

# config.py - Configuration Management

---

## Overview

---

This module centralizes all configuration settings for the solar panel detection system. It loads values from environment variables with sensible defaults, providing a single source of truth for system parameters.

---

## Logic

---

### Class: Config

---

A static configuration class with class attributes and class methods - no instantiation required.

### Configuration Categories

---

Category	Settings
API	GOOGLE_MAPS_API_KEY
Model	MODEL_PATH, CONFIDENCE_THRESHOLD
Directories	INPUT_FOLDER, OUTPUT_FOLDER, INPUT_FILENAME
Image	MAP_REQUEST_SIZE, MAP_SCALE, FINAL_IMAGE_SIZE, ZOOM_LEVEL
Detection	SLICE_HEIGHT, SLICE_WIDTH, OVERLAP_RATIO
Buffers	BUFFER_RADIUS_1_SQFT, BUFFER_RADIUS_2_SQFT
Quality Control	BRIGHTNESS_THRESHOLD_*, CLOUD_THRESHOLD, MIN_IMAGE_VARIANCE

---

# How It Works

---

## 1. Environment Variable Loading

---

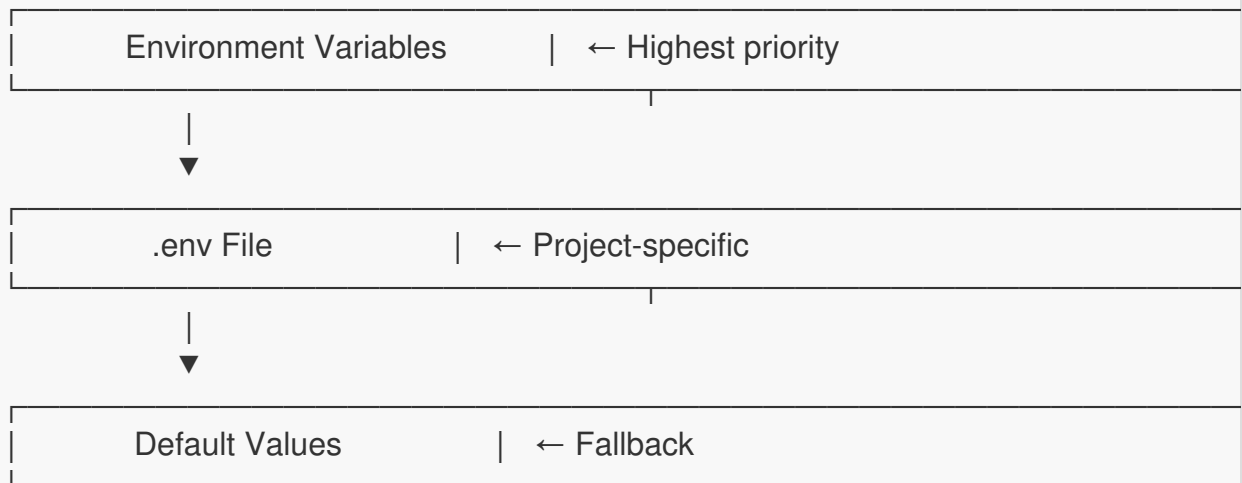
```
from dotenv import load_dotenv
load_dotenv() # Loads .env file at module import

GOOGLE_MAPS_API_KEY = os.getenv('GOOGLE_MAPS_API_KEY', '')
ZOOM_LEVEL = int(os.getenv('ZOOM_LEVEL', 20))
```

The `load_dotenv()` call reads the `.env` file and populates `os.environ`. Each setting uses `os.getenv()` with a fallback default value.

## 2. Configuration Hierarchy

---



## 3. Key Methods

---

`validate()`

Ensures critical settings are properly configured:

```
@classmethod
def validate(cls):
    if not cls.GOOGLE_MAPS_API_KEY:
        raise ValueError("Google Maps API Key not configured")

    if not Path(cls.MODEL_PATH).exists():
        raise FileNotFoundError(f"Model not found: {cls.MODEL_PATH}")
```

**setup\_directories()**

Creates required folders if they don't exist:

```
@classmethod
def setup_directories(cls):
    cls.INPUT_FOLDER.mkdir(exist_ok=True)
    cls.OUTPUT_FOLDER.mkdir(exist_ok=True)
```

**get\_model\_path()**

Handles model file fallback logic:

```
@classmethod
def get_model_path(cls):
    if Path(cls.MODEL_PATH).exists():
        return cls.MODEL_PATH

    fallback = cls.MODEL_PATH.replace("best.pt", "last.pt")
    if Path(fallback).exists():
        return fallback
```

---

## Why It Works

---

### Separation of Concerns

---

- **Code** contains logic, not configuration values
- **Environment** controls deployment-specific settings
- **Defaults** ensure the system works out of the box

### Type Conversion

---

Environment variables are always strings. The config handles conversion:

```
CONFIDENCE_THRESHOLD = float(os.getenv('CONFIDENCE_THRESHOLD', 0.25))
ZOOM_LEVEL = int(os.getenv('ZOOM_LEVEL', 20))
```

### Buffer Zone Design

---

Two buffer radii (1200 and 2400 sq.ft) create concentric zones:

- **Inner buffer** (1200 sq.ft) - High confidence match
- **Outer buffer** (2400 sq.ft) - Extended search area

## Quality Control Thresholds

---

```
BRIGHTNESS_THRESHOLD_LOW = 30    # Reject dark/shadowed images
BRIGHTNESS_THRESHOLD_HIGH = 225  # Detect overexposed/cloudy areas
CLOUD_THRESHOLD = 0.7            # Max 70% bright pixels allowed
MIN_IMAGE_VARIANCE = 100         # Reject blank/uniform images
```

## Usage in Main Pipeline

---

### Initialization

---

```
# In pipeline.py
from .config import Config

class SolarDetectionPipeline:
    def __init__(self):
        self.config = Config

        self.maps_client = GoogleMapsClient(
            api_key=Config.GOOGLE_MAPS_API_KEY,
            zoom_level=Config.ZOOM_LEVEL,
            ...
        )
```

### Pipeline Startup

---

```
def run(self):  
    Config.validate()    # Fail fast on misconfig  
    Config.setup_directories() # Ensure folders exist  
  
    input_path = Config.INPUT_FOLDER / Config.INPUT_FILENAME  
    ...
```

## Configuration Flow

---

```
main.py  
|  
▼ imports  
pipeline.py  
|  
▼ uses  
Config (config.py)  
|  
▼ reads  
.env file
```

All pipeline components access Config directly, ensuring consistent settings across:

- API client
- Quality checker
- Detector
- Geometry calculations
- Visualizer