WORD-LEVEL BLUR FOR REAL-TIME SCREEN ABUSE DETECTION: IMPLEMENTATION AND EVALUATION

Authors:

[Nikesh Aote], Department of Computer Science and Engineering, [Priyadarshini college of Engineering]

[Sohel Sayyad], Department of Computer Science and Engineering, [Priyadarshini college of Engineering]

[Soheb Pathan], Department of Computer Science and Engineering, [Priyadarshini college of Engineering]

[Sheron Shiekh], Department of Computer Science and Engineering, [Priyadarshini college of Engineering]

[Sumit Kalbande], Department of Computer Science and Engineering, [Priyadarshini college of Engineering]

[Akash Bisen], Department of Computer Science and Engineering, [Priyadarshini college of Engineering]

Abstract:

Traditional screen abuse detection systems employ region-based blurring that obscures entire screen sections, causing significant visual disruption and poor user experience. This paper presents a novel word-level blur approach that uses optical character recognition (OCR) combined with natural language processing (NLP) to detect and blur only specific offensive words rather than entire regions. Our system employs a two-stage detection pipeline combining fast wordlist matching with context-aware NLP classification, partitioning the screen into nine parallel processing zones to achieve real-time performance. Experimental evaluation demonstrates 100ms frame processing time with 95%+ accuracy on clear text, representing 78% reduction in visual disruption compared to region-based methods while maintaining equivalent detection rates. The system supports English and Hindi text processing with transliteration handling and operates effectively in both GUI and headless deployment modes. User studies with 50 participants show 91% recommendation rate and significantly improved satisfaction scores over traditional region-based approaches, validating the practical benefits of surgical content filtering over blunt region-based methods.

Keywords: Abuse detection, word-level blurring, OCR, multilingual NLP, parallel processing, content moderation, real-time systems, two-stage detection

1. Introduction

1.1 Problem Background and Motivation

With over three billion users actively engaging on real-time communication platforms worldwide, screen-based content moderation has become an increasingly critical challenge for both individual users and enterprise organizations. Recent studies indicate that approximately 15% of active users encounter inappropriate or abusive content on a daily basis, creating an urgent need for effective automated protection systems that can operate seamlessly in real-time environments.

Existing commercial and academic solutions predominantly employ region-based blurring methodologies, where detection of potentially offensive content triggers the obscuring of entire predefined screen sections—often covering areas of 640×360 pixels or larger. While this approach provides a technical solution to content filtering, it creates substantial visual disruption that obscures legitimate content alongside any potentially offensive material. Users frequently find themselves unable to read surrounding context, follow conversations, or access important interface elements that happen to fall within these coarse blurring zones.

The fundamental disconnect lies in the treatment of text as regions rather than discrete semantic units. When a single offensive word appears in a chat window, current systems respond by blurring the entire conversation area, effectively destroying the user's ability to maintain context and continue productive interaction. This creates a frustrating paradox where the protective system itself becomes a significant impediment to normal usage.

1.2 Critical Limitations of Current Approaches

Through extensive analysis of existing screen abuse detection systems, we have identified three fundamental limitations that severely impact their practical utility:

Over-blurring and Context Destruction: Region-based approaches fundamentally misunderstand the granularity of textual content. Single offensive words trigger blurring of entire conversation windows or application sections, destroying not just the problematic content but all surrounding context and usability. Users face an unacceptable choice between viewing potentially harmful content or missing important information that happens to be spatially proximate to detected material.

High False Positive Rates Eroding Trust: Current generation systems exhibit false positive rates approximately averaging 12%, meaning that more than one in ten blurring events incorrectly hide content that was actually benign and appropriate. This level of inaccuracy systematically erodes user trust, teaching users that the system cannot be reliably depended upon. Consequently, many users eventually disable the protection entirely, defeating the fundamental purpose of having such a system installed.

Performance Bottlenecks on Modern Displays: Sequential processing approaches—where the system checks one screen region at a time in a linear fashion—create noticeable latency issues, particularly when operating on the high-resolution displays that have become standard in modern computing environments. A 4K monitor contains over eight million pixels, and checking them through single-threaded processing creates lag that users immediately perceive and find unacceptable. This performance limitation makes real-time protection increasingly impractical as display resolutions continue to advance.

1.3 Our Research Contributions

In response to these identified limitations, we have developed and implemented a word-level blur system incorporating several significant technical innovations:

Precision Targeting Through OCR Integration: Our approach utilizes advanced optical character recognition not merely to extract text content, but to determine the precise spatial coordinates of each individual word appearing on screen. This capability enables surgical blur application targeting only specific offensive words with appropriately sized padding buffers of 10-15% for visual comfort. The result preserves user ability to read all surrounding content normally while effectively obscuring only problematic material.

Parallel Processing Architecture for Scalability: Rather than checking the screen through slow sequential passes, we have implemented an architecture that divides the display into nine independent processing regions operating simultaneously through multi-threading. This design effectively leverages modern multi-core processors and maintains system responsiveness even when operating on high-resolution 4K displays where traditional approaches fail.

Comprehensive Multilingual Support: Real-world abusive content does not respect language boundaries. Users frequently encounter offensive material expressed through code-switching between languages, transliteration, or mixed linguistic contexts. We have built native support for both English and Hindi from the ground up, including sophisticated handling of the common practice of writing Hindi words using English alphabet characters (transliteration).

Flexible Deployment Options: Not every deployment scenario requires or benefits from a visible user interface. We have implemented a comprehensive background mode enabling the system to operate without any GUI components, making it suitable for server deployments, enterprise environments, and situations where users should remain unaware that protection mechanisms are active.

Intelligent Persistence Management: A blur effect that rapidly flickers on and off between frames creates visual distraction nearly as problematic as over-blurring itself. Our persistence system maintains active blur effects for 30-second durations by default, with automatic extension capabilities when offensive content remains visible, eliminating distracting flicker while ensuring appropriate protection duration.

1.4 Paper Organization and Structure

The remainder of this paper is organized as follows: Section 2 provides a comprehensive review of related work in content filtering, OCR applications, and NLP advances. Section 3 details our system architecture and the design principles that guided development decisions. Section 4 presents the core word-level blur algorithm with technical implementation details. Section 5 covers practical implementation specifics including optimization strategies and deployment configurations. Section 6 presents extensive experimental results across accuracy, performance, and user experience dimensions. Section 7 discusses system limitations, ethical considerations, and directions for future research. Finally, Section 8 concludes with a summary of contributions and broader implications for the field.

2. Related Work and Background

2.1 Evolution of Content Filtering Systems

Content moderation as a research field has evolved significantly over the past three decades, though the fundamental challenges have remained surprisingly persistent. Early approaches developed during the 1990s and early 2000s focused primarily on static content environments—websites, documents, and stored media files—where processing latency was less critical and content could be analyzed in its entirety before presentation to users.

Image-Based Detection Approaches: The past decade has witnessed substantial research investment in computer vision techniques for content moderation. Systems employing convolutional neural networks trained on extensive datasets of inappropriate imagery have demonstrated impressive accuracy rates for visual content classification. These systems can effectively identify nudity, violence, and various other visual violations with high precision. However, they face a fundamental architectural limitation: they process pixel patterns rather than semantic meaning. Word-based insults, profanity, and textual abuse pass entirely through image detection systems unless specifically trained to recognize particular text patterns—a training approach that cannot scale to the infinite variety of offensive expressions possible in natural language.

Text Analysis Without Spatial Context: On the language processing side, researchers have developed increasingly sophisticated natural language processing techniques moving far beyond early keyword matching approaches. Modern systems utilize machine learning classifiers including support vector machines, recurrent neural networks, and transformer-based architectures. While these systems demonstrate unprecedented understanding of textual content and context, they typically treat text as abstract sequential data without maintaining awareness of where that text physically appears on screen. This spatial blindness fundamentally prevents them from providing the precise targeting capabilities that real-time screen protection applications require.

2.2 Optical Character Recognition: From Documents to Screens

Optical character recognition technology has undergone remarkable advancement from early days of simple template matching and font-specific recognition. Modern OCR systems including Tesseract 5.0 and various commercial alternatives now employ deep learning architectures capable of reading text under almost any conceivable condition.

Success Stories in Specialized Domains: OCR technology has found tremendous practical success in specific application domains. License plate recognition systems can accurately read moving vehicle plates at highway speeds, enabling automated toll collection, traffic enforcement, and law enforcement applications. Document processing systems extract text from scanned paperwork, receipts, and forms with near-human accuracy levels. Mobile applications now allow users to point smartphone cameras at text in foreign languages and receive instant translations overlaid on the live camera feed.

The Screen Environment Gap: Despite these impressive achievements, the research community has largely overlooked application of OCR to dynamic, real-time screen content. Screen environments present unique technical challenges that break assumptions underlying document OCR: fonts change constantly between applications and interface elements, backgrounds are unpredictable and frequently animated, content moves and animates in ways that confuse frame-to-frame tracking, and text appears at arbitrary sizes, orientations, and color combinations. Static document processing assumptions break down completely when dealing with the chaotic visual environment of live screen capture. Our work specifically bridges this gap by adapting OCR techniques for the unique challenges of real-time screen content.

2.3 Natural Language Processing Advances for Abuse Detection

Natural language processing has undergone revolutionary transformation over the past five years, driven primarily by transformer architectures and massive pre-trained language models that capture subtle patterns in human communication.

The Transformer Revolution: Models such as BERT and its numerous variants have fundamentally expanded the boundaries of what is possible in computational text understanding. These models capture context, nuance, implication, and subtle meaning in ways that previous approaches could not approach. For abuse detection applications, this technological shift enables moving beyond simplistic keyword matching toward genuine understanding of when language becomes harmful versus merely containing words that might appear on a blocklist.

Zero-Shot Learning Capabilities: Zero-shot classification represents another major methodological advance with significant implications for abuse detection. These techniques enable models to classify text into categories they were never explicitly trained to recognize, leveraging the general language understanding built into large pre-trained models. For multilingual abuse detection scenarios, this capability is transformative—it means systems can

potentially detect harmful content in languages and linguistic varieties that developers never specifically included in training data.

The Spatial Integration Challenge: While NLP capabilities have advanced dramatically, applying these techniques to screen-based abuse detection requires integration with spatial awareness systems. Knowing that text is offensive represents only half the solution; one must also know precisely where that text appears on screen to apply appropriately targeted protection. This integration of sophisticated language understanding with millimeter-level spatial precision is where our research contribution is positioned.

2.4 Parallel Processing in Computer Vision Applications

The computer vision research community has long recognized that purely sequential processing cannot maintain pace with modern application demands. Parallel processing approaches utilizing GPU acceleration and multi-threading are now standard baseline requirements for object detection, video processing, and real-time analytics applications.

Unique Challenges in Text Processing: However, applying parallel processing architectures to OCR-based abuse detection presents distinctive challenges that do not arise in conventional computer vision. Unlike object detection where each spatial region can be processed completely independently, text frequently spans arbitrary region boundaries. A word might begin in one processing zone and end in another, or a sentence might flow across multiple partition lines. Our architecture addresses these complications through careful ROI design that minimizes boundary issues while maximizing the efficiency gains possible through parallel execution.

3. System Architecture and Design

3.1 Core Design Principles

Before examining specific technical implementation details, it is important to understand the guiding principles that shaped our architectural decisions. We approached this design with the explicit goal of creating a system that would be not merely technically correct in isolation, but practically useful in real-world deployment scenarios.

Minimal Visual Disruption: The primary objective guiding all design decisions was minimizing the visual footprint of protection measures. Users should be able to continue normal work and communication activities with minimal awareness that content filtering is occurring. This principle directly informed our word-level targeting approach, persistence management strategies, and blur rendering techniques.

Real-Time Performance Requirements: Protection that arrives after the fact is protection that has failed. Every component in our architecture was designed with strict latency requirements—target processing times under 150ms per frame to maintain user experience quality. This constraint drove decisions about parallel processing, model selection, and optimization strategies.

Multilingual Capability as Baseline: Rather than treating non-English language support as an afterthought or future extension, we built comprehensive multilingual handling into the fundamental architecture. This decision ensured that data structures, processing pipelines, and detection algorithms could naturally accommodate multiple scripts and linguistic varieties.

Deployment Flexibility: We recognized that different use cases would require different deployment configurations—some users would want a visible interface with controls and status information, while enterprise deployments would require silent background operation. Our architecture accommodates both modes through shared core components with interchangeable interface layers.

User Transparency and Control: Finally, we maintained a commitment to user transparency and control throughout the design. Users should understand what the system is doing, have access to detection logs, and be able to adjust sensitivity and behavior parameters according to their specific needs and preferences.

3.2 System Component Overview

The implemented system comprises six primary functional modules working in coordinated sequence:

Screen Capture Module: Handles acquisition of frame data from the display, supporting both physical display capture through pyautogui/mss and virtual display capture for headless deployments. Includes frame rate control and resolution normalization.

ROI Partitioning Engine: Divides the captured frame into nine processing regions (3×3 grid) for parallel processing. Each region represents 1/9 of total screen area and is assigned to dedicated worker threads through ThreadPoolExecutor.

OCR Processing Engine: Applies Tesseract 5.0+ through pytesseract interface to extract text content, bounding box coordinates, and confidence scores from each ROI. Includes multi-language support (English and Hindi) and confidence threshold filtering.

Abuse Detection Engine: Implements two-stage detection pipeline—fast wordlist matching followed by context-aware NLP classification when needed. Returns detected abusive words with confidence scores and spatial coordinates.

Blur Application Module: Calculates precise blur region coordinates from detected word positions, applies padding buffers (10-15%), and renders Gaussian blur overlays through the interface layer (tkinter/Kivy).

Persistence Management System: Tracks active blur regions, manages 30-second duration timers with automatic extension, handles cleanup of expired regions, and prevents frame-to-frame flickering through state maintenance.

3.3 Parallel Processing Architecture

The screen partitioning approach represents a fundamental architectural decision balancing processing efficiency against complexity management. We selected a 3×3 grid (nine regions) based on several technical considerations:

Scalability with CPU Cores: Modern consumer processors typically feature 4-16 cores. A nine-region partition provides sufficient parallelism to utilize available cores effectively while avoiding the thread management overhead that would accompany significantly larger partition counts.

Region Size Optimization: Each region represents approximately 11% of total screen area. This size is large enough to capture meaningful text context while small enough to enable efficient OCR processing and minimize cross-boundary text complications.

ThreadPoolExecutor Implementation: We utilize Python's concurrent.futures.ThreadPoolExecutor for worker thread management, providing automatic thread pooling, load balancing, and result aggregation without requiring manual thread lifecycle management.

3.4 Complete Data Flow Architecture

The end-to-end data flow through the system follows a carefully orchestrated pipeline:

1. Screen Capture: Acquisition of raw frame data at configurable intervals (default 30 FPS capture, processed at effective 8-12 FPS)

2. ROI Partitioning: Division of frame into nine regions with coordinate tracking

3. Parallel OCR: Simultaneous text extraction from all regions with confidence scoring

4. Result Aggregation: Collection and coordinate transformation of OCR results to global screen space

5. Heuristic Filtering: Removal of UI elements and noise from OCR output

6. Abuse Detection: Two-stage analysis (wordlist matching, conditional NLP classification)

7. Coordinate Calculation: Conversion of word bounding boxes to blur regions with padding

8. Blur Rendering: Gaussian blur overlays at calculated coordinates

9. Persistence Management: State tracking, duration management, and automatic cleanup

3.5 Technology Stack and Dependencies

The system is implemented in Python 3.8+ and builds upon established libraries in the computer vision and NLP ecosystems:

Core Processing:

- OpenCV (cv2) for image processing and blur rendering

- NumPy for numerical operations and array manipulation

- Pillow (PIL) for image format handling and conversion

OCR and Text Processing:

- Tesseract 5.0+ OCR engine (system dependency)

- pytesseract for Python interface to Tesseract

Natural Language Processing:

- transformers library for pre-trained model access

- torch as underlying deep learning framework

- Toxic-BERT model for English toxicity classification

- mDeBERTa-v3-base-mnli-xnli for multilingual zero-shot classification

User Interface:

- Kivy for cross-platform UI framework

- KivyMD for Material Design components

- tkinter for lightweight overlay rendering

Screen Capture:

- pyautogui for desktop display capture

- mss as alternative/backup capture method

- Xvfb and ImageMagick for headless/virtual display environments

Utilities and Infrastructure:

- concurrent.futures for parallel processing

- SQLite3 for detection logging and data persistence

- JSON for configuration management

- systemd for service management (Linux deployments)

4. Word-Level Blur Algorithm

4.1 Core Algorithm Concept and Innovation

The central technical innovation of our approach is the shift from region-based to word-based content filtering. Rather than defining fixed screen regions and blurring them entirely when any offensive content is detected, our system identifies specific offensive words and applies blur effects only to the precise spatial areas occupied by those words.

The algorithm operates through the following conceptual sequence: First, optical character recognition extracts all visible text from the screen along with the bounding box coordinates (x, y, width, height) specifying exactly where each word appears. Second, the abuse detection pipeline identifies which extracted words contain offensive content. Third, for each detected offensive word, the system calculates an expanded blur region by applying a configurable padding percentage (default 10-15%) to ensure complete coverage while maintaining precision. Finally, Gaussian blur is rendered only within these calculated regions, leaving all other screen content fully visible and readable.

This approach fundamentally changes the user experience equation. Where region-based systems might blur 25-30% of the screen area for a single detected word, word-level targeting typically affects less than 2% of total screen area for the same detection scenario.

4.2 OCR Output Filtering Pipeline

Raw OCR output contains substantial noise that would create false positives if passed directly to abuse detection. Screen environments are filled with UI elements, button labels, status text, and other non-content text that must be filtered before detection.

Our filtering pipeline implements multiple heuristic layers:

Confidence Thresholding: Tesseract provides confidence scores (0-100) for each recognized word. We filter out results below 60% confidence, eliminating most recognition errors while preserving legitimate text.

Length Filtering: Very short text fragments (fewer than 3 characters) are typically UI elements or recognition artifacts rather than meaningful content. These are filtered to reduce noise.

UI Element Blacklist: Common UI words like "screen", "detector", "running", "button", "click", "menu", and numeric sequences are automatically excluded from processing.

python

```
def filter_ocr_output(ocr_data):
    """
Filter OCR output to remove UI elements and low-confidence results.
    Args:
    ocr_data: List of OCR word objects with text, confidence, bbox
        Returns:
       Filtered list of relevant word objects
    """
    filtered = []
    ui_words = ['screen', 'detector', 'running', 'button',
                'click', 'menu', 'start', 'stop', 'close']
    for word in ocr_data:
        Skip low confidence detections
       if word.confidence < 60:
          continue
        Skip very short text (likely UI elements)
       if len(word.text) < 3:
          continue
        Skip common UI words
       if word.text.lower() in ui_words:
          continue
        Skip purely numeric sequences
       if word.text.isdigit():
          continue
         filtered.append(word)
    return filtered
```

4.3 Two-Stage Detection Pipeline

Our detection architecture implements a cascaded approach balancing speed and accuracy through two complementary stages:

Stage 1: Fast Wordlist Matching

The first stage performs exact matching against a curated database of abusive words and their common variations. This stage is extremely fast (2-5ms per word) and catches the majority of explicit abusive content.

The wordlist includes:

- Direct abusive terms in English and Hindi

- Common misspellings and obfuscations (e.g., "sh1t" for "shit")

- Transliterated Hindi terms written in English letters

- Stemmed variations and conjugations

If Stage 1 detects any abusive words, the system proceeds directly to blur calculation without invoking the slower NLP models.

Stage 2: NLP Classification (Conditional)

When Stage 1 finds no matches but text is present, the system conditionally invokes NLP models for context-aware detection. This stage is more computationally expensive (50-100ms) but catches implicit toxicity, creative insults, and context-dependent abuse that wordlists cannot capture.

We utilize two specialized models:

- Toxic-BERT: Fine-tuned BERT model specifically for English toxicity detection across multiple severity levels

- Zero-Shot Classifier: mDeBERTa-v3-base-mnli-xnli for multilingual support without language-specific training

The confidence threshold for NLP flagging is set at 0.7, providing good balance between detection sensitivity and false positive prevention.

4.4 Coordinate Calculation and Padding

Upon detecting an abusive word at OCR coordinates (x, y, width, height), the system calculates the final blur region through the following formula with padding percentage p (default 0.15):

blur_x = x - (width × p)

blur_y = y - (height × p)

blur_width = width + (2 × width × p)

blur_height = height + (2 × height × p)

This formula applies symmetric padding in all directions, ensuring the blur region completely covers the word while extending slightly beyond it for visual comfort. The 15% padding value was determined through user testing as providing the best balance between complete coverage and minimal footprint.

For boundary conditions where calculated regions extend beyond screen edges, the system clamps coordinates to valid screen bounds.

4.5 Blur Persistence and Anti-Flickering

A significant usability challenge in real-time content filtering is frame-to-frame flickering. If blur effects appear and disappear rapidly as frames are processed, the result is visually distracting and nearly as problematic as over-blurring.

Our persistence management system addresses this through several mechanisms:

Duration-Based Persistence: When a blur region is created, it is assigned a default duration of 30 seconds. The blur remains active for this entire period regardless of whether the offensive word is still detected in subsequent frames.

Automatic Extension: If the offensive word is detected again while its blur is still active, the duration timer resets, effectively extending the blur period. This ensures continuous protection as long as problematic content remains visible.

Background Cleanup: A dedicated background thread periodically scans active blur regions and removes expired entries, freeing resources and preventing memory growth.

python

```
class BlurManager:
    """
    Manages active blur regions with persistence and automatic cleanup.
    """
    def __init__(self, default_duration=30):
        self.active_blurs = {}
        self.default_duration = default_duration
        self.lock = threading.Lock()
     def add_or_extend_blur(self, region, word):
        """Add new blur or extend existing blur duration."""
        with self.lock:
            current_time = time.time()
            self.active_blurs[word] = {
                'region': region,
                'expires': current_time + self.default_duration
            }
       def get_active_regions(self):
        """Return list of currently active blur regions."""
        with self.lock:
            return [b['region'] for b in self.active_blurs.values()]
    def cleanup_expired(self):
        """Remove expired blur regions."""
        with self.lock:
            current_time = time.time()
            expired = [w for w, b in self.active_blurs.items()
                       if b['expires'] < current_time]
            for word in expired:
                del self.active_blurs[word]
```

5. Implementation Details

5.1 Core Processing Pipeline Implementation

The system implements a continuous processing loop that captures, analyzes, and responds to screen content in real-time. The pipeline architecture ensures efficient data flow while maintaining modularity for testing and extension.

The processing sequence operates as follows: (1) Frame acquisition from screen capture module at configured intervals; (2) Division into nine ROI partitions with coordinate tracking; (3) Parallel OCR execution across all regions

through ThreadPoolExecutor; (4) Result aggregation with coordinate transformation to global screen space; (5) Heuristic filtering to remove UI elements and noise; (6) Two-stage abuse detection (wordlist matching, conditional NLP classification); (7) Coordinate calculation with padding for detected words; (8) Blur rendering through UI overlay system; (9) Persistence management and state update.

Each stage is implemented as a distinct module with well-defined interfaces, enabling independent testing, debugging, and optimization. Error handling at each stage ensures that failures in one component do not cascade to crash the entire system.

5.2 Performance Optimization Strategies

Several optimization techniques were implemented to achieve real-time performance targets:

Adaptive Frame Skipping: Processing every captured frame would waste substantial CPU resources, particularly when screen content remains static. The system implements motion detection between consecutive frames, calculating pixel difference thresholds to identify significant changes. When minimal change is detected, the system skips OCR processing for that frame, reducing CPU utilization by approximately 40% during typical usage.

Intelligent OCR Caching: User interface elements (taskbars, menus, window chrome) typically remain static for extended periods. The system caches OCR results for detected UI regions with a 5-second time-to-live, eliminating redundant text recognition on unchanged content. This cache is indexed by region hash for O(1) lookup performance.

Singleton Model Loading: NLP models (Toxic-BERT, zero-shot classifier) require approximately 2GB RAM and 10-15 seconds to load from disk. Rather than loading models for each detection event, the system implements a singleton pattern that loads models once at application startup and shares the loaded instance across all detection operations. This eliminates initialization overhead entirely during normal operation.

Region-of-Interest Optimization: By processing only nine specific regions rather than the entire screen, the system reduces OCR workload by focusing on areas likely to contain text content. Dynamic ROI adjustment allows the system to shift focus based on detection history, concentrating processing resources where they are most needed.

5.3 Headless Operation for Server Deployment

Deployment in server environments such as AWS EC2 requires operation without physical display hardware. Standard screen capture libraries (pyautogui) fail in such environments because they depend on X11 display servers that do not exist on headless systems.

We implemented a dual-mode architecture that automatically detects the execution environment and selects appropriate capture mechanisms:

```
python

import os

 Environment detection

IS_HEADLESS = os.environ.get('ABUSE_DETECTOR_BACKGROUND', False)

if IS_HEADLESS:

    Use virtual display capture for AWS/server environments

   from aws_screen_capture import screenshot, screen_size

else:

    Use standard desktop capture

   import pyautogui

   screenshot = pyautogui.screenshot

   screen_size = pyautogui.size
```

The headless mode implementation uses Xvfb (X virtual framebuffer) to create a virtual display that exists only in memory. Screen capture is performed through ImageMagick's `import` command targeting this virtual display. This

approach enables full system functionality—including screenshot capture for detection logging—without requiring physical monitors or graphics hardware.

5.4 Configuration and Deployment Management

The system supports comprehensive JSON-based configuration enabling customization without code modification:

```
json
{
  "blur_settings": {
    "padding": 0.15,
    "persistence_duration": 30,
    "extension_enabled": true
  },
  "detection_settings": {
    "confidence_threshold": 0.7,
    "wordlist_path": "data/abusive_words.json",
    "nlp_enabled": true,
    "ocr_confidence": 60
  },
  "language_settings": {
    "supported_languages": ["eng", "hin"],
    "script_detection": true
  },
  "performance_settings": {
    "parallel_workers": 9,
    "frame_skip_threshold": 0.05,
    "ocr_cache_ttl": 5
  },
  "deployment_settings": {
    "headless_mode": false,
    "log_level": "INFO",
    "database_path": "data/detections.db"
  }
}
```

Configuration is loaded at startup and can be modified through the GUI or by editing the configuration file directly. Changes to most parameters take effect immediately without requiring restart.

5.5 Detection Logging and Analysis

The system maintains a SQLite database recording all detection events for post-hoc analysis and system tuning:

Logged Data Fields:

- Timestamp (ISO 8601 format)

- Detected word text

- Detection confidence score

- Detection method (wordlist or NLP)

- Screenshot file path (if captured)

- Screen coordinates of detection

- Language detected

This data enables several analysis capabilities: frequency analysis of detected terms, identification of false positive patterns, performance metrics calculation, and detection trend visualization over time. Users can export logs for external analysis or review through the built-in dashboard.

6. Experimental Results and Evaluation

6.1 Evaluation Methodology

We conducted comprehensive evaluation across three primary dimensions: detection accuracy, computational performance, and user experience quality. Testing employed a dataset of 500 test images representing diverse screen content scenarios, including 150 images containing known abusive content in various languages and presentation formats.

Hardware test configurations included: (1) Consumer desktop (Intel i7, 16GB RAM, GTX 1060), (2) Laptop (Intel i5, 8GB RAM, integrated graphics), (3) AWS EC2 t3.large instance (2 vCPU, 8GB RAM) for server deployment testing.

6.2 Detection Accuracy Analysis

Wordlist Detection Performance:

On clearly rendered text with standard fonts and adequate contrast, wordlist matching achieved 97.2% precision and 94.8% recall. Performance degrades gracefully under adverse conditions: 89% precision on low-contrast text, 85% precision on significantly rotated text (>15 degrees).

NLP Classification Results:

Toxic-BERT achieved 91.3% F1-score on English test data across multiple toxicity categories (insult, profanity, threat, identity attack). Zero-shot classification reached 84.7% accuracy on Hindi transliterated text, demonstrating effective cross-lingual capability without specific training.

Combined System Accuracy:

The integrated system (wordlist + NLP) achieved overall accuracy of 95.1% with 4.9% false positive rate. This represents a 59% reduction in false positives compared to baseline region-based systems' average 12% false positive rate.

6.3 Performance and Efficiency Metrics

Comparative performance analysis between region-based and word-level approaches:

| Performance Metric | Region-Based | Word-Level | Improvement |

||--||-|

| Average Processing Time | 245ms | 98ms | 60% faster |

| Visual Disruption Area | 28% of screen | 6% of screen | 78% reduction |

| False Positive Rate | 12% | 4.9% | 59% reduction |

| CPU Usage (average) | 45% | 31% | 31% reduction |

| Memory Footprint | 1.2GB | 2.1GB | 75% increase |

| User Satisfaction | 3.2/5 | 4.6/5 | 44% increase |

The increased memory footprint reflects NLP model loading; this is a fixed cost regardless of detection volume.

6.4 Multilingual Performance Evaluation

Language-specific detection performance:

| Language | Precision | Recall | F1-Score | Notes |

|-|--|--|-|-|

| English (standard) | 96.8% | 95.2% | 96.0% | Baseline performance |

| Hindi (Devanagari) | 93.4% | 89.1% | 91.2% | Native script support |

| Hindi (Transliterated) | 88.7% | 84.3% | 86.4% | Romanized text |

| Mixed (Eng+Hin) | 94.2% | 91.7% | 92.9% | Code-switching |

| Emoji-heavy text | 89.1% | 87.5% | 88.3% | Modern messaging |

Transliterated Hindi (written in English letters) presents the greatest challenge due to spelling variations and context-dependent interpretation. However, 86.4% F1-score remains acceptable for practical deployment.

6.5 Scalability Across Display Resolutions

Testing across common display resolutions demonstrated consistent performance:

| Resolution | Processing Time | CPU Usage | Memory |

||--|--|--|

| 1920×1080 (FHD) | 89ms | 23% | 2.1GB |

| 2560×1440 (QHD) | 94ms | 31% | 2.1GB |

| 3840×2160 (4K) | 108ms | 45% | 2.3GB |

| 5120×2880 (5K) | 127ms | 52% | 2.3GB |

Parallel processing architecture maintains sub-150ms latency even at 5K resolution, demonstrating effective scalability. Memory usage remains essentially constant as display resolution increases because the nine-region partitioning normalizes processing workload.

6.6 User Experience Study Results

Fifty participants (ages 18-45, diverse technical backgrounds) used the system for one week in their normal computing activities. Quantitative results:

- 78% found word-level blur "significantly less annoying" than previous experiences with region-based systems

- 82% reported maintaining conversation context better with word-level approach

- 91% indicated they would recommend the system to friends or colleagues

- 87% preferred word-level blur for long-term daily usage

- Average annoyance rating: 1.8/5 (compared to 3.9/5 for region-based systems)

Qualitative feedback highlighted appreciation for "knowing what's happening in the conversation" and "not missing important messages just because someone used bad language."

7. Discussion

7.1 System Strengths and Advantages

The word-level approach delivers fundamental improvements in user experience quality. By preserving surrounding context and targeting only specific offensive words, users maintain situational awareness and can continue productive activities without disruption. The parallel architecture ensures scalability across the full range of modern display

resolutions, from laptop screens to professional 4K and 5K monitors. Multilingual support addresses the linguistic reality of modern digital communication, where code-switching between languages is commonplace.

The flexibility to operate in both GUI and headless modes enables deployment across diverse scenarios—from individual user protection to enterprise server environments. The persistence management system successfully eliminates the flickering problem that has historically plagued real-time content filtering systems.

7.2 Acknowledged Limitations

OCR Dependency Constraints: System accuracy fundamentally depends on OCR quality. Poor text quality factors—low contrast between text and background, unusual or decorative fonts, extreme rotation angles, motion blur, or complex background patterns—reduce detection accuracy. Handwritten text is not supported and will not be detected.

Computational Resource Requirements: NLP models require approximately 2GB RAM and benefit significantly from GPU acceleration. Resource-constrained environments (older hardware, minimal cloud instances) may experience degraded performance or may need to disable NLP features entirely.

Maintenance Overhead: Language evolves continuously, with new terms, slang, and creative misspellings emerging regularly. The wordlist requires periodic updates to maintain detection effectiveness against evolving abusive language patterns.

Cross-Region Text Complications: Text spanning multiple ROI boundaries may be split during processing, potentially affecting detection of very long words or phrases. While our padding algorithm mitigates this issue, it cannot completely eliminate boundary effects.

7.3 Ethical Considerations and Responsible Deployment

Privacy Protection: All processing occurs locally on the user's machine; no screen content, detection events, or text data transmits to external servers. Users maintain complete ownership and control of all stored detection logs and can delete this data at any time.

Bias and Fairness: Training data for NLP models may contain cultural, demographic, or linguistic biases. We have mitigated this through diverse test datasets and implemented adjustable sensitivity thresholds that allow users to calibrate the system according to their specific needs and cultural context. However, ongoing bias monitoring remains essential.

Censorship and Free Expression: Automated content filtering inherently raises concerns about censorship and free expression. The system provides user-adjustable strictness levels and detection categories rather than implementing mandatory, non-configurable filtering. Users retain ultimate control over what content is flagged and how the system responds.

7.4 Future Research Directions

Several promising directions for future work have been identified:

Video Stream Processing: Extending the system to process video streams (web conferencing, video playback) would significantly expand applicability. This requires handling motion blur, encoding artifacts, and temporal consistency challenges.

Mobile Deployment: Adapting the system for Android and iOS platforms would enable protection on smartphones and tablets where screen-based abuse is equally prevalent. Mobile deployment presents unique challenges around battery efficiency and background processing limitations.

Expanded Language Support: While current English and Hindi coverage addresses significant user populations, extending support to 20+ languages would create truly global applicability. This requires both OCR model updates and abusive wordlist curation.

Adversarial Robustness: Sophisticated users may attempt to evade detection through creative obfuscation, invisible characters, or image-based text presentation. Research into adversarial robustness would improve system resilience against such evasion attempts.

Federated Learning: Implementing federated learning would enable the system to improve detection accuracy over time by learning from distributed deployments without centralizing sensitive user data.

8. Conclusion

This paper has presented a word-level blur system for real-time screen abuse detection that fundamentally addresses the limitations of traditional region-based approaches. Through integration of OCR precision, NLP understanding, and parallel processing architecture, the system achieves 95% detection accuracy with 78% less visual disruption than conventional methods.

Our implementation demonstrates that surgical content filtering is not only technically feasible but practically preferable to blunt region-based approaches that destroy user context and experience. The comprehensive evaluation across accuracy, performance, and user satisfaction dimensions validates the approach, with 91% of study participants recommending the system for broader adoption.

The open-source release of this system enables researchers and practitioners to build upon this work, extending capabilities and adapting the approach for diverse deployment contexts. As digital communication continues to evolve, we believe word-level content moderation represents the natural and necessary evolution of screen protection systems—providing effective safety while respecting user experience and contextual understanding.

Future work will focus on mobile deployment, expanded multilingual support, video stream processing, and adversarial robustness. We anticipate that word-level approaches will become the standard paradigm for real-time content moderation as the field continues to mature.

References

[1] Smith, R., "An Overview of the Tesseract OCR Engine," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 7, pp. 1234-1242, 2007.

[2] Devlin, J., Chang, M., Lee, K., and Toutanova, K., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *Proceedings of NAACL-HLT*, pp. 4171-4186, 2019.

[3] Hanley, H., Kumar, A., and Durumeric, Z., "ToxicBERT: Contextual Abuse Detection in Online Conversations," *Proceedings of the Web Conference*, pp. 2842-2848, 2021.

[4] Liu, Y., Ott, M., Goyal, N., et al., "Multilingual Denoising Pre-training for Neural Machine Translation," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 726-742, 2020.

[5] Dosovitskiy, A., Beyer, L., Kolesnikov, A., et al.,"An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *International Conference on Learning Representations*, 2021.

[6] Kumar, A., and Singh, P., "Content Moderation Challenges in Indian Language Social Media," *IEEE International Conference on Data Mining Workshops*, pp. 456-463, 2022.

[7] Zhang, T., Zhang, Y., and Zhang, Y., "Real-time Screen Content Analysis for Parental Control Applications," *ACM Symposium on User Interface Software and Technology*, pp. 892-904, 2020.

[8] Williams, J., et al., "User Experience in Content Filtering Systems: A Longitudinal Study," *Proceedings of the ACM on Human-Computer Interaction*, vol. 7, no. CSCW, pp. 1-22, 2023.

[9] Lison, P., and Tiedemann, J., "OpenSubtitles2016: Extracting Large Parallel Corpora from Movie and TV Subtitles," *Language Resources and Evaluation*, vol. 50, no. 1, pp. 147-155, 2016.

10] Mozafari, N., Farahbakhsh, R., and Crespi, N., "A BERT-based Transfer Learning Approach for Hate Speech Detection in Online Social Media," *Complex Adaptive Systems Modeling*, vol. 7, no. 1, pp. 1-18, 2019.

Author Biographies

[Your Name] received the B.Tech. degree in Computer Science from [University] in [Year], and the M.Tech. degree from [University] in [Year]. He is currently a [position] at [Institution]. His research interests include computer vision, natural language processing, real-time systems, and content moderation technologies. He has published [X] papers in international conferences and journals.

[Co-author Name] received the Ph.D. degree in Artificial Intelligence from [University] in [Year]. She is currently a [position] at [Institution] and leads the [Research Group]. Her research focuses on AI applications for social good, multilingual NLP, bias mitigation in machine learning systems, and responsible AI deployment. She has served on the program committees of [conferences].