ThriftAssist System Block Diagram

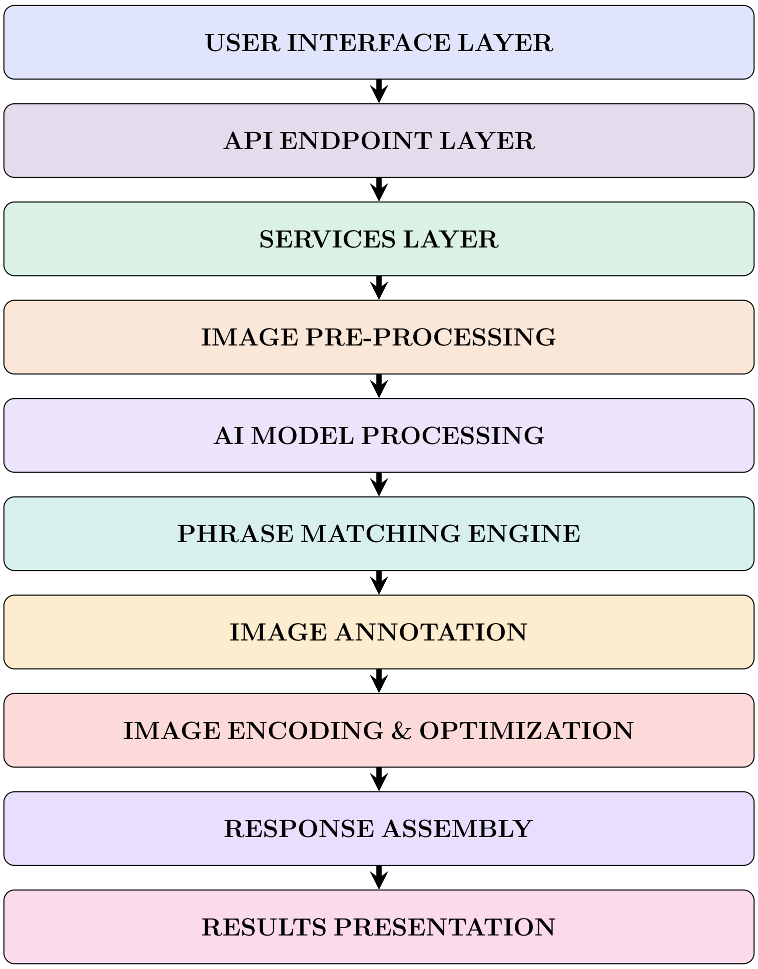
System Architecture Documentation

December 6, 2025

System Architecture Overview

ThriftAssist is a comprehensive OCR-based phrase detection system that identifies specific text phrases in images using Google Cloud Vision API, applies intelligent multi-strategy matching, annotates results visually, and presents them through a responsive web interface.

1 High-Level Architecture



2 Component Details

2.1 User Interface Layer

Location: `public/web_app.html`

UI Components

- **Input Form**
  - File upload (single/batch mode)
  - Search phrases input
  - Threshold slider (50-100%)
  - Text scale control
- **Gallery View**
  - Thumbnail preview
  - Multi-select capability
  - Image selection controls
  - Add/remove images
- **Results Display**
  - Match summary statistics
  - Confidence metrics
  - Explainability panels
  - Annotated image with zoom/pan

2.2 API Endpoint Layer

Location: `backend/api/routes/ocr.py`

/upload-and-detect Endpoint

Request Processing Pipeline:

1. Validate input (file type, size, parameters)
2. Generate cache key (MD5 hash of image + text_scale)
3. Check cache for existing result
   - If cache hit → Return cached result immediately
   - If cache miss → Continue processing
4. Save temporary image file
5. Resize image if width > max_image_width
6. Invoke OCR processing
7. Format match results
8. Encode annotated image to base64
9. Cache result for future requests
10. Return JSON response

2.3 Services Layer

Location: `backend/services/`

Service Components

Image Service	Validates image format, decodes base64, saves temporary files, manages cleanup
Cache Service	Implements MD5 hashing, LRU cache with TTL, automatic eviction, cache hit/miss tracking
OCR Service	Orchestrates detection pipeline, formats results for API, measures processing time, coordinates with Vision API

2.4 Image Pre-Processing

Location: OpenCV/PIL Pipeline

Pre-Processing Steps

1. Load image using `cv2.imread()`
2. Check dimensions against `max_image_width`
3. Resize if necessary (maintaining aspect ratio)
4. Optimize for display performance
5. Save as temporary file for Vision API

2.5 AI Model Processing

Location: `vision/detector.py`

Google Cloud Vision API Integration

VisionPhraseDetector.detect() Process:

1. **Call Google Vision API**
   - Execute `document_text_detection()`
   - Extract full text annotations
   - Retrieve bounding boxes and vertices
2. **Parse Text Lines**
   - Extract text content from annotations
   - Calculate Y-position (vertical location)
   - Determine rotation angle
   - Store bounding box coordinates
3. **Build Text Structure**
   - Sort lines by Y-position
   - Group multi-line text blocks
   - Preserve spatial relationships

2.6 Phrase Matching Engine

Location: `vision/matcher.py`

PhraseMatcher.find_matches()

Multi-Strategy Matching:

A. **EXACT MATCHING**
   - Case-insensitive comparison
   - Direct string match
   - 100% confidence score

B. **FUZZY MATCHING (RapidFuzz)**
   - Levenshtein distance calculation
   - Partial ratio scoring
   - Token-based matching
   - Similarity score (0-100)

C. **MULTI-LINE SPANNING**
   - Combine adjacent text lines
   - Match phrases split across lines
   - Window-based search (2-3 lines)

D. **UPSIDE-DOWN DETECTION**
   - Check rotation angles ($\pm 180^\circ$)
   - Reverse text comparison
   - Match inverted text

E. **THRESHOLD FILTERING**
   - Apply user-defined threshold (50-100%)
   - Filter low-confidence matches
   - Rank by similarity score

F. **DEDUPLICATION**
   - Remove duplicate matches
   - Merge overlapping bounding boxes
   - Keep highest-confidence match

2.7 Image Annotation

Location: `vision/detector.py + OpenCV`

Annotation Process

1. **Draw Bounding Boxes**
   - Green rectangles around matches
   - Line thickness based on confidence
   - Color intensity varies by score
2. **Add Text Labels**
   - Display match phrase text
   - Show confidence percentage
   - Font size scaled by `text_scale` parameter
   - Positioned above/below bounding box
   - Mobile boost: 4× text scale for mobile devices
3. **Handle Overlaps**
   - Adjust label positions to avoid overlap
   - Layer annotations by confidence
4. **Maintain Image Quality**
   - Preserve original aspect ratio
   - Anti-aliasing for smooth rendering

2.8 Image Encoding & Optimization

Location: `backend/utils/image_utils.py`

Encoding Pipeline

1. **Convert to PIL Image**
   - OpenCV BGR → RGB conversion
   - Maintain numpy array structure
2. **Calculate Optimal JPEG Quality**
   - Based on image dimensions
   - Larger images: lower quality (60-75)
   - Smaller images: higher quality (85-95)
   - Balance file size vs visual quality
3. **Encode to Base64**
   - JPEG compression with calculated quality
   - Optimization enabled
   - Base64 encoding for JSON transport
4. **Memory Management**
   - Immediate cleanup of temporary objects
   - Garbage collection after encoding

2.9 Response Assembly

Location: `backend/api/routes/ocr.py`

JSON Response Structure

```
{
  "success": true,
  "total_matches": ⟨count⟩,
  "matches": {
    "phrase1": [
      {
        "text": "matched text",
        "score": 95.5,
        "match_type": "exact|fuzzy|spanning|upside_down",
        "angle": 0,
        "bounding_box": [[x1,y1], [x2,y2], ...],
        "explanation": {
          "confidence_level": "Very High|High|Medium|Low",
          "reasoning": ["reason1", "reason2"],
          "recommendation": "explanation text"
        }
      }
    ]
  },
  "processing_time_ms": 1234.5,
  "image_dimensions": [width, height],
  "annotated_image_base64": "base64_encoded_jpeg",
  "cached": false
}
```

2.10 Results Presentation (UI)

Location: `public/web_app.html`

UI Presentation Components

Results Summary
Display total matches, processing time, average confidence, image dimensions

Match Details Collapsible cards grouped by search phrase with:

- Individual match items
- Matched text preview
- Confidence badges (Very High, High, Medium, Low)
- Similarity score percentage
- Rotation angle indicator

Explainability "Why?" button reveals:

- Confidence level explanation
- Reasoning factors list
- Match type description
- User recommendations

Annotated Image Display
Features:

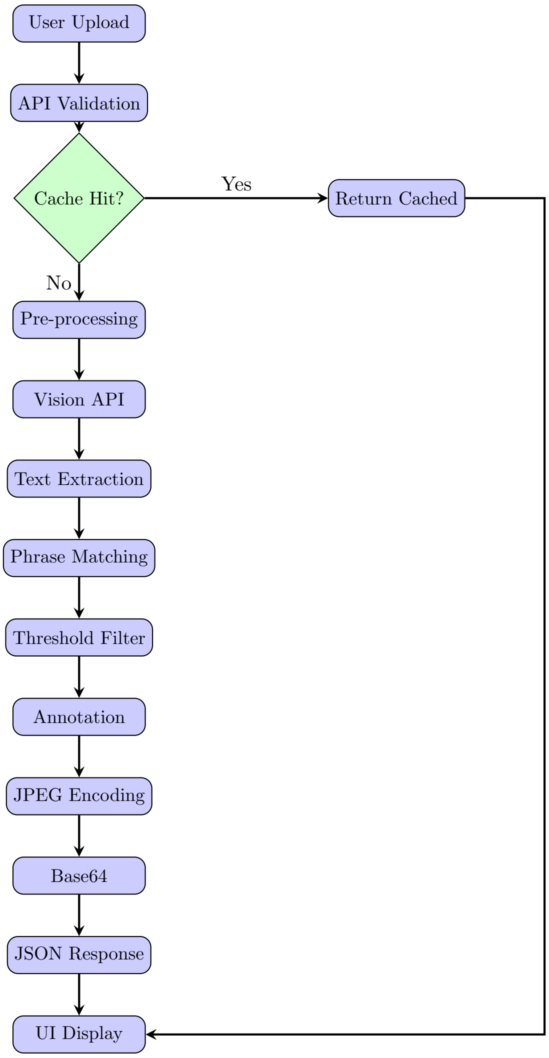
- Base64 decode → data URL
- Responsive sizing (mobile/tablet/desktop)
- Aspect ratio preservation
- Zoom & pan controls (Panzoom.js)
- Touch gesture support
- Device-specific optimizations

Batch Mode Results
For multiple images:

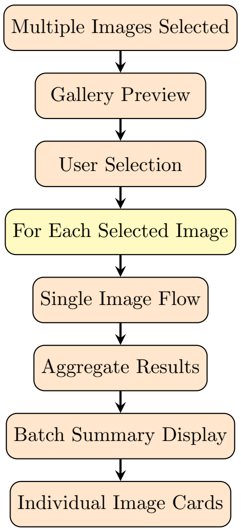
- Batch summary (total images, success count)
- Individual image result cards
- Per-image annotated display
- Click to enlarge (modal view)

3 Data Flow Diagrams

3.1 Single Image Processing Flow



3.2 Batch Processing Flow



4 Key Performance Features

Performance Optimizations

1. **Caching Layer**
   - MD5-based LRU cache with TTL
   - Avoids redundant OCR processing
   - Significant performance improvement for repeated queries
2. **Image Optimization**
   - Automatic resizing for large images
   - Dynamic JPEG quality adjustment
   - Optimal balance between quality and transfer size
3. **Memory Management**
   - Aggressive cleanup throughout pipeline
   - Explicit garbage collection
   - Prevents memory leaks in long-running processes
4. **Responsive Design**
   - Device-specific rendering
   - Mobile/tablet/desktop optimizations
   - Touch gesture support
5. **Batch Processing**
   - Sequential processing with progress tracking
   - Individual error handling per image
   - Parallel-ready architecture
6. **Explainability**
   - Detailed reasoning for each match
   - Confidence metrics and recommendations
   - Transparent AI decision-making

5 Technology Stack

Component	Technology
Frontend	Vanilla JavaScript, HTML5, CSS3, Panzoom.js
Backend	FastAPI (Python), Uvicorn ASGI server
AI/ML	Google Cloud Vision API, RapidFuzz
Image Processing	OpenCV (cv2), PIL/Pillow
Caching	In-memory LRU cache (OrderedDict)
Data Transport	Base64-encoded JPEG in JSON

Table 1: Technology Stack Components

6 File Structure Reference

```
thrift_assist/
├── public/
│   └── web_app.html              # Frontend UI
├── backend/
│   ├── api/
│   │   ├── main.py               # FastAPI application
│   │   └── routes/
│   │       └── ocr.py            # /upload-and-detect endpoint
│   ├── services/
│   │   ├── image_service.py      # Image validation & I/O
│   │   ├── cache_service.py      # LRU caching
│   │   └── ocr_service.py        # OCR orchestration
│   ├── utils/
│   │   └── image_utils.py        # JPEG quality calculation
│   └── core/
│       └── config.py             # Configuration settings
├── vision/
│   ├── detector.py               # Vision API integration
│   └── matcher.py                # Phrase matching engine
└── utils/
    └── text_utils.py             # Text normalization
```