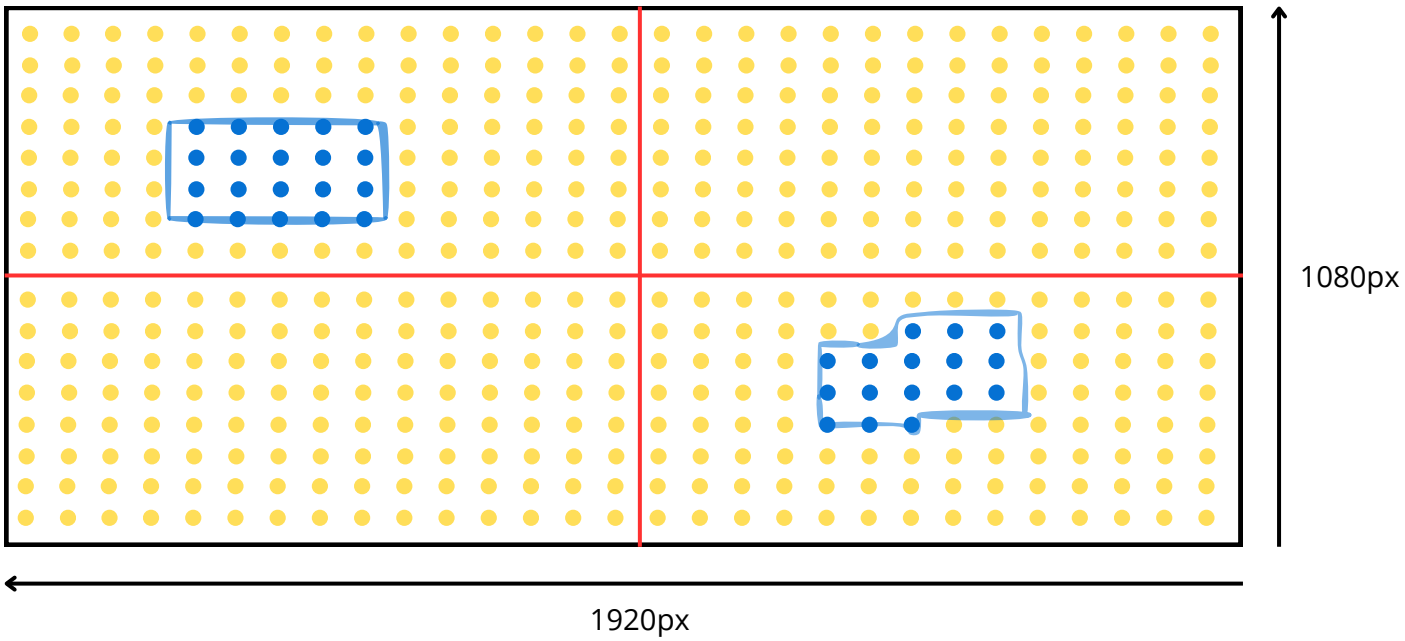
Training-Free Pipeline for Object Detection and Classification

Abdelrrahim Kaci Chaouche

abdelrrahim.com

Abstract

Traditional object detection systems often necessitate extensive training data collection, model training, and fine-tuning, posing significant barriers to rapid deployment and interpretability for non-expert users. This paper introduces a novel, lightweight, and highly configurable detection and classification system designed for dynamic object detection. Eschewing conventional deep learning model training, the system employs a unique multi-stage heuristic-driven region segmentation and grouping pipeline, augmented by a pre-trained multimodal embedding and a vector similarity database. This approach allows for identification and classification of images or UI elements (buttons, icons) and specific custom objects based on user-provided reference images, without requiring any model training or fine-tuning from the end-user. The system's architecture, methodology, mathematical formulations, implementation details, and capabilities are presented, demonstrating a pathway to intuitive and explainable visual content understanding.



Contents

1. Introduction

The ability for computer vision systems to accurately "see," analyze, and semantically interpret visual information is a cornerstone of modern artificial intelligence. This capability is fundamental across a vast array of applications, from autonomous navigation and industrial inspection to medical imaging analysis and content understanding. The core challenge in this domain lies in robustly identifying and classifying objects within complex and dynamic visual data.

While contemporary object detection, largely driven by state-of-the-art deep learning models, has achieved remarkable accuracy and efficiency, their practical implementation often presents significant barriers. These typically include: the demanding process of collecting and meticulously annotating vast datasets; the requirement for substantial computational resources and time for model training; and the specialized expertise needed for model development, optimization, and fine-tuning. Furthermore, the inherent "black box" nature of many deep neural networks can impede interpretability, making it challenging to understand their decision-making processes and hindering trust in their autonomous operation.

This paper introduces a novel approach to object detection and classification that addresses these fundamental complexities. Our system operates on a "train-free" paradigm, meaning it enables the recognition of custom objects without requiring the user to engage in any machine learning model training or fine-tuning. It achieves this by skillfully integrating a sophisticated pipeline of computer vision techniques for initial region segmentation and intelligent hierarchical grouping. This is combined with the power of pre-trained, multimodal embedding models and an efficient vector database for rapid similarity search. The core innovation lies in its multi-stage grouping strategy and selective semantic analysis, which together enable robust and efficient object recognition within diverse visual data. Users simply provide a small set of example images for custom objects, and the system autonomously processes these to facilitate their detection and classification.

1.1. Contributions

Two-Stage Hierarchical Grouping

- A novel segmentation and grouping pipeline that progresses from fine-grained pixel clusters to coarse-grained semantic regions, improving the logical coherence of detected bounding boxes.

Heuristic-Based Image or UI Classification (UI for agents)

- An explainable method for categorizing image/screen regions (e.g., for agents : ("icon-like", "button-like"), "general region" for images/UI) based on interpretable visual properties and defined mathematical thresholds. (for image only processing we rely only on regions)

Train-Free Semantic Object Recognition

- Integration of a pre-trained external multimodal embedding and a vector database to enable classification of user-defined custom objects purely through vector similarity to reference images, eliminating the need for local model training.

Selective Embedding Strategy

- An optimized approach that applies resource-intensive semantic embedding only to specific, more ambiguous region types, significantly enhancing system efficiency and reducing computational overhead.

Mathematical Foundations

- Explicit presentation of the mathematical principles underpinning color similarity, geometric property extraction, and region overlap analysis.

2. System Architecture and Methodology

The system operates as a sequential, multi-stage pipeline, prioritizing efficiency, configurability, and explainability. Figure 1 provides a high-level overview of its components and data flow.

I. Initial System Setup & Reference Data Ingestion
This phase represents the system's learning and initialization process, typically executed once or whenever new custom objects need to be introduced.

- 1. User Provides Reference Images:
  - **Description:** This is the starting point where the user inputs their example images of specific objects they want the system to recognize .

  - **Data Flow:** The system takes these uploaded images and prepares them for vector conversion.

- 2. Multimodal Embedding (Image to Vector Conversion):
  - **Description:** A dedicated embedding model that acts as an "AI visual interpreter." It receives image data from the user's reference images.

  - **Process:** This service leverages a powerful, pre-trained embedding model to analyze the visual content of each image. It then transforms this visual information into a unique sequence of numbers, known as a high-dimensional vector. This vector is essentially a "numerical fingerprint" or "digital DNA" that mathematically represents the image's semantic meaning.

  - **Data Flow:** It outputs a "High-Dimensional Vector" for each input image.

- **3. Vector Database (Store Reference Embeddings):**
  - **Description:** A specialized, high-performance database designed to store and efficiently search through vast collections of high-dimensional vectors. It acts as the system's long-term "visual memory." It receives the numerical fingerprints from the embedding model.

  - **Process:** The database stores each high-dimensional vector along with associated metadata, such as the object's label that is specified by the user. This process, called "upserting," makes these reference fingerprints available for future similarity searches.

- **4. System Ready for Detection:**
  - **Description:** The system is now fully initialized and prepared to begin its real-time object detection and classification tasks.

  - **Transition:** From this state, the system transitions to the Object Detection & Classification" phase.

II. Object Detection & Classification

Once the system is initialized and its visual memory (the vector database) is populated with reference object embeddings, it enters its operational phase, continuously analyzing dynamic visual content.

- **5. Real-time Visual Input Capture:**
  - **Process:** A high-speed screen capture mechanism is employed to acquire a pixel-perfect copy of the target area (image/computer screen/ camera). This digital image is then prepared for subsequent processing steps.

  - **Data Flow:** The output is a "Current Screen Image" (a pixel matrix).

- **6. Initial Region Segmentation (Pixel-Level Homogeneity Detection):**
  - **Description:** This initial analysis phase breaks down the complex "Current Screen Image" into fundamental, homogeneous visual units. It's akin to a painter identifying distinct color patches on a canvas.

  - **Process:**
    - **Sparse Pixel Sampling:** Instead of analyzing every single pixel (which would be computationally prohibitive for high-resolution images), the system intelligently samples pixels at a predefined sampling_spacing (e.g.,2px). This creates a grid of representative pixel points.
    - **Color-Based Region Growing:** A Breadth-First Search (BFS) algorithm is initiated from each sampled pixel that has not yet been processed. It then "grows" a region by iteratively including adjacent sampled pixels whose colors are deemed "similar" within a color_similarity_threshold. This process identifies connected components of visually uniform areas.
    - **Precise Bounding Box Extraction:** For each identified cluster of sampled pixels belonging to a homogeneous region, an accurate axis-aligned bounding box is computed. To achieve a very tight fit around the potentially irregular shape of the pixel cluster, the Convex Hull algorithm is applied to the coordinates of all pixels within that region. The bounding box is then derived from the minimum and maximum X and Y coordinates of the Convex Hull's vertices.
    - **Size Filtering (optional):** Only bounding boxes whose calculated area falls within a configurable min_region_size_pixels and max_region_size_pixels are retained. This filters out very small noise artifacts as well as overwhelmingly large, undifferentiated background areas.

  - **Data Flow:** The output is a collection of "Initial Region Bounding Boxes" (each with a unique ID and pixel coordinates).

7. Multi-Stage Hierarchical Grouping (From Fine to Coarse Structures):

- **Description:** This is a crucial, multi-step process designed to assemble the numerous "Initial Region Bounding Boxes" into more semantically meaningful, higher-level visual components, mimicking how humans perceive grouped elements on a screen.

- **Process:**
  - **Stage 1 Grouping (Horizontal Proximity):** The system first focuses on combining small regions that are very close horizontally. This is effective for merging elements like individual letters into words, parts of a segmented icon, or components of a complex button. Configurable parameters (GROUPING_STAGE1_MAX_X_GAP, GROUPING_STAGE1_MAX_Y_OVERLAP_ALLOWANCE, GROUPING_STAGE1_MAX_RECTS_PER_GROUP) dictate how aggressively this horizontal merging occurs. The result is a set of "Stage 1 Grouped Rectangles."

  - **Initial image/UI Classification (Heuristic-Based Guessing):** Each "Stage 1 Grouped Rectangle" is then immediately analyzed using a set of explicit, configurable heuristics. These heuristics evaluate the rectangle's geometric properties (width, height, area, aspect ratio) and its internal color variance. Based on these properties, the system makes an initial "guess" about its likely UI role, assigning labels such as "icon-like," "button-like," "wide-complex-region," "wide-region," or simply "region" if it doesn't fit a specific UI pattern. This step provides an early, explainable understanding of the region's nature.

  - **Stage 2 Grouping (Broader Proximity for General Content Blocks):** Only the "Stage 1 Grouped Rectangles" that were classified as generic "region" types (often representing larger, less distinct content blocks) are then passed to a second grouping stage (modified to the user's preferences). This stage uses broader proximity rules (GROUPING_STAGE2_MAX_X_GAP, GROUPING_STAGE2_MAX_Y_OVERLAP_ALLOWANCE, GROUPING_STAGE2_MAX_RECTS_PER_GROUP) to merge these areas into even larger, more encompassing visual blocks. This consolidates large image areas, paragraphs, or entire content sections into single "Stage 2 Grouped Rectangles."

  - **Final Consolidation:** The system then compiles a definitive list of "Final Processed Bounding Boxes" for subsequent steps. This list includes:
    - All "Stage 2 Grouped Rectangles" (assigned a new "stage2-grouped-region" label).

        - All "Stage 1 Grouped Rectangles" that were initially classified as a specific UI type (like "icon-like" or "button-like") and were *not* significantly overlapped by any "Stage 2 Grouped Rectangle." This careful overlap check prevents redundant or overlapping detections.

- **Data Flow:** The output is a comprehensive list of "Final Processed Bounding Boxes," each with its calculated pixel coordinates and a preliminary classification label (e.g., "icon-like," "button-like," "stage2-grouped-region," etc.).

8. Selective Semantic Analysis (The "Smart AI Eye" & Targeted Deep Understanding):

- Description: This is where the system intelligently decides which "Final Processed Bounding Boxes" require deep, semantic understanding using advanced AI, optimizing for both accuracy and computational efficiency.

- **Process:** For each "Final Processed Bounding Box":
    - The system first checks its preliminary classification label.
    - **Conditional Embedding:** If the label is "stage2-grouped-region," "wide-complex-region," or "wide-region," the system determines that this region is ambiguous enough (i.e., not clearly an icon or button) and potentially contains a custom object of interest (user datasets). For these specific types:
        - The corresponding sub-image (the pixel content within that bounding box) is precisely cropped from the "Current Screen Image."
        - This cropped image is then sent to a dedicated Multimodal Embedding .
        - The model analyzes the image's visual features and converts them into a unique "numerical fingerprint" – a high-dimensional vector. This vector semantically represents the image's content.

    - **Skipped Embedding (optional: based on preferences):** For all other preliminary classification labels (e.g., "icon-like," "button-like," "small-uniform-region"), the system intelligently skips this embedding process. It assumes these highly structured UI elements are unlikely to be the custom objects it's looking for, saving valuable processing time.

- **Data Flow:** For selected regions, the output is a "Region Embedding Vector." For optional skipped regions, no embedding is generated.

9. Vector Search and Final Classification:

- Description: For each "Region Embedding Vector" generated in the previous step, the system rapidly searches its "visual memory" (the vector database) to determine if it semantically matches any of the user-provided reference objects.

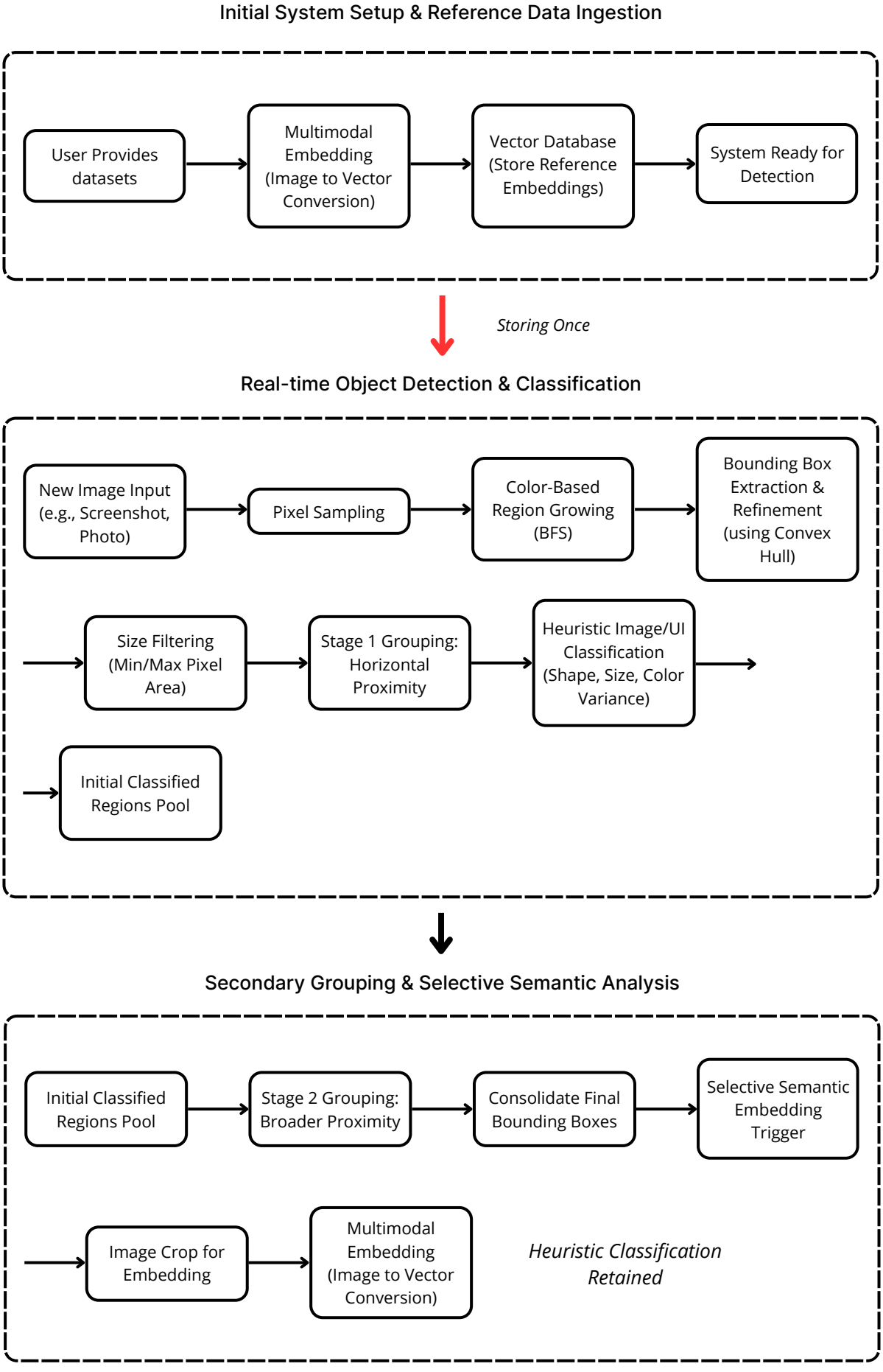
- Process:
  - The "Region Embedding Vector" is sent as a query to the Vector Database.
  - The database, optimized for high-dimensional vector similarity search, quickly compares the query vector against all stored reference embeddings using a mathematical distance metric (cosine similarity).
  - It returns the closest matching reference embedding, along with its associated metadata and a similarity score.
  - Thresholding and Label Assignment: The system then evaluates this similarity score. If the score is above a predefined OBJECT_DETECTION_THRESHOLD (indicating a strong match) AND the matched reference object's label, then the "Final Processed Bounding Box" is assigned this specific object label.
  - If the similarity score is below the threshold, or if the closest match is not one of the target custom objects, the region retains its preliminary classification label (e.g., "stage2-grouped-region," "wide-complex-region").

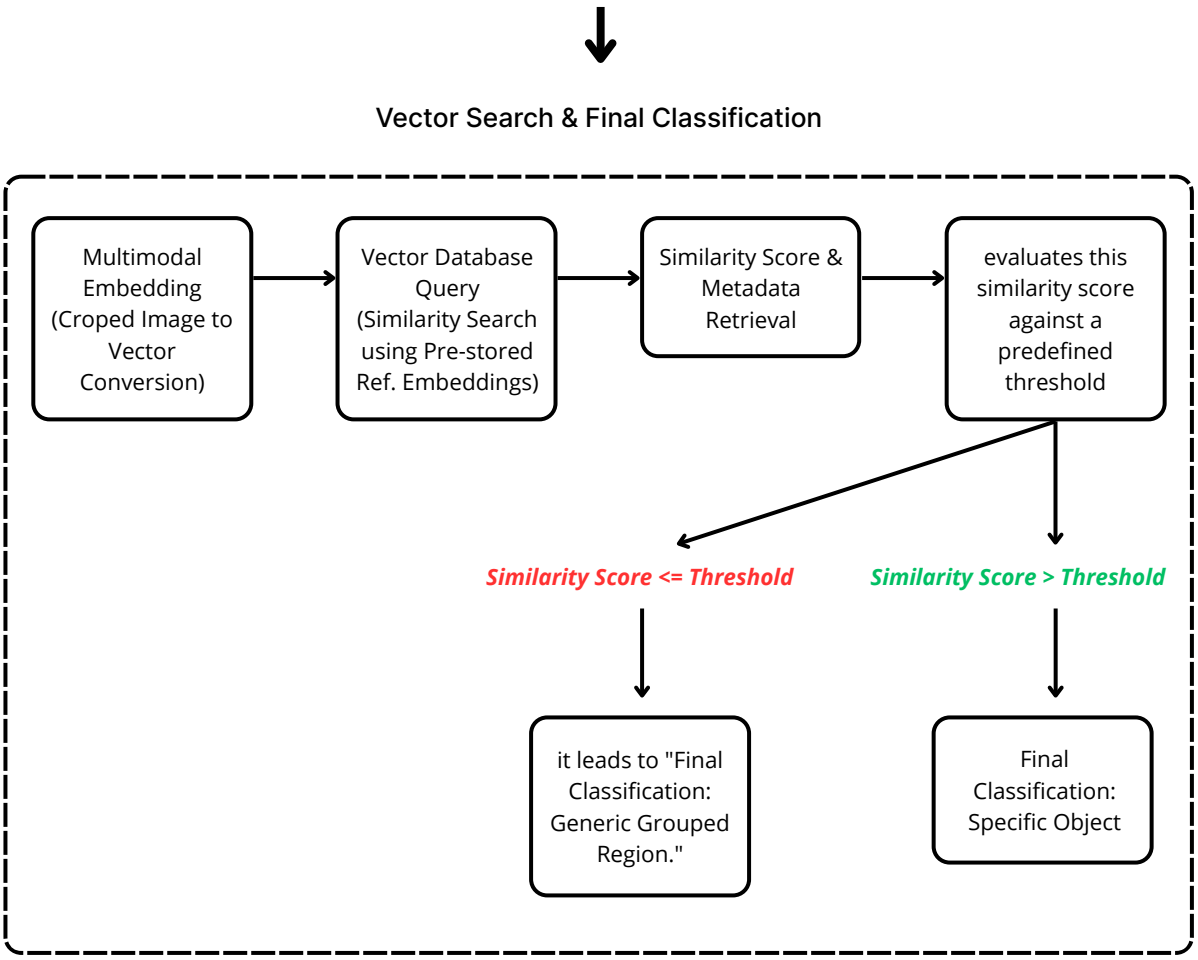
- **Data Flow:** The output for each "Final Processed Bounding Box" is a "Final Classification Label".

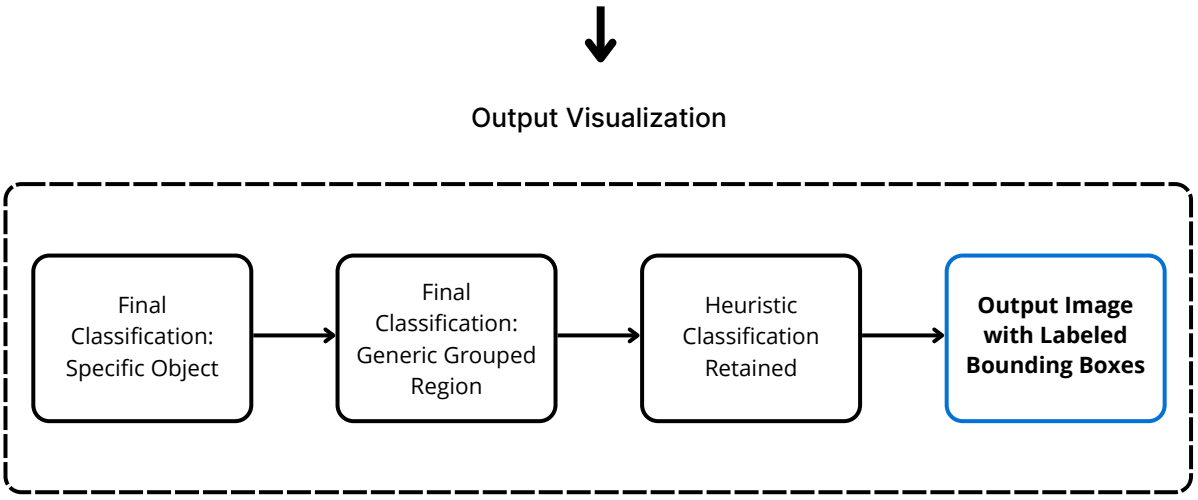
10. Visual Output Generation (Highlighting the Insights):

- **Description:** The final step involves visually representing the system's understanding by annotating the original image with the detected bounding boxes and their assigned classification labels.

- Process:
  - Using the list of "Final Processed Bounding Boxes" and their "Final Classification Labels," the system draws a distinct rectangular outline around each identified visual element.
  - A unique color is used for each classification type (e.g., specific colors for "icon-like," "button-like," "stage2-grouped-region," and dedicated colors for the detected object.
  - The corresponding "Final Classification Label" is then rendered as text adjacent to its bounding box, providing immediate, human-readable insights.


Initial System Setup & Reference Data Ingestion
User Provides datasets
Multimodal Embedding (Image to Vector Conversion)
Vector Database (Store Reference Embeddings)
System Ready for Detection
Storing Once
Real-time Object Detection & Classification
New Image Input (e.g., Screenshot, Photo)
Pixel Sampling
Color-Based Region Growing (BFS)
Bounding Box Extraction & Refinement (using Convex Hull)
Size Filtering (Min/Max Pixel Area)
Stage 1 Grouping: Horizontal Proximity
Heuristic Image/UI Classification (Shape, Size, Color Variance)
Initial Classified Regions Pool
Secondary Grouping & Selective Semantic Analysis
Initial Classified Regions Pool
Stage 2 Grouping: Broader Proximity
Consolidate Final Bounding Boxes
Selective Semantic Embedding Trigger
Image Crop for Embedding
Multimodal Embedding (Image to Vector Conversion)
Heuristic Classification Retained


Vector Search & Final Classification
Multimodal Embedding (Croped Image to Vector Conversion)
Vector Database Query (Similarity Search using Pre-stored Ref. Embeddings)
Similarity Score & Metadata Retrieval
evaluates this similarity score against a predefined threshold
Similarity Score <= Threshold
Similarity Score > Threshold
it leads to "Final Classification: Generic Grouped Region."
Final Classification: Specific Object


Output Visualization
Final Classification: Specific Object
Final Classification: Generic Grouped Region
Heuristic Classification Retained
Output Image with Labeled Bounding Boxes

3. In-Depth System Architecture and Methodology

3.1. Screen Acquisition and Initial Region Segmentation

The process begins by capturing the image (computer screen or uploaded image).
This raw pixel data, P(x,y)=(R,G,B) , serves as the input for analysis. To manage computational load on high-resolution images or displays, a sparse sampling strategy is employed. Instead of processing every pixel, the system inspects pixels at regular intervals defined by sampling_spacing.

- **Color-Based Region Growing:** A Breadth-First Search **(BFS)** algorithm is initiated from each unvisited sampled pixel **(x0,y0)**. It expands to neighboring sampled pixels **(x,y)** if their color **P(x,y)** is sufficiently similar to the seed pixel's color **P(x0,y0)**. Color similarity is determined by a component-wise Manhattan distance or a maximum absolute difference threshold:

$$\text{sim}(P_1, P_2) \text{ iff } \forall c \in \{R, G, B\}, |P_1(c) - P_2(c)| \leq \text{threshold}$$

This process segments the image or screen into distinct "color families" or homogeneous regions of sampled pixels.

Bounding Box Extraction: For each identified region, an initial axis-aligned bounding box
[xmin, ymin, xmax, ymax] is calculated. To refine this and ensure the bounding box tightly encapsulates the irregular shape formed by the sampled pixels, the Convex Hull algorithm is applied to all pixel coordinates corresponding to the sampled points within the region. The bounding box is then derived from the minimum and maximum coordinates of the vertices of this convex hull.

Size Filtering: Only regions whose area falls within a configurable **min_region_size_pixels** and **max_region_size_pixels** are retained, suppressing noise or overly large, undifferentiated background areas.

3.2. Multi-Stage Hierarchical Grouping

The system employs a two-stage grouping process to synthesize raw segmented regions into more coherent and semantically meaningfull components or content blocks.

3.2.1. Multi-Stage Hierarchical Grouping

Stage 1 Grouping (Horizontal Proximity):

- **Purpose:** To merge fine-grained, horizontally adjacent regions that often constitute a single logical visual unit (e.g., characters forming a word, or segments of a complex icon).

- **Mechanism:** An iterative merging process is applied to **initial_region_bboxes_with_ids**. Two rectangles,

  $R1=[x11,y11,x12,y12]$ and $R2=[x21,y21,x22,y22]$ are considered proximate if they satisfy:

  - **Horizontal Gap:**

    $$(\min(x11,x21)\leq\max(x12,x22)+max\_x\_gap) \text{ AND } (\max(x11,x21)\leq\min(x12,x22)+max\_x\_gap)$$

  - **Vertical Overlap:**

    $$(\min(y12,y22)>\max(y11,y21)-max\_y\_overlap\_allowance) \text{ AND } (\max(y12,y22)<\min(y11,y21)+max\_y\_overlap\_allowance$$

- Efficient spatial querying (e.g., using R-trees) accelerates this process, particularly for large numbers of initial regions. This results in **stage1_grouped_rects**.

3.2.2. Initial Image/UI Classification (Heuristic-Based):

Purpose: To categorize each stage1_grouped_rect into broad UI types based on observable geometric and color properties. This provides inherent explainability for the classification.

Mechanism: For each rectangle R=[x1,y1,x2,y2], its dimensions (W=x2−x1, H=y2−y1), area (A=W×H), and aspect ratio (AR=W/H) are computed. Additionally, the standard deviation of pixel color values (σc) within the rectangle is calculated for each RGB channel and averaged:

$$\sigma_{RGB} = \frac{1}{3} \sum_{c \in \{R,G,B\}} \sqrt{\frac{1}{N} \sum_{i=1}^{N} (P_i(c) - \mu_c)^2}$$

Classification rules are applied sequentially:

- **Icon-like:** If AR is within [ARmin_icon,ARmax_icon]and A is within [Amin_icon,Amax_icon], and σRGB is above a min_color_std_dev (indicating internal detail).

- **Button-like:** If AR is above ARmin_button and A is within [Amin_button,Amax_button], and σRGB is below a max_color_std_dev (indicating uniform background).

- **Wide-complex-region / Wide-region / Small-uniform-region:** Other heuristics capture nuanced shapes and color uniformities.

- **General Region:** If none of the above specific rules are met, the region is labeled as a generic "region".

3.2.3. Stage 2 Grouping (Broader Proximity of "General Regions"):

- **Purpose:** To consolidate the larger, less defined region types from Stage 1 into more encompassing content blocks (e.g., paragraphs, image areas), forming higher-level visual components.

- **Mechanism:** Only stage1_grouped_rects initially classified as region are subjected to this second grouping pass. The same proximity criteria (are_rects_horizontally_proximate) are used, but with potentially different, often less strict, GROUPING_STAGE2_MAX_X_GAP, GROUPING_STAGE2_MAX_Y_OVERLAP_ALLOWANCE, and GROUPING_STAGE2_MAX_RECTS_PER_GROUP values. This yields stage2_final_merged_rects.

- **Consolidation:** The final set of bounding boxes (_final_rects_to_draw) is constructed by:
  - Adding all stage2_final_merged_rects (labeled stage2-grouped-region).
  - Iterating through stage1_grouped_rects. If a stage1_grouped_rect was not initially classified as region (i.e., it was an icon, button, etc.), OR if it was a region but was not significantly covered by any stage2_final_merged_rect (to avoid duplicates and ensure hierarchy), it is added with its original heuristic label. Significant overlap is defined by an Intersection over Area (IoA) metric:

$$IoA(R_1, R_2) = \frac{Area(R_1 \cap R_2)}{Area(R_1)}$$

where R1∩R2 is the intersection of the two rectangles. If IoA(R1,R2)≥min_overlap_ratio (e.g., 0.8), R1is considered covered by R2.

3.3. Semantic Object Recognition via Embedding

This module provides the system's core capability for recognizing specific custom objects without requiring local model training.

- **Multimodal Embedding:** Instead of deploying and managing a local deep learning model, the system leverages a pre-trained Multimodal Embedding, it projects images into a high-dimensional vector space where semantic similarities are preserved.
- **Image Preparation:** For any selected region, the corresponding sub-image (cropped from the original image/screenshot) is converted into a Base64-encoded JPEG string, which is the standard format for transmission.
- **Vector Generation:** This Base64 image is sent to the embedding model. The returns a fixed-dimensional vector embedding (e.g., 1024-dimensions), which is a dense numerical representation capturing the visual semantics of the image.

3.4. Vector Search and Classification in a Vector Database

This module facilitates rapid and efficient similarity comparisons for object classification.

- **Vector Database:** A specialized vector database is employed for storing and searching high-dimensional embeddings. It is initialized with a specific vector dimension (e.g., 1024, matching the embedding model output) and uses cosine similarity as its distance metric.
- **Reference Image Embedding and Storage:** During initial setup, the system processes user-provided reference images. Each reference image is embedded using the same Multimodal Embedding. Its resultant vector is then stored in the vector database along with a unique identifier and a descriptive metadata label. This constitutes the system's "memory" for custom objects.

Querying and Classification: For a detected region requiring semantic classification, its embedding is generated via the embeding model. This embedding serves as a query vector to the vector database. The database performs a similarity search, returning the top-k nearest neighbor vectors from its stored collection, ranked by cosine similarity. Cosine similarity between two non-zero vectors A and B is defined as:

$$\text{similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| \cdot ||\mathbf{B}||}$$

The system then examines the top match. If the match's similarity score exceeds a predefined OBJECT_DETECTION_THRESHOLD (e.g., 0.9) and its metadata label corresponds to a target object, the region is confidently classified with that specific label. Otherwise, it retains its generic classification.

- **Selective Embedding Strategy:** A critical optimization is implemented here: the computationally intensive image embedding and vector search operations are only performed for regions classified as stage2-grouped-region, wide-complex-region, or wide-region.(it can be modified based on the user’s preferences) These are the most ambiguous and likely candidates for containing the specific objects of interest. Icons, buttons, and other clearly defined UI elements are not subjected to this semantic analysis, significantly improving overall system efficiency.

3.5. Visual Output Generation

The final step involves overlaying the detected bounding boxes. Different colors are assigned to distinct classification types (e.g., yellow for "icon-like", cyan for "button-like", purple for "stage2-grouped-region"), providing immediate and intuitive visual feedback. An optional coordinate plane can also be drawn for spatial reference.

4. Implementation Details and Parameters

The system is implemented in Python, leveraging libraries for image capture (mss), image processing (Pillow, NumPy, SciPy). A key aspect of its user-friendliness is the centralized configuration of parameters at the top of the script.

- **Image Acquisition:** Captures the image, *monitor_number* (for agents) specifies the display to capture.
- **Initial Region Segmentation:**
  - sampling_spacing: Typically 2-5 pixels, controlling pixel density analysis.
  - color_similarity_threshold: Range 0-255, defining RGB color difference tolerance for region growth.
  - min_region_size_pixels, max_region_size_pixels: Pixel area bounds for initial regions.
- **Grouping Parameters:** These are explicitly set for each stage:
  - Stage 1: GROUPING_STAGE1_MAX_X_GAP, GROUPING_STAGE1_MAX_Y_OVERLAP_ALLOWANCE, GROUPING_STAGE1_MAX_RECTS_PER_GROUP.
  - Stage 2: GROUPING_STAGE2_MAX_X_GAP, GROUPING_STAGE2_MAX_Y_OVERLAP_ALLOWANCE, GROUPING_STAGE2_MAX_RECTS_PER_GROUP.
- **UI/Image Classification Rules (classify_rules):** Configurable parameters for geometric properties (aspect ratios, areas) and color variance thresholds.
- **Semantic Classification:**
  - OBJECT_DETECTION_THRESHOLD: Tunable cosine similarity score (e.g., 0.9).
  - EMBEDDING_DIM: Set to 1024, depends on the chosen embedding.
- **Output Visualization:** grouped_outline_color_map allows customization of bounding box colors per classification, and label_font_size controls text size.