Visual Attention Analyzer

AI-powered Chrome extension for advanced webpage attention analysis

Presented By
Chandra Prakash S

01

Problem Statement

- Web designers and marketers struggle to predict where users will focus
- Traditional eye-tracking studies are expensive and time-consuming
- Need for accessible tools to understand visual attention patterns
- Lack of solutions combining visual and structural webpage analysis

Solution Overview

01. Chrome extension that analyzes visual attention patterns on webpages

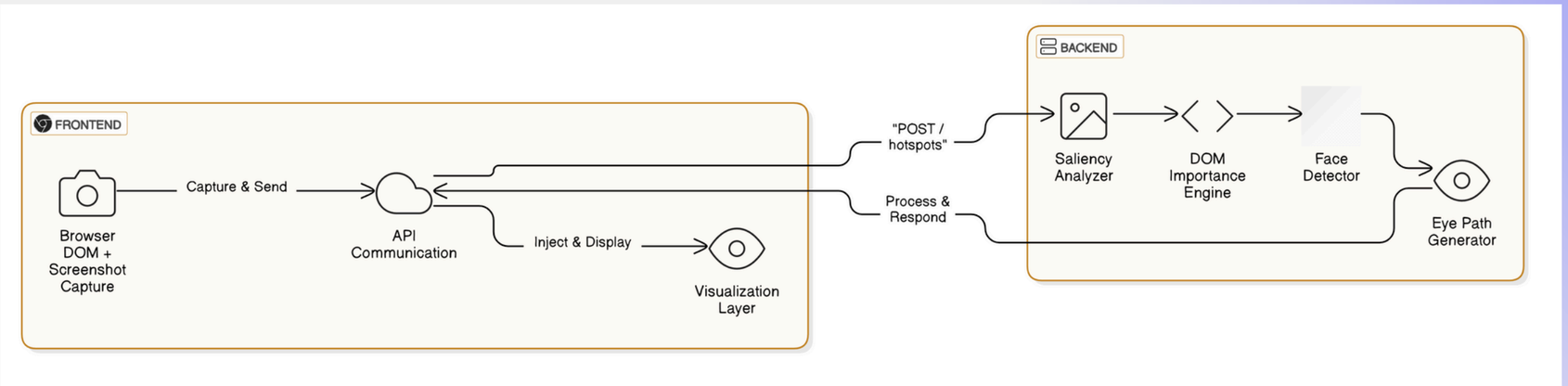
02. Combines computer vision with DOM analysis for comprehensive results
Provides interactive visualizations overlaid on the actual webpage

03. Makes advanced attention analytics accessible to everyone

02

Technical Architecture

- Frontend: Chrome Extension APIs (content scripts, background workers)
- Screenshot Capture: Chrome tabs API for full page capture
- DOM Analysis: JavaScript parsing of page structure and attributes
- Backend Processing: FastAPI Python server with computer vision algorithms
- Data Flow: Screenshot + DOM data → API → Visualization overlays

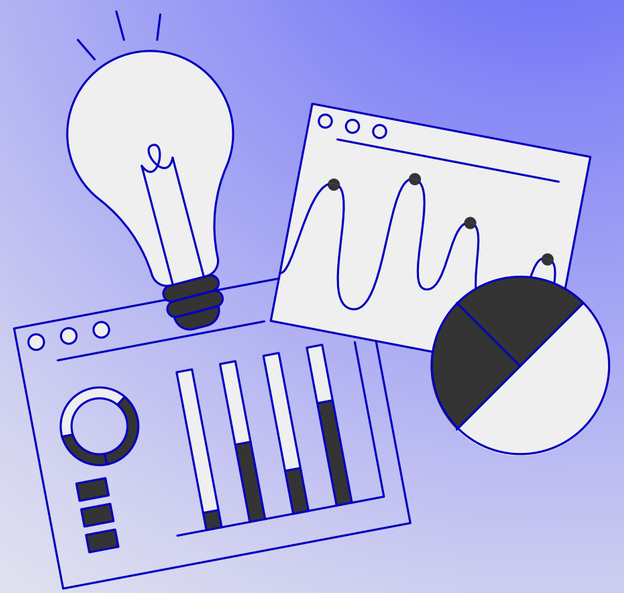


03

04 Core Algorithm

- **Spectral Residual Saliency Detection:**
  Identifies visually distinctive regions
- **Center Bias Application:**
  Accounts for natural human tendency to focus on center
- **MediaPipe Face Detection:**
  Prioritizes human faces in attention analysis
- **Hybrid Scoring System:**
  Combines visual saliency with DOM importance
- **Non-Maximum Suppression:**
  Merges overlapping regions for cleaner results

Visualization Features



- **Hotspot Regions:** Color-coded boxes with rank indicators
- **Eye Movement Path:** Numbered points with directional lines
- **Face Detection:** Purple outlines identify faces as attention anchors
- **Attention Heatmap:** Color-gradient overlay of attention distribution
- **Interactive Controls:** Toggle visibility of different visualization layers

05

Code Deep Dive: Backend

- **FastAPI Framework:** Efficient, async **Python** web server
- **Key Components:**
  - Saliency detection with **OpenCV**
  - Center bias application for natural viewing patterns
  - **DOM** importance calculation (tag weights, position, content)
  - Face detection integration with **MediaPipe**
  - **Hotspot** scoring algorithm that balances visual and semantic signals

Code Deep Dive: Frontend

- **Extension Components:**
  - Content script for DOM analysis and visualization rendering
  - Background script for screenshot capture
  - Interactive control panel for visualization toggles
- **Enhanced DOM Data Extraction:**
  - Element positions, attributes, and styles
  - Interactive element detection
  - Visibility checking and DOM depth calculation

07

Real-World Applications

Exploring creativity

01 UX designers optimizing webpage layouts

02 Marketing teams improving ad and CTA placement

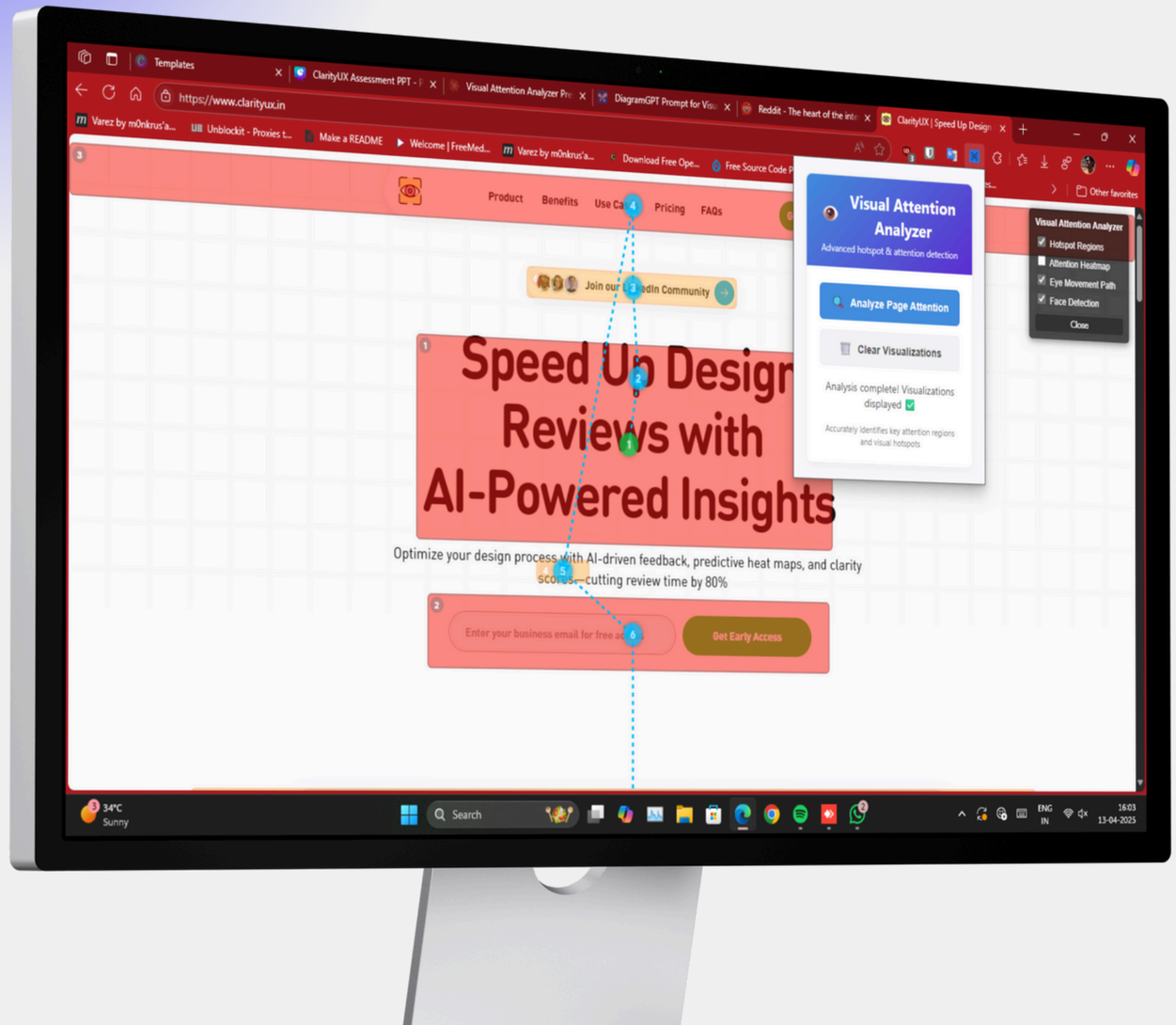
03 Content creators enhancing visual hierarchy

04 Accessibility specialists identifying focus issues

05 E-commerce sites optimizing product displays

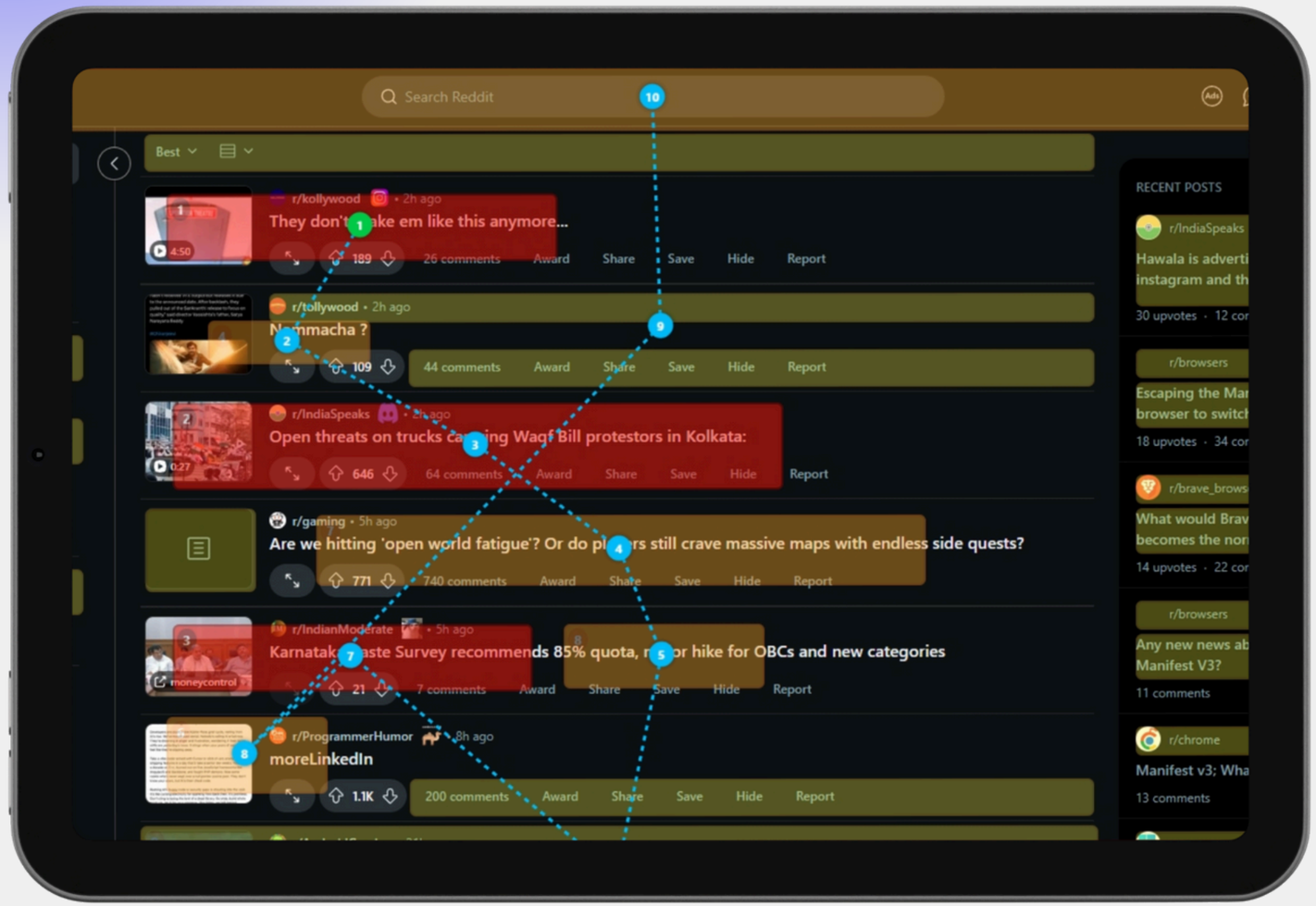
06 News sites arranging content for maximum engagement

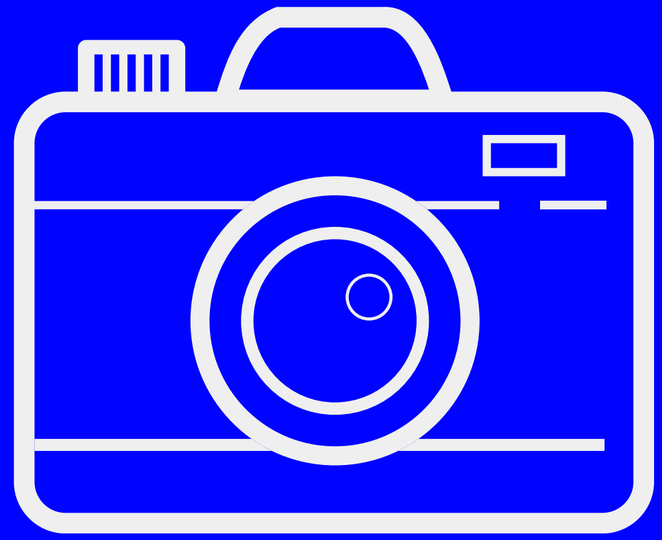
Screenshots



09

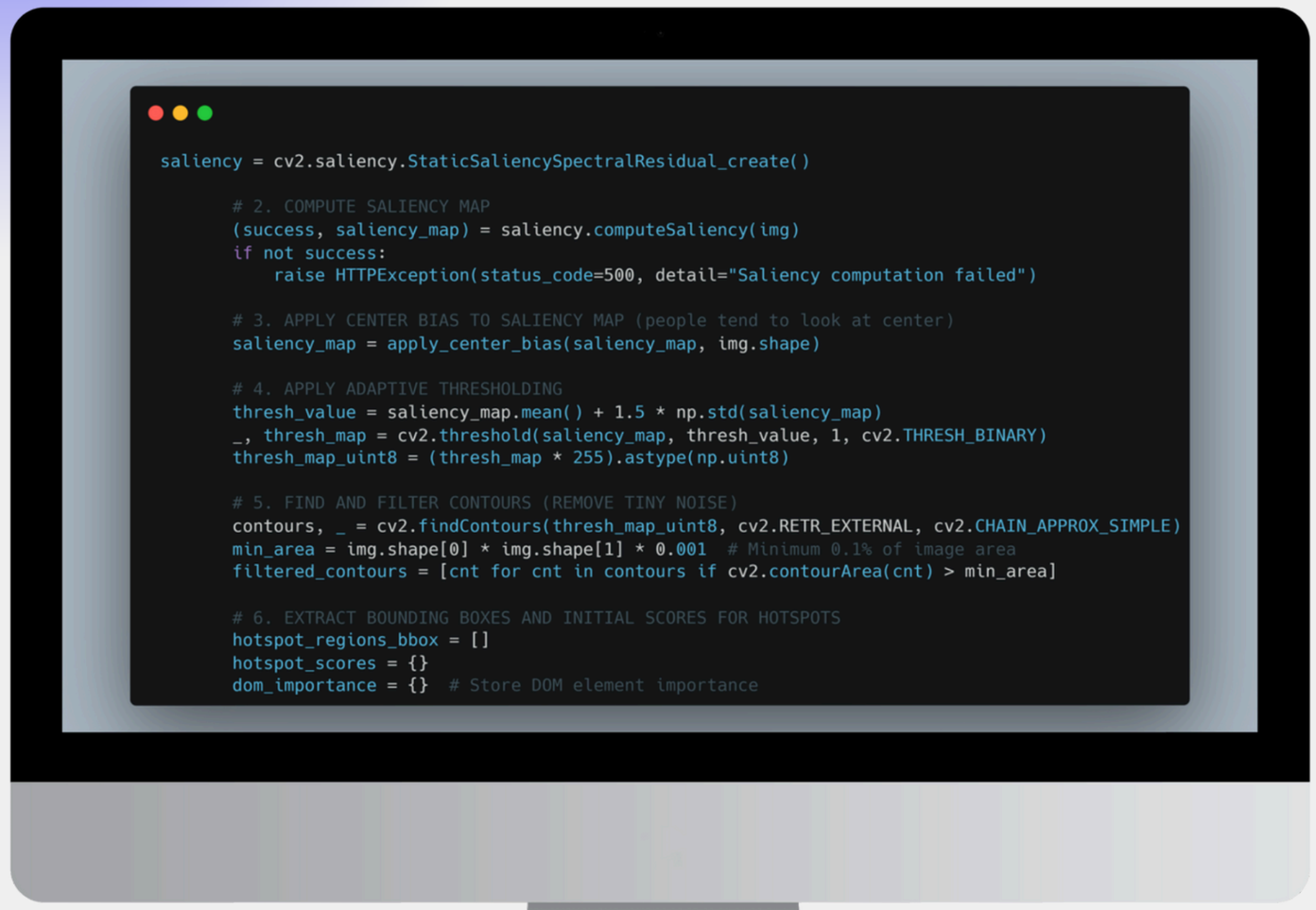
Screenshots





10

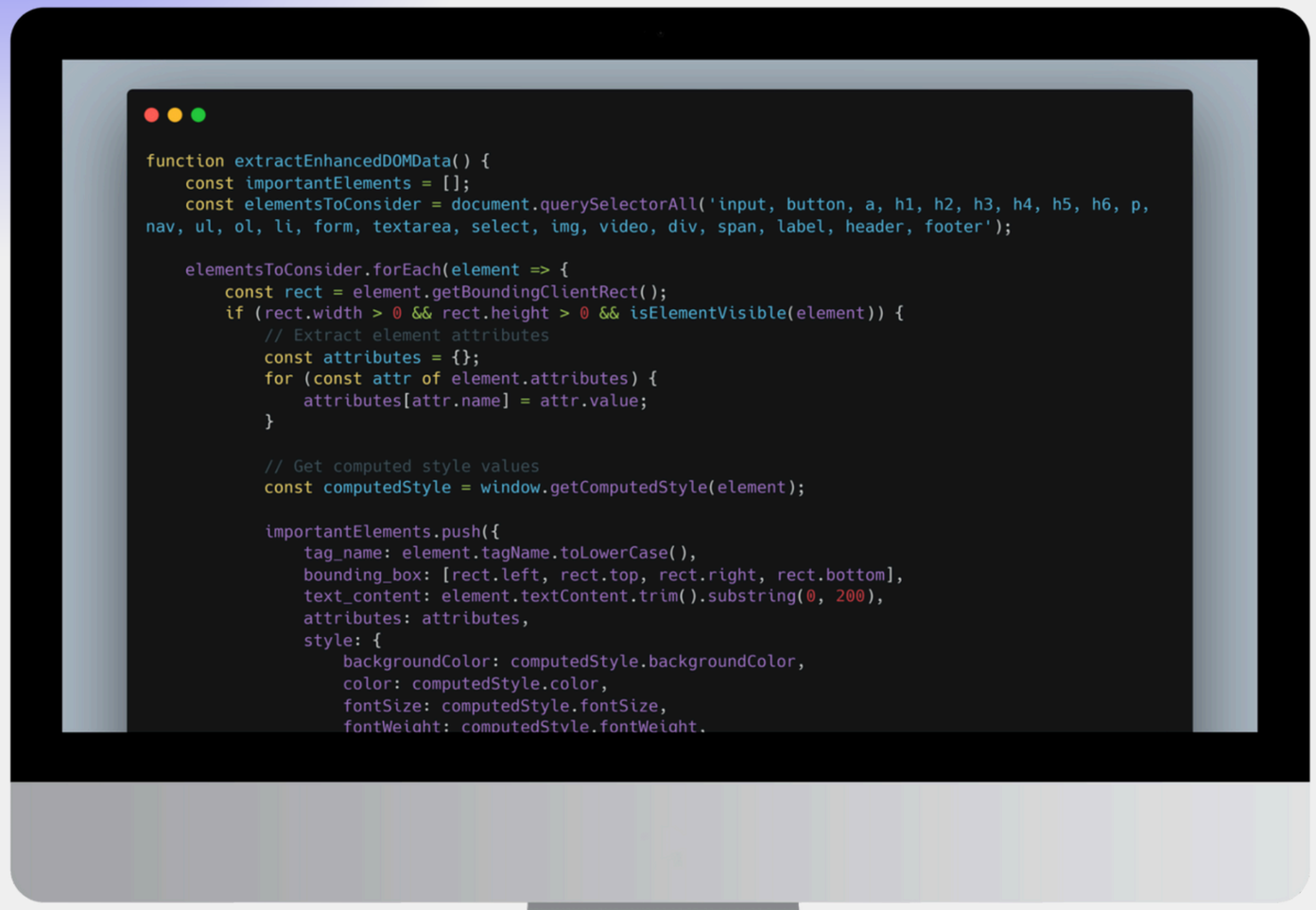
Code Snippet



```
saliency = cv2.saliency.StaticSaliencySpectralResidual_create()

        # 2. COMPUTE SALIENCY MAP
        (success, saliency_map) = saliency.computeSaliency(img)
        if not success:
            raise HTTPException(status_code=500, detail="Saliency computation failed")

        # 3. APPLY CENTER BIAS TO SALIENCY MAP (people tend to look at center)
        saliency_map = apply_center_bias(saliency_map, img.shape)

        # 4. APPLY ADAPTIVE THRESHOLDING
        thresh_value = saliency_map.mean() + 1.5 * np.std(saliency_map)
        _, thresh_map = cv2.threshold(saliency_map, thresh_value, 1, cv2.THRESH_BINARY)
        thresh_map_uint8 = (thresh_map * 255).astype(np.uint8)

        # 5. FIND AND FILTER CONTOURS (REMOVE TINY NOISE)
        contours, _ = cv2.findContours(thresh_map_uint8, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        min_area = img.shape[0] * img.shape[1] * 0.001  # Minimum 0.1% of image area
        filtered_contours = [cnt for cnt in contours if cv2.contourArea(cnt) > min_area]

        # 6. EXTRACT BOUNDING BOXES AND INITIAL SCORES FOR HOTSPOTS
        hotspot_regions_bbox = []
        hotspot_scores = {}
        dom_importance = {}  # Store DOM element importance
```

11

Code Snippet



```
function extractEnhancedDOMData() {
    const importantElements = [];
    const elementsToConsider = document.querySelectorAll('input, button, a, h1, h2, h3, h4, h5, h6, p,
nav, ul, ol, li, form, textarea, select, img, video, div, span, label, header, footer');

    elementsToConsider.forEach(element => {
        const rect = element.getBoundingClientRect();
        if (rect.width > 0 && rect.height > 0 && isElementVisible(element)) {
            // Extract element attributes
            const attributes = {};
            for (const attr of element.attributes) {
                attributes[attr.name] = attr.value;
            }

            // Get computed style values
            const computedStyle = window.getComputedStyle(element);

            importantElements.push({
                tag_name: element.tagName.toLowerCase(),
                bounding_box: [rect.left, rect.top, rect.right, rect.bottom],
                text_content: element.textContent.trim().substring(0, 200),
                attributes: attributes,
                style: {
                    backgroundColor: computedStyle.backgroundColor,
                    color: computedStyle.color,
                    fontSize: computedStyle.fontSize,
                    fontWeight: computedStyle.fontWeight,
```

12

Future Development

Exploring creativity

01 Machine learning model to incorporate user behavior data

02 Comparative analysis between multiple webpage versions

03 Integration with design tools (Figma, Adobe XD)

04 Mobile website/app support

05 Accessibility focus mode for inclusive design

06 API for integration with analytics platforms

14

Conclusion

- Visual Attention Analyzer bridges the gap between design and user perception
- Combines cutting-edge computer vision with website structure analysis
- Provides actionable insights for design optimization
- Open-source approach encourages community improvement

Thank You

Chandra Prakash