Wildfire Analysis: End-to-End Blueprint

✅ Phase 1: Core Pipeline (COMPLETED)

What We've Built:

1. **Data Extraction Module** (`data_extraction.py`)
   - Unified interface for VIIRS, ERA5, and DEM data
   - Spatial clipping and coordinate transformation
   - Feature extraction with error handling
2. **Configuration System** (`config_module.py`)
   - Centralized configuration management
   - Easy customization of parameters
   - Environment-based API key loading
3. **Transformer Model** (`transformer_model.py`)
   - PyTorch-based transformer architecture
   - Training pipeline with validation
   - Model checkpointing and loading
4. **Main Pipeline** (`main_pipeline.py`)
   - Command-line interface
   - Step-by-step or full pipeline execution
   - Support for both transformer and Random Forest
5. **Testing Suite** (`test_script.py`)
   - Comprehensive tests for all components
   - Synthetic data testing
   - End-to-end validation

🚀 Phase 2: Local Testing (NEXT STEP)

Tasks:

1. Setup Environment

```bash
# Create virtual environment
python -m venv wildfire-env
source wildfire-env/bin/activate  # On Windows: wildfire-env\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Verify installation
python test_script.py
```

2. Run Pipeline with Your Data

```bash
# Full pipeline with transformer
python main_pipeline.py --step full --model transformer

# Check outputs
ls outputs/
# Should see:
# - wildfire_features.csv
# - transformer_model.pth
# - wildfire_predictions.csv
```

3. Evaluate Results

```python
```

```
import pandas as pd

# Load predictions
df = pd.read_csv('outputs/wildfire_predictions.csv')

# Check accuracy
accuracy = (df['confidence_num'] == df['predicted_confidence']).mean()
print(f"Accuracy: {accuracy:.2%}")

# Confusion matrix
confusion = pd.crosstab(
    df['confidence'],
    df['predicted_confidence_label']
)
print(confusion)
```

4. Experiment with Hyperparameters

python

```
# In config_module.py, try:
model.d_model = 256  # Increase model capacity
model.num_layers = 6  # Deeper network
model.dropout = 0.2  # More regularization

# Then retrain
python main_pipeline.py --step train --model transformer
```

🌐 Phase 3: Open-Source Website

Architecture Overview

```
Frontend (React/Next.js)
    ↓
API Gateway (FastAPI)
    ↓
ML Service (PyTorch)
    ↓
Database (PostgreSQL + PostGIS)
```

3.1 Backend API (FastAPI)

File: `api/main.py`

python

```python
from fastapi import FastAPI, File, UploadFile
from fastapi.middleware.cors import CORSMiddleware
import torch
import pandas as pd
from transformer_model import WildfireTransformer, WildfireModelTrainer

app = FastAPI(title="Wildfire Prediction API")

# CORS for frontend
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
)

# Load model at startup
@app.on_event("startup")
async def load_model():
    global model, trainer
    model = WildfireTransformer(input_dim=5)
    trainer = WildfireModelTrainer(model)
    trainer.load_checkpoint('outputs/transformer_model.pth')

# Health check
@app.get("/")
async def root():
    return {"status": "online", "model": "wildfire_transformer"}

# Single prediction
@app.post("/predict")
async def predict(
    frp: float,
    u10: float,
    v10: float,
    elevation: float,
    slope: float
):
    features = [[frp, u10, v10, elevation, slope]]
    prediction = trainer.predict(features)[0]
    confidence_map = {0: "low", 1: "nominal", 2: "high"}

    return {
        "prediction": confidence_map[prediction],
```

```
        "confidence_score": int(prediction),
        "input_features": {
            "frp": frp,
            "wind_u": u10,
            "wind_v": v10,
            "elevation": elevation,
            "slope": slope
        }
    }

# Batch prediction
@app.post("/predict_batch")
async def predict_batch(file: UploadFile = File(...)):
    df = pd.read_csv(file.file)
    features = df[['frp', 'u10', 'v10', 'elevation', 'slope']].values
    predictions = trainer.predict(features)

    df['predicted_confidence'] = predictions
    return df.to_dict(orient='records')

# Model statistics
@app.get("/stats")
async def get_stats():
    return {
        "training_history": trainer.history,
        "model_parameters": {
            "d_model": 128,
            "num_layers": 4,
            "nhead": 8
        }
    }
```

Run API:

```
bash

uvicorn api.main:app --reload --port 8000
```

3.2 Frontend (React/Next.js)

File: `frontend/pages/index.js`

```
jsx
```

```
import { useState } from 'react';
import dynamic from 'next/dynamic';

// Lazy load map component
const MapComponent = dynamic(() => import('../components/Map'), {
  ssr: false
});

export default function Home() {
  const [prediction, setPrediction] = useState(null);
  const [loading, setLoading] = useState(false);

  const [formData, setFormData] = useState({
    frp: 10,
    u10: 2.5,
    v10: 3.0,
    elevation: 500,
    slope: 30
  });

  const handlePredict = async () => {
    setLoading(true);
    try {
      const response = await fetch('http://localhost:8000/predict', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(formData)
      });
      const data = await response.json();
      setPrediction(data);
    } catch (error) {
      console.error('Prediction failed:', error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="container">
      <h1>Wildfire Confidence Predictor</h1>

      <div className="input-panel">
        <label>
```

```
        Fire Radiative Power (MW):
        <input
          type="number"
          value={formData.frp}
          onChange={(e) => setFormData({...formData, frp: e.target.value})}
        />
      </label>

      <label>
        Wind U Component (m/s):
        <input
          type="number"
          value={formData.u10}
          onChange={(e) => setFormData({...formData, u10: e.target.value})}
        />
      </label>

      {/* More inputs... */}

      <button onClick={handlePredict} disabled={loading}>
        {loading ? 'Predicting...' : 'Predict Confidence'}
      </button>
    </div>

    {prediction && (
      <div className={`result ${prediction.prediction}`}>
        <h2>Prediction: {prediction.prediction.toUpperCase()}</h2>
        <p>Confidence Score: {prediction.confidence_score}/2</p>
      </div>
    )}

    <MapComponent predictions={prediction} />
  </div>
  );
}
```

3.3 Interactive Map Component

File: `frontend/components/Map.js`

jsx

```
import { MapContainer, TileLayer, Marker, Popup, CircleMarker } from 'react-leaflet';
import 'leaflet/dist/leaflet.css';

export default function Map({ fireData }) {
  const center = [34.05, -118.25];

  const getColor = (confidence) => {
    return {
      low: '#808080',
      nominal: '#FFA500',
      high: '#FF0000'
    }[confidence];
  };

  return (
    <MapContainer
      center={center}
      zoom={9}
      style={{ height: '500px', width: '100%' }}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; OpenStreetMap contributors'
      />

      {fireData && fireData.map((fire, idx) => (
        <CircleMarker
          key={idx}
          center={[fire.latitude, fire.longitude]}
          radius={fire.frp / 5}
          fillColor={getColor(fire.predicted_confidence)}
          color={getColor(fire.predicted_confidence)}
          fillOpacity={0.7}
        >
          <Popup>
            <strong>FRP:</strong> {fire.frp} MW<br/>
            <strong>Confidence:</strong> {fire.predicted_confidence}<br/>
            <strong>Elevation:</strong> {fire.elevation}m<br/>
            <strong>Slope:</strong> {fire.slope}°
          </Popup>
        </CircleMarker>
      ))}
    </MapContainer>
```

```
  );
}
```

3.4 Database Schema (PostgreSQL + PostGIS)

sql

```sql
-- Fire detections table
CREATE TABLE fire_detections (
    id SERIAL PRIMARY KEY,
    latitude DOUBLE PRECISION NOT NULL,
    longitude DOUBLE PRECISION NOT NULL,
    geometry GEOMETRY(Point, 4326),
    frp REAL,
    confidence TEXT,
    predicted_confidence TEXT,
    u10 REAL,
    v10 REAL,
    elevation REAL,
    slope REAL,
    detection_time TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Spatial index
CREATE INDEX idx_fire_geom ON fire_detections USING GIST (geometry);

-- Predictions history
CREATE TABLE predictions (
    id SERIAL PRIMARY KEY,
    fire_id INTEGER REFERENCES fire_detections(id),
    model_version TEXT,
    prediction TEXT,
    confidence_score INTEGER,
    created_at TIMESTAMP DEFAULT NOW()
);
```

3.5 Deployment (Docker)

File: `docker-compose.yml`

yaml

```yaml
version: '3.8'

services:
  # Database
  postgres:
    image: postgis/postgis:15-3.3
    environment:
      POSTGRES_DB: wildfire_db
      POSTGRES_USER: wildfire_user
      POSTGRES_PASSWORD: secure_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  # Backend API
  api:
    build: ./api
    command: uvicorn main:app --host 0.0.0.0 --port 8000
    volumes:
      - ./api:/app
      - ./outputs:/app/outputs
    ports:
      - "8000:8000"
    depends_on:
      - postgres
    environment:
      DATABASE_URL: postgresql://wildfire_user:secure_password@postgres:5432/wildfire_db

  # Frontend
  frontend:
    build: ./frontend
    command: npm run dev
    volumes:
      - ./frontend:/app
      - /app/node_modules
    ports:
      - "3000:3000"
    depends_on:
      - api

  # Nginx reverse proxy
  nginx:
```

```
    image: nginx:alpine
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    ports:
      - "80:80"
    depends_on:
      - frontend
      - api

volumes:
  postgres_data:
```

Deploy:

```
bash

docker-compose up -d
```

3.6 Features to Implement

Core Features:

- [ ] Real-time fire detection dashboard
- [ ] Historical fire data visualization
- [ ] Batch prediction upload (CSV)
- [ ] Interactive map with filters
- [ ] Model performance metrics
- [ ] Export predictions as GeoJSON

Advanced Features:

- [ ] User authentication (Firebase/Auth0)
- [ ] Custom model training interface
- [ ] Alert system for high-confidence fires
- [ ] Time-series analysis and trends
- [ ] Mobile app (React Native)
- [ ] API rate limiting and authentication

3.7 Project Structure

```
wildfire-website/
├── api/
│   ├── main.py              # FastAPI application
```

```
│   ├── models/              # ML model files
│   ├── database.py          # Database connection
│   ├── schemas.py           # Pydantic models
│   └── requirements.txt
│
├── frontend/
│   ├── pages/
│   │   ├── index.js         # Home page
│   │   ├── dashboard.js     # Analytics dashboard
│   │   └── upload.js        # Batch upload
│   ├── components/
│   │   ├── Map.js           # Interactive map
│   │   ├── PredictionForm.js
│   │   └── StatsPanel.js
│   ├── styles/
│   └── package.json
│
├── database/
│   └── init.sql             # Database initialization
│
├── docker-compose.yml
├── nginx.conf
└── README.md
```

📋 Implementation Checklist

Week 1: Local Testing & Optimization

- [x] Set up development environment
- [ ] Run full pipeline on your data
- [ ] Tune hyperparameters
- [ ] Document model performance
- [ ] Create sample predictions

Week 2: Backend Development

- [ ] Set up FastAPI project
- [ ] Implement prediction endpoints
- [ ] Add database integration
- [ ] Write API tests
- [ ] Deploy locally with Docker

Week 3: Frontend Development

- [ ] Set up Next.js project
- [ ] Create prediction form
- [ ] Integrate Leaflet map
- [ ] Connect to API
- [ ] Add error handling

Week 4: Integration & Deployment

- [ ] End-to-end testing
- [ ] Performance optimization
- [ ] Documentation
- [ ] Deploy to cloud (AWS/GCP/Heroku)
- [ ] Set up CI/CD pipeline

Week 5: Polish & Launch

- [ ] Add analytics
- [ ] Create demo video
- [ ] Write blog post
- [ ] Open-source release
- [ ] Community engagement

🎯 Success Metrics

Technical:

- API response time < 200ms
- Model accuracy > 85%
- 99.9% uptime
- Support 1000+ concurrent users

Community:

- 100+ GitHub stars in first month
- 10+ contributors
- 5+ forks
- Active Discord/Slack community

📚 Resources for Website Development

Learning:

- FastAPI Tutorial
- Next.js Documentation
- React Leaflet Examples
- Docker Compose Guide

Hosting Options:

1. **Free Tier Options:**
   - Frontend: Vercel, Netlify
   - Backend: Railway, Render
   - Database: Supabase, ElephantSQL
2. **Production Options:**
   - AWS (EC2 + RDS + S3)
   - Google Cloud Platform
   - DigitalOcean

🤝 Getting Help

Stuck on Local Testing?

- Check `test_script.py` output for specific errors
- Review logs in `test_pipeline.log`
- Verify data files with `ls -lh *.geojson *.nc *.tif`

Stuck on Website Development?

- Join our Discord server (create one!)
- Open GitHub issues for bugs
- Check Stack Overflow for common problems

🎉 Next Steps

1. **Right now**: Run `python test_script.py` to validate your setup
2. **Today**: Execute full pipeline and review predictions

3. **This week**: Start FastAPI backend
4. **Next week**: Begin frontend development
5. **This month**: Deploy public beta!

You're ready to build something amazing! 🚀