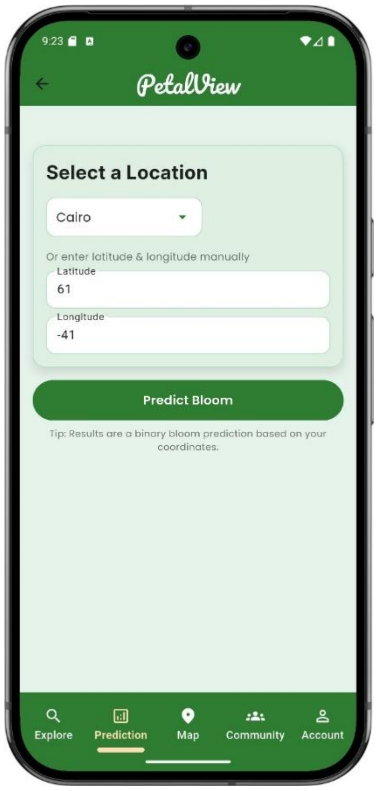
1. **ESA Sentinel-2 (European Space Agency):** High-resolution optical imagery used for vegetation health and bloom detection.

2. **JAXA ALOS (Japan Aerospace Exploration Agency):** Additional Earth observation data considered for vegetation cover monitoring.

Use of AI

The backend AI model, developed using TensorFlow/Keras, predicts blooming timelines based on processed NDVI values and climatic inputs. It uses supervised learning with labeled historical bloom cycles and adapts to user location inputs for localized forecasting. This allows PetalView to provide predictive, personalized, and region-specific insights.

The Prediction interface


9:23

PetalView

Select a Location

Cairo

Or enter latitude & longitude manually

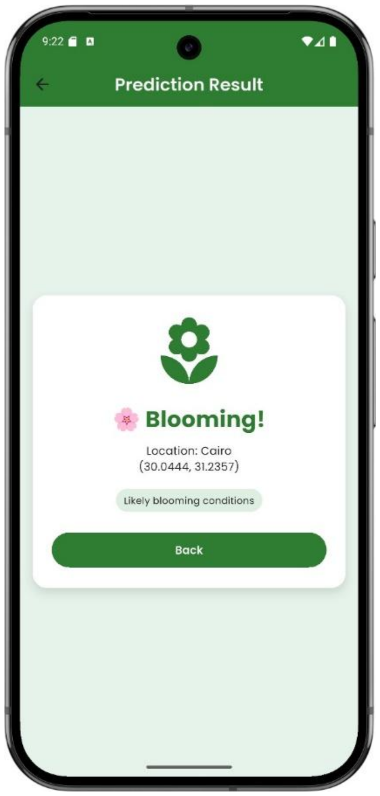
Latitude: 61

Longitude: -41

Predict Bloom

Tip: Results are a binary bloom prediction based on your coordinates.

Model Test on Cairo


9:22

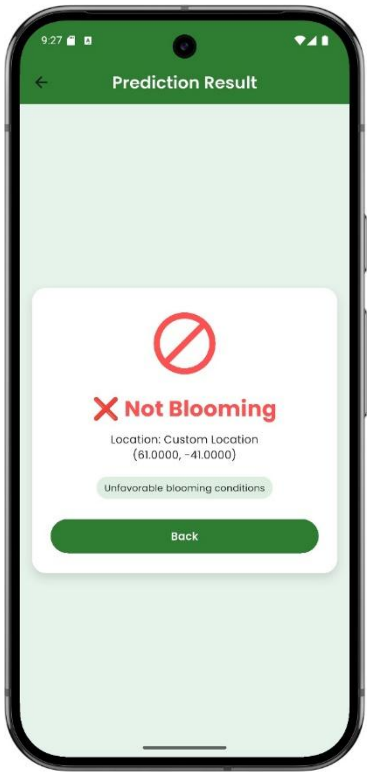
Prediction Result

🌸 Blooming!

Location: Cairo
(30.0444, 31.2357)

Likely blooming conditions

Model Test on Greenland Coordinates

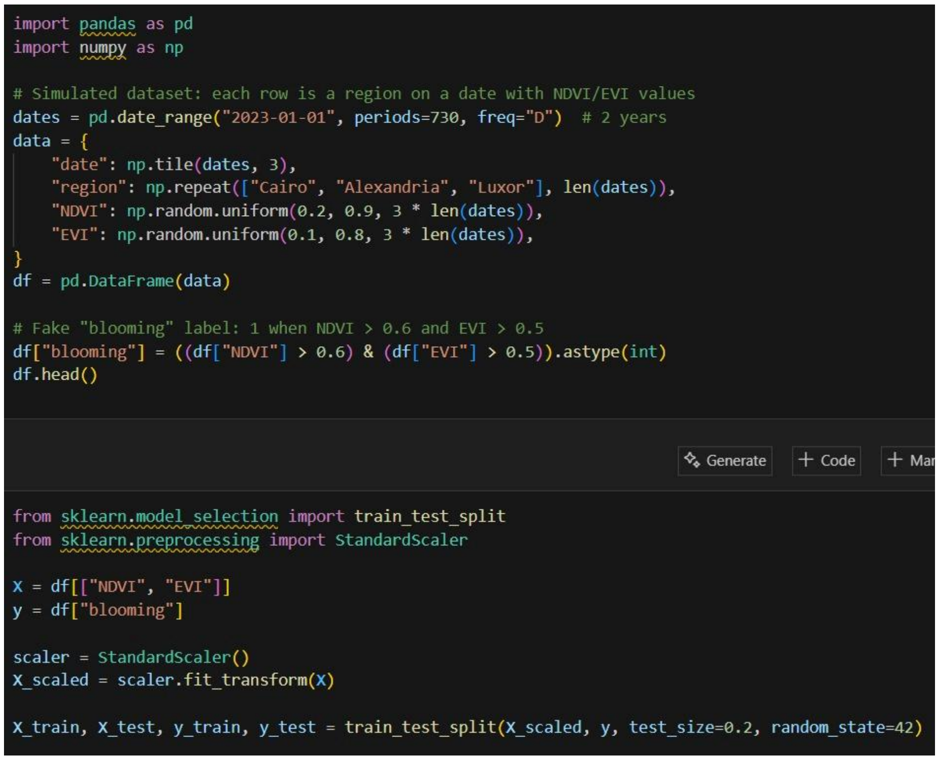

9:27

Prediction Result

❌ Not Blooming

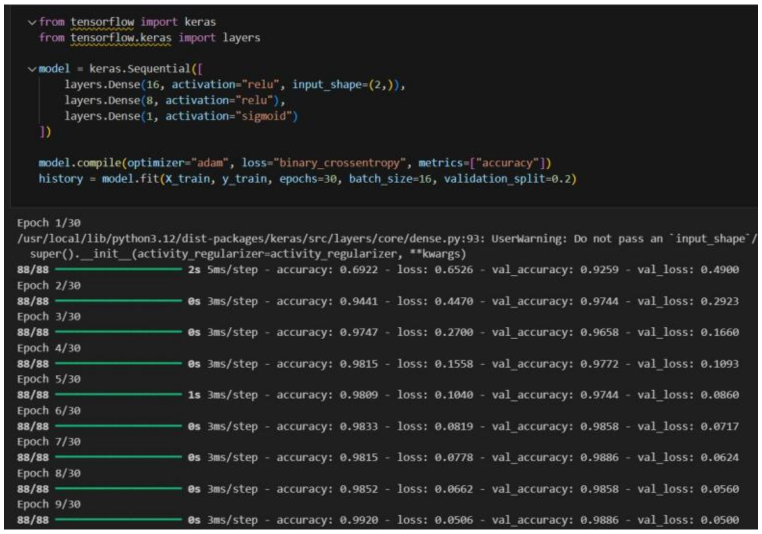
Location: Custom Location
(61.0000, -41.0000)

Unfavorable blooming conditions



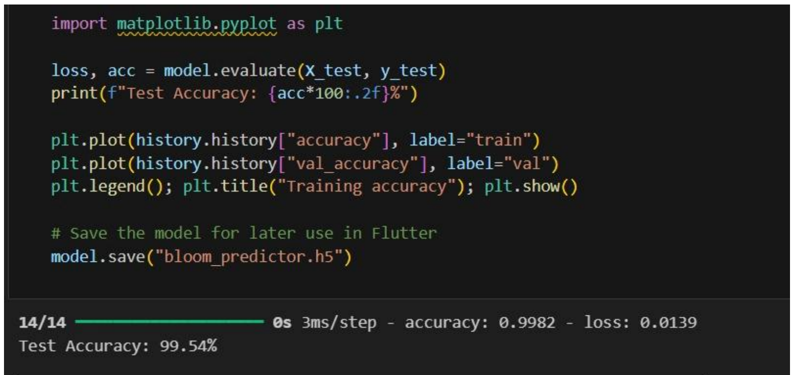
```python
import pandas as pd
import numpy as np

# Simulated dataset: each row is a region on a date with NDVI/EVI values
dates = pd.date_range("2023-01-01", periods=730, freq="D")  # 2 years
data = {
    "date": np.tile(dates, 3),
    "region": np.repeat(["Cairo", "Alexandria", "Luxor"], len(dates)),
    "NDVI": np.random.uniform(0.2, 0.9, 3 * len(dates)),
    "EVI": np.random.uniform(0.1, 0.8, 3 * len(dates)),
}
df = pd.DataFrame(data)

# Fake "blooming" label: 1 when NDVI > 0.6 and EVI > 0.5
df["blooming"] = ((df["NDVI"] > 0.6) & (df["EVI"] > 0.5)).astype(int)
df.head()
```

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = df[["NDVI", "EVI"]]
y = df["blooming"]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```



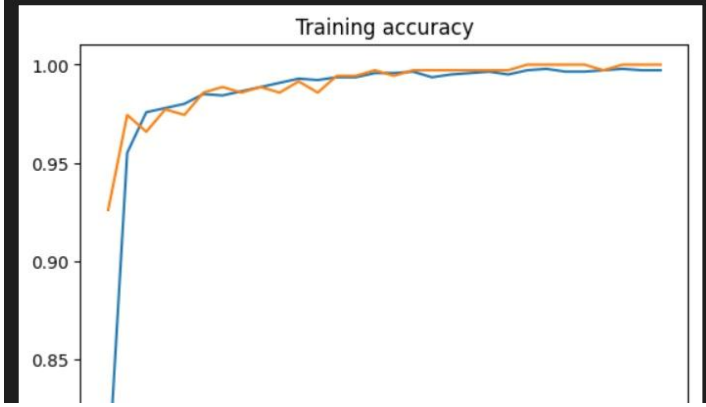
```python
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(16, activation="relu", input_shape=(2,)),
    layers.Dense(8, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=30, batch_size=16, validation_split=0.2)
```

```
Epoch 1/30
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
88/88 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.6922 - loss: 0.6526 - val_accuracy: 0.9259 - val_loss: 0.4900
Epoch 2/30
88/88 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9441 - loss: 0.4470 - val_accuracy: 0.9744 - val_loss: 0.2923
Epoch 3/30
88/88 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9747 - loss: 0.2700 - val_accuracy: 0.9658 - val_loss: 0.1660
Epoch 4/30
88/88 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9815 - loss: 0.1558 - val_accuracy: 0.9772 - val_loss: 0.1093
Epoch 5/30
88/88 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9809 - loss: 0.1040 - val_accuracy: 0.9744 - val_loss: 0.0860
Epoch 6/30
88/88 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9833 - loss: 0.0819 - val_accuracy: 0.9858 - val_loss: 0.0717
Epoch 7/30
88/88 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9815 - loss: 0.0778 - val_accuracy: 0.9886 - val_loss: 0.0624
Epoch 8/30
88/88 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9852 - loss: 0.0662 - val_accuracy: 0.9858 - val_loss: 0.0560
Epoch 9/30
88/88 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9920 - loss: 0.0506 - val_accuracy: 0.9886 - val_loss: 0.0500
```



```python
import matplotlib.pyplot as plt

loss, acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {acc*100:.2f}%")

plt.plot(history.history["accuracy"], label="train")
plt.plot(history.history["val_accuracy"], label="val")
plt.legend(); plt.title("Training accuracy"); plt.show()

# Save the model for later use in Flutter
model.save("bloom_predictor.h5")
```

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9982 - loss: 0.0139
Test Accuracy: 99.54%
```


Training accuracy