pipeline.py - Main Detection Pipeline Orchestrator

Overview

This is the core orchestration module that ties all components together. It manages the complete workflow from loading input data, fetching satellite imagery, running detection, and saving results.

Logic

Class: SolarDetectionPipeline

Method	Purpose
__init__()	Initialize all pipeline components
run()	Execute the complete detection workflow
_process_sample()	Process a single location
_find_best_detection()	Match detections to buffer zones
_load_or_create_input()	Load Excel input or create sample
_save_combined_results()	Save all results to JSON
_print_header() / _print_summary()	Console output

Component Dependencies

```
┌──────────────────────────────────────────────────────────────
│                SolarDetectionPipeline                    │
│                                              │
│
│  ┌─────────────────────────────┐  ┌───────────────────────────────┐  ┌─
│  │ GoogleMaps   │   │ ImageQuality   │   │ SolarPanel      │   │
│  │ Client       │   │ Checker        │   │ Detector        │   │
│  └─────────────────────────────┘  └───────────────────────────────┘  └─
│                                              │
│
│  ┌─────────────────────────────┐  ┌─────────────────────────────────
│  │ Detection    │   │ Geometry Functions                    │  │
│  │ Visualizer   │   │ (get_meters_per_pixel, find_best_panel)  │  │
│  └─────────────────────────────┘  └─────────────────────────────────
│                                              │
│
│  ┌──────────────────────────────────────────────────────────────
│  │ Config (shared configuration)                        │   │
│  └──────────────────────────────────────────────────────────────
│
└──────────────────────────────────────────────────────────────
```

How It Works

1. Pipeline Initialization

```python
def __init__(self):
    self.config = Config

    self.maps_client = GoogleMapsClient(...)
    self.quality_checker = ImageQualityChecker(...)
    self.detector = SolarPanelDetector(...)
    self.visualizer = DetectionVisualizer()
```

All components are instantiated with configuration values but **not yet activated** (model not loaded).

2. Execution Flow (run())

```
┌──────────────────────────────────────────────────────────────────
│                    run()                          │
├──────────────────────────────────────────────────────────────────
│                                                   │
│  1. _print_header()        Print startup banner                │
│                        ↓                          │
│  2. Config.validate()      Check API key & model exist         │
│                        ↓                          │
│  3. Config.setup_directories() Create input/output folders     │
│                        ↓                          │
│  4. _load_or_create_input()  Load Excel file with coordinates  │
│                        ↓                          │
│  5. detector.initialize()   Load YOLOv8 model (once)           │
│                        ↓                          │
│  6. FOR each row in Excel:  ──────────────────────────────────────
│       │                                         │    │
│       └────► _process_sample()  Download, check, detect, save  │    │
│                        ↓                      ↓   │
│  7. _save_combined_results() Write all_predictions.json        │
│                        ↓                          │
│  8. _print_summary()        Show final statistics              │
│                                                   │
└──────────────────────────────────────────────────────────────────
```

3. Sample Processing (_process_sample())

```python
def _process_sample(self, row, current, total):
    sample_id = row['sample_id']
    lat, lon = row['latitude'], row['longitude']

    # Create output folder
    sample_folder = Config.OUTPUT_FOLDER / str(sample_id)
    sample_folder.mkdir(parents=True, exist_ok=True)

    # Step 1: Download satellite image
    image_path = self.maps_client.download_satellite_image(lat, lon, ...)

    # Step 2: Quality check
    is_verifiable, quality_reason = self.quality_checker.check_quality(image_path)

    # Step 3: Run detection
    all_polygons = self.detector.detect(image_path)

    # Step 4: Match to buffer zones
    result_data = self._find_best_detection(all_polygons, ...)

    # Step 5: Save JSON result
    with open(json_path, 'w') as f:
        json.dump(output_record, f, indent=4)

    # Step 6: Create visualization
    self.visualizer.draw_results(...)

    return output_record
```

4. Buffer Zone Matching (_find_best_detection())

```
┌──────────────────────────────────────────────────────────────
│              Buffer Zone Strategy                │
│                                             │
│   Detected panels: ■■■■■                       │
│                                             │
│         ┌─────────────────────────────────────┐                        │
│         │  2400 sq.ft buffer    │                   │
│         │   ┌───────────────────────────┐   │                   │
│         │   │  1200 sq.ft      │    │                   │
│         │   │    ■ ← Found!    │    │                   │
│         │   │              │    │                   │
│         │   └───────────────────────────┘   │                   │
│         │      ■           │   ← Would check here      │
│         └─────────────────────────────────────┘   only if 1200 empty
│                                             │
└──────────────────────────────────────────────────────────────
```

Logic:

1. Try 1200 sq.ft buffer first (higher confidence)
2. If no panel found, expand to 2400 sq.ft
3. Select panel with **maximum overlap area**
4. Record which buffer zone was used

5. Output Structure

```
predictions/
├── 1001/
│   ├── 1001.jpg          (satellite image)
│   ├── 1001.json         (detection result)
│   └── 1001_overlay.png   (visualization)
├── 1002/
│   └── ...
└── all_predictions.json   (combined results)
```

Sample JSON Output:


```json
{
  "sample_id": 1001,
  "lat": 23.908454,
  "lon": 71.182617,
  "has_solar": true,
  "confidence": 0.8742,
  "pv_area_sqm_est": 12.45,
  "buffer_radius_sqft": 1200,
  "qc_status": "VERIFIABLE",
  "bbox_or_mask": "[[512, 480], [580, 480], ...]",
  "image_metadata": {
    "source": "Google Maps Static API",
    "size": "1024x1024",
    "meters_per_px": 0.06912,
    "quality_check": "Good quality"
  }
}
```

Why It Works

Modular Architecture

Each component has a single responsibility:

- **API Client**: Fetch imagery
- **Quality Checker**: Validate imagery
- **Detector**: Find panels
- **Geometry**: Spatial math
- **Visualizer**: Create overlays

This separation enables:

- Independent testing
- Easy component swapping

- Clear debugging

Two-Stage Buffer Strategy

Real-world property boundaries aren't precise. The two-buffer approach:

- **1200 sq.ft**: Confident the panel belongs to this property
- **2400 sq.ft**: Catches edge cases (property line errors, GPS drift)

Fail-Safe Processing

```python
if not image_path:
    return None  # Skip bad downloads

# Detection runs even on low-quality images
# Quality status is recorded, not blocking
```

This ensures:

- One failed sample doesn't crash the pipeline
- All data is collected for analysis
- Quality flags enable post-processing filtering

Excel Input with Auto-Generation

```python
def _load_or_create_input(self, input_path):
    if not input_path.exists():
        # Create sample file with demo coordinates
        df = pd.DataFrame({
            'sample_id': [1001, 1002],
            'latitude': [23.908454, 28.7041],
            'longitude': [71.182617, 77.1025]
        })
        df.to_excel(input_path, index=False)
```

New users get a working example immediately.

Usage in Main Application

Entry Point (main.py)

```python
from src.pipeline import SolarDetectionPipeline

def main():
    setup_logging()

    try:
        pipeline = SolarDetectionPipeline()
        pipeline.run()
    except KeyboardInterrupt:
        print("Pipeline interrupted by user")
    except Exception as e:
        logging.error(f"Pipeline failed: {e}", exc_info=True)
        raise

if __name__ == "__main__":
    main()
```

Full Execution Flow

```
main.py
    |
    ▼ creates
SolarDetectionPipeline
    |
    ▼ calls
run()
    |
    ├───► Config.validate()
    ├───► detector.initialize()
    |
    ▼ for each sample
_process_sample()
    |
    ├───► maps_client.download_satellite_image()
    ├───► quality_checker.check_quality()
    ├───► detector.detect()
    ├───► geometry functions
    └───► visualizer.draw_results()
```

Command Line Usage

```
python main.py
```

Reads input_folder/input_data.xlsx, outputs to predictions/.