🌥 PM2.5 Air Quality Prediction Dashboard

Real-time PM2.5 monitoring and 24-hour prediction system using Transformer neural network, deployed on Railway with auto-updating hourly crawler.

📋 Features

- ✅ **Real-time PM2.5 Monitoring** - Live data from Taiwan EPA API
- 🔮 **24-Hour Predictions** - Transformer model forecasting
- 📊 **Interactive Visualizations** - Chart.js powered charts
- 💬 **AI Chatbot** - Gemini-powered RAG for historical queries
- ⏰ **Auto-Update** - Hourly crawler + model inference
- 🎨 **Modern UI** - Glassmorphism design, light mode

🏗 Architecture

- **Frontend**: HTML/CSS/JavaScript + Chart.js
- **Backend**: Flask + APScheduler
- **Database**: SQLite with Railway Volume (ALL data 2018-2025, ~61,000+ hours)
- **Model**: Transformer (uses last 720h → predicts 24h)
- **RAG**: Gemini API (queries full 2018-2025 history)
- **Deployment**: Railway with Docker

📁 Project Structure

```
pm25_dashboard/
├── backend/
│   ├── app.py              # Flask main app
│   ├── config.py           # Configuration
│   ├── database.py         # SQLite utilities
│   ├── init_db.py          # Initialize DB from CSV
│   ├── crawler.py          # EPA API crawler
│   ├── prediction_service.py  # Model inference
│   ├── rag_service.py      # Gemini chatbot
│   └── scheduler.py        # APScheduler
├── frontend/
```

```
│   ├── index.html
│   └── static/
│       ├── css/style.css
│       └── js/
│           ├── dashboard.js
│           └── chat.js
├── models/
│   └── best_model.keras    # YOUR TRAINED MODEL
├── data/
│   └── pm25_data.db        # Historical database
├── Dockerfile
├── railway.toml
├── requirements.txt
└── README.md
```

🚀 Deployment Instructions

Step 1: Prepare Your Data & Model

1. **Initialize database with historical data** (run locally):

```bash
python -m backend.init_db --csv /path/to/all_pm25_data.csv
```

Expected output:

```
📊 Data Range:
Start: 2018-01-01 00:00
End: 2025-11-23 14:00
Total: 61,320 hours (2,555 days)

✅ Database initialized successfully!
Total measurements: 61,320
Database size: ~8 MB
```

2. **Copy your trained model**:

```bash
cp /path/to/best_model.keras models/
```

3. **Ensure database is in repository**:
    - The file `data/pm25_data.db` should be in your repo
    - Make sure it's NOT in `.gitignore`

Step 2: Get API Keys

1. **Taiwan EPA API Key**:
    - Visit: https://data.moenv.gov.tw/
    - Register account → Get API key
2. **Google Gemini API Key**:
    - Visit: https://makersuite.google.com/app/apikey
    - Create API key

Step 3: Railway Setup

3.1 Create Railway Account

1. Go to https://railway.app
2. Sign up with GitHub (required for verification)
3. Verify your account to enable Full Trial

3.2 Create New Project

1. Click "New Project"
2. Select "Deploy from GitHub repo"
3. Connect your GitHub account
4. Select your PM2.5 project repository

3.3 Configure Environment Variables

In Railway dashboard → **Variables** tab, add:

```env
```

```
EPA_API_KEY=your_epa_api_key_here
GEMINI_API_KEY=your_gemini_api_key_here
SECRET_KEY=your_random_secret_key
DATABASE_PATH=/data/pm25_data.db
MODEL_PATH=./models/best_model.keras
SITE_NAME=板橋
TZ=Asia/Taipei
FLASK_ENV=production
RAILWAY_RUN_UID=0
```

Important: `RAILWAY_RUN_UID=0` is required for volume write permissions.

3.4 Create Volume

Railway volumes provide persistent storage that survives container restarts.

1. In Railway dashboard, click **"+ New"** button
2. Select **"Volume"**
3. Configure:
   - **Mount Path**: `/data`
   - **Name**: `pm25-data` (or any name you prefer)
4. Attach the volume to your service

Note: If you can't use Cmd+K shortcut, use the "+ New" button in the Railway UI instead.

Step 4: Deploy

1. Railway will automatically detect `Dockerfile`
2. Click **"Deploy"** (or push to GitHub to auto-deploy)
3. Wait for build (~3-5 minutes)
4. Monitor build logs for any errors

Step 5: Generate Public Domain

1. Go to **Settings → Networking**
2. Click **"Generate Domain"**
3. Railway will provide a public URL like:

```
https://pm25-prediction-project-production.up.railway.app
```

Step 6: Verify Deployment

1. Visit your Railway URL
2. Check **Deploy Logs** tab for:

```
📦 Initializing database from backup...
✅ Database initialized!
⏰ SCHEDULER STARTED
📡 Fetching data from EPA API...
✅ Model loaded successfully
🚀 Starting Flask app on port 8080...
```

3. Test the dashboard:
   - Current PM2.5 displays
   - Historical charts load
   - 24-hour predictions show
   - Chatbot responds (try: "What was PM2.5 in 2020?")

🔄 How It Works

Hourly Cycle

```
00:00 → Crawler fetches new data from EPA API
    → Clean & forward-fill missing values
    → Insert into SQLite (preserves ALL historical data)
    → Model: Read last 720 hours → Predict 24 hours
    → Store predictions in database
    → Frontend auto-refreshes via JavaScript
```

Data Flow

```
User → Railway (Flask API) → SQLite Database (ALL historical data)
        ↓
    APScheduler (Hourly)
        ↓
    EPA API Crawler → Model Inference (last 720h)
```

📊 Model Details

Architecture Evolution

Initial Model: Single-Layer Transformer

- Architecture: 1 Transformer Block
- Embedding Dim: 64
- Feed-Forward Dim: 256
- Total Parameters: 177,560 (~693 KB)
- Validation Loss: ~0.0045

Final Model: Dual-Layer Transformer

- Architecture: 2 Stacked Transformer Blocks
- Embedding Dim: 128
- Feed-Forward Dim: 512
- Total Parameters: 1,358,872 (~5.18 MB)
- Validation Loss: 0.00393
- **Performance Improvement**: 15% reduction in validation loss

Model Components

1. **Projection Layer**: Maps 1D input to 128-dimensional feature space
2. **Positional Encoding**: Sinusoidal encoding for temporal information
3. **Transformer Encoder Block 1**:
   - Multi-Head Attention (8 heads, key_dim=16)
   - Feed-Forward Network (128 → 512 → 128)
   - Residual Connections & Layer Normalization
4. **Transformer Encoder Block 2**: Same structure as Block 1
5. **Global Average Pooling**: Aggregate temporal features
6. **Dense Output Layer**: 128 → 24 (predict 24 hours)

Training Results

Performance Metrics:

- Overall R^2: 0.4166
- Overall MAE: 4.60 µg/m^3
- Overall RMSE: 6.33 µg/m^3
- Hour +1 prediction: R^2=0.80, MAE=2.73 µg/m^3
- Hour +24 prediction: R^2=0.25, MAE=5.25 µg/m^3

Training Details:

- Training Time: 258.81 minutes (31 epochs)
- Average: 8.0 min/epoch
- Hardware: Google Colab T4 GPU
- Optimizer: Adam (lr=1e-3)
- Loss Function: MSE
- Early Stopping: Patience=5 epochs

Input/Output Specification

- **Input**: (batch_size, 720, 1) - 720 hours of PM2.5 data
- **Output**: (batch_size, 24) - 24 hours of predictions
- **Lookback Window**: 30 days (720 hours)
- **Forecast Horizon**: 24 hours

📝 Notes

- System requires at least 720 hours (30 days) of data for predictions
- All historical data preserved (2018-2025, ~61,000+ hours)
- Chatbot can query entire historical dataset via RAG
- Hourly auto-update via EPA API crawler

Good luck! 🚀