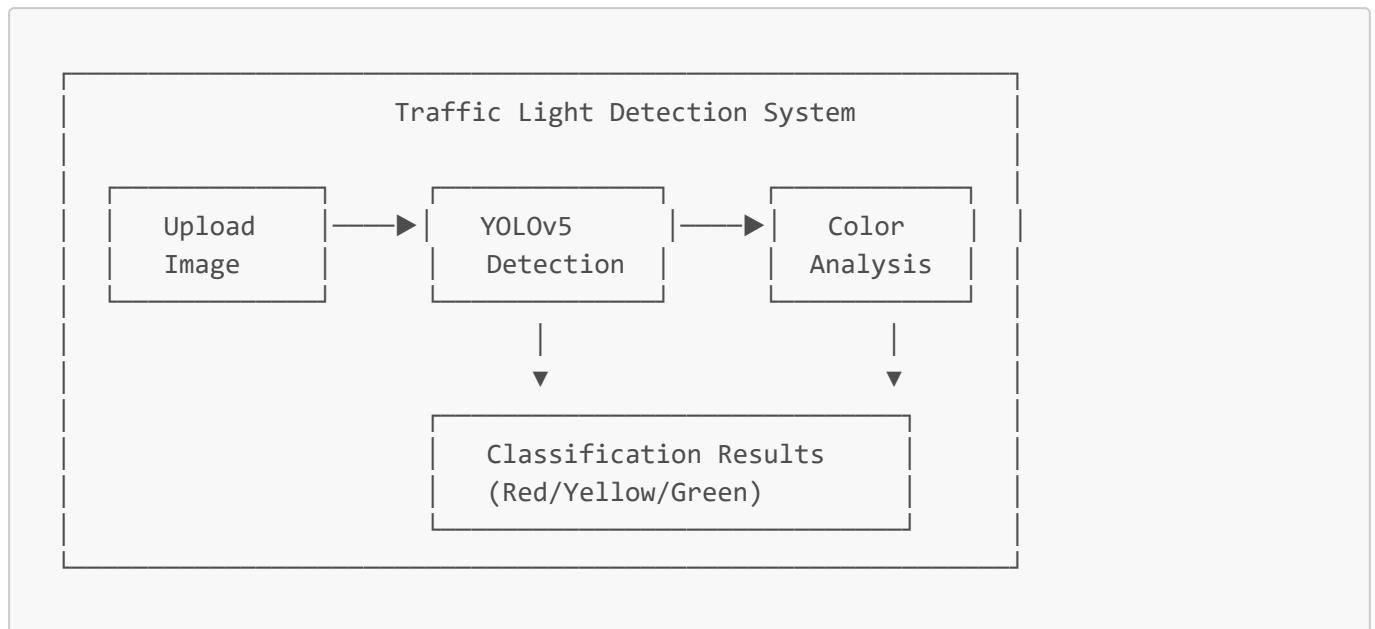


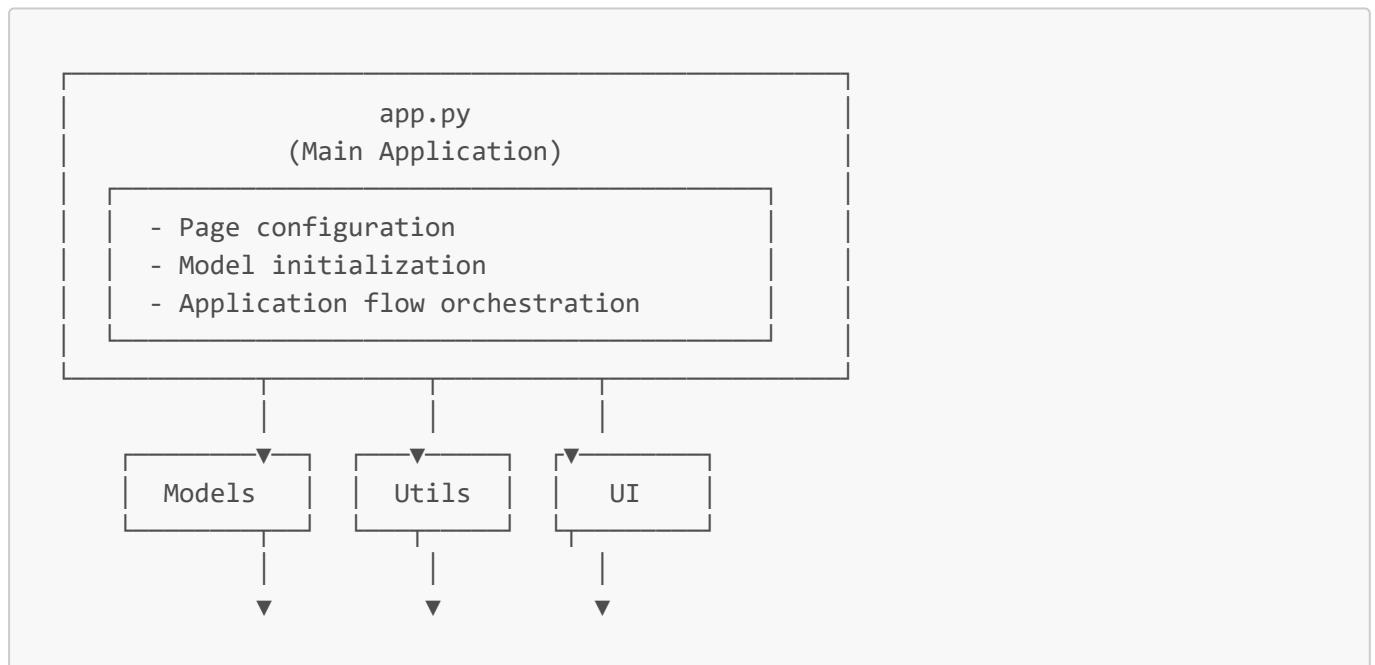
Project Architecture

System Overview



Module Architecture

Core Components



1. Models Package ([src/models/](#))

```
YOLOModelHandler
├── __init__.py
├── _load_model()          # Load YOLOv5 with caching
└── detect(image)          # Perform detection
```

```
|__ is_loaded()          # Check model status
  __ get_model_info()    # Model metadata
```

Purpose: Manages YOLO model lifecycle and inference

2. Utils Package ([src/utils/](#))

detection.py

```
# Color Detection Functions
|__ detect_traffic_light_color(image, box)
  __ _count_red_pixels(hsv_image)
  __ _count_green_pixels(hsv_image)
  __ _count_yellow_pixels(hsv_image)
  __ _determine_dominant_color(r, g, y)
  __ validate_color_ranges()
```

image_processing.py

```
# Detection Processing
|__ DetectionResult (class)
  __ process_detection_results(output, image)
  __ get_detection_summary(results)
  __ filter_results_by_color(results, color)
  __ get_highest_confidence_detection(results)
```

Purpose: Core detection algorithms and result processing

3. UI Package ([src/ui/](#))

components.py

```
# UI Components
|__ render_header()
  __ render_about_section()
  __ render_sidebar()
  __ render_upload_section()
  __ render_detection_result(result)
  __ render_detection_results(results)
  __ render_summary_statistics(summary)
  __ render_annotated_image(image, width)
```

styles.py

```
# Styling
└── apply_custom_styles()
    ├── Traffic light color cards
    ├── Button styling
    └── Layout styling
```

Purpose: User interface rendering and styling

4. Config Package ([config/](#))

```
# config.py
├── Application Settings
│   ├── APP_TITLE
│   ├── APP_ICON
│   └── PAGE_LAYOUT

├── Model Configuration
│   ├── MODEL_NAME
│   ├── MODEL_REPO
│   └── CONFIDENCE_THRESHOLD

├── Color Ranges (HSV)
│   ├── COLOR_RANGES["red"]
│   ├── COLOR_RANGES["green"]
│   └── COLOR_RANGES["yellow"]

└── Display Settings
    ├── COLOR_EMOJIS
    └── COLOR_MESSAGES
```

Purpose: Centralized configuration and constants

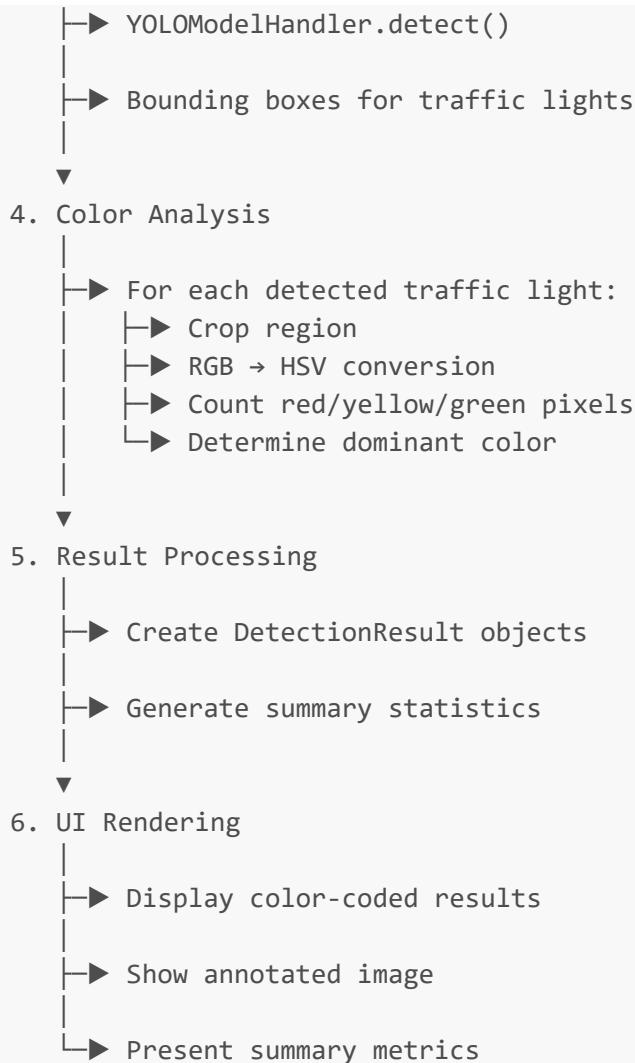
Data Flow

Detection Pipeline

1. User Upload
 - Image File (PNG/JPG)

2. Image Loading
 - PIL.Image → NumPy Array

3. YOLO Detection



🔒 Design Patterns

1. Separation of Concerns

- Models: AI/ML logic
- Utils: Business logic
- UI: Presentation logic
- Config: Configuration data

2. Caching Strategy

```
@st.cache_resource
def _load_model():
    # Model loaded once and cached
    # Improves performance significantly
```

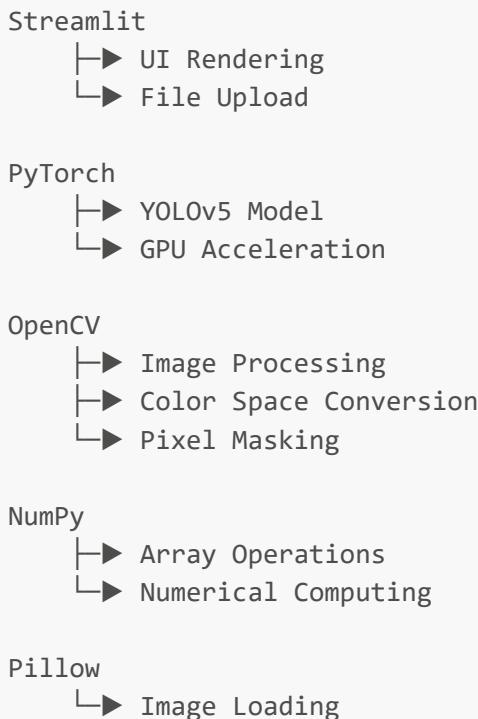
3. Error Handling

```
try:  
    # Attempt operation  
except Exception as e:  
    logger.error(f"Error: {e}")  
    st.error("User-friendly message")
```

4. Modular Functions

- Single responsibility
- Clear inputs/outputs
- Well documented
- Testable

📦 Dependencies Graph



🎯 Extension Points

Adding New Features

1. New Detection Algorithm

- Add function to `src/utils/detection.py`
- Update `config/config.py` if needed
- Create tests in `tests/`

2. New UI Component

- Add function to `src/ui/components.py`

- Update styles in `src/ui/styles.py`
- Call from `app.py`

3. Different Model

- Modify `src/models/yolo_model.py`
- Update `config/config.py`
- Adjust detection parameters

4. Additional Colors

- Add ranges to `config/config.py`
- Extend `detection.py` functions
- Update UI components

Best Practices Used

- Modular architecture
- Type hints and docstrings
- Logging for debugging
- Configuration management
- Error handling
- Code reusability
- Clear naming conventions
- Professional structure