# Technical Design Specifications: Hybrid OSINT Deepfake Detection System

Version: 1.1 (Refined for OSINT Specificity)

Target Implementation: Python 3.10+

Architectural Pattern: "Cyborg" (Signal Processing + Semantic Reasoning)

## 1. Architecture Overview

### 1.1 Pipeline Flow

The system executes a serial pipeline designed to enrich the prompt context before inference:

1. **Input:** Image File Path + Configuration (scenario, watermark\_mode).
2. **Stage 1 (Signal Processing):** forensics.py analyzes the image file for statistical anomalies (FFT, ELA, Metadata).
3. **Stage 2 (Context Injection):** detector.py constructs a dynamic system prompt based on the OSINT scenario (Auto/Military/Disaster).
4. **Stage 3 (Semantic Analysis):** VLM generates a Chain-of-Thought (CoT) reasoning block analyzing physical consistency vs. forensic data.
5. **Stage 4 (Classification):** VLM is forced into a binary A/B decision using logprob probing.

### 1.2 Latency Strategy (KV-Cache)

* **Constraint:** Do NOT initiate a new API request for the final verdict.
* **Implementation:** The Verdict call must be a continuation of the Analysis call.
  + **Request 1:** System Prompt + Image + Analysis User Prompt -> Returns history.
  + **Request 2:** Append Verdict User Prompt to history -> Returns 1 token.
* **Benefit:** This allows local engines (vLLM) and API providers to reuse the Key-Value cache for the heavy image tokens, reducing the cost/latency of the second call to near-zero.

### 1.3 Deployment Compatibility

The system must abstract the inference backend.

* **Supported Backends:** vLLM (OpenAI-compatible endpoint), OpenAI API, Google Gemini API.
* **Method:** "A/B Logprob Probing" (Provider-agnostic).
  + max\_tokens=1
  + logprobs=True (top\_k=5 minimum)

## 2. Python Module Specification (forensics.py)

**Dependencies:** opencv-python-headless, Pillow, numpy, exifread

### 2.1 Core Function

def analyze\_image(image\_path: str, context: str = "general") -> dict:  
 """  
 Orchestrates forensic tests and returns a summary dictionary.  
 Returns keys: 'fft\_status', 'ela\_status', 'metadata\_flags', 'auto\_fail\_confidence'  
 """

### 2.2 FFT Analysis Logic

* **Preprocessing:** Resize image to **512x512** (grayscale) to standardize frequency bin distribution.
* **Filtering:** Apply a High-Pass Filter (HPF) or mask the center (low frequencies) to ignore the standard "cross" pattern.
* **Detection:**
  + Calculate magnitude spectrum.
  + **Dynamic Thresholding (OSINT Tuning):**
    - Base Peak Threshold: 20 (arbitrary units, must be calibrated).
    - **IF context == "military":** Increase Threshold by **+20%** (to 24) to account for uniforms/formations.
  + Detect local maxima (peaks) exceeding threshold.
* **Output Metric:**
  + If > 3 distinct non-center peaks -> fft\_status="High Freq Artifacts".
  + Else -> fft\_status="Natural Decay".
* **Error Handling:** Wrap in try/except. Return "N/A" on failure (e.g., corrupt image).

### 2.3 ELA (Error Level Analysis) Logic

* **Process:**
  1. Load original image in memory (PIL).
  2. Save as JPEG (Quality=95) to a io.BytesIO buffer (Do NOT write to disk).
  3. Load buffer image.
  4. Compute absolute pixel difference: |Original - Resaved|.
* **Metric:** Calculate variance of the difference map.
  + Low Variance (< 2.0) -> ela\_status="Uniform (Consistent)".
    - *Note:* "Uniform" can imply *either* Original OR high-quality AI generation.
  + High Variance (>= 2.0) -> ela\_status="Inconsistent (Potential Splicing/Edit)".

### 2.4 Metadata Logic

* **Extraction:** Parse Exif data for Image Make, Image Model, and Software.
* **Empty Check:**
  + If Exif data is empty/null -> metadata\_flags=["Stripped/Social Media"]. (Neutral signal).
* **Blacklist Check:** Compare against SUSPICIOUS\_TOOLS list: ['Photoshop', 'GIMP', 'Adobe', 'Edit'].
* **Auto-Fail (Crucial):**
  + If Software contains "Midjourney", "Stable Diffusion", "ComfyUI", or "DALL-E":
  + Set auto\_fail\_confidence = 1.0.

## 3. VLM Interaction Module Specification (detector.py)

### 3.1 Interface

def detect(  
 image\_path: str,  
 scenario: str = "auto", # Options: "auto", "military", "disaster", "propaganda"  
 watermark\_mode: str = "ignore" # Options: "ignore", "analyze"  
) -> dict

### 3.2 Context Injection Logic

* **Scenario Handling:**
  + If scenario == "auto": Load the "Universal OSINT Protocol" (Case A/B/C) into the System Prompt.
  + If scenario is explicit: Append a specific overriding instruction for that domain.
* **Watermark Handling:**
  + Inject the specific rule text for "ignore" or "analyze" into the System Prompt.

### 3.3 Safety Layer ("Yellow Flag" Thresholds)

The raw probability score ($P\_{fake}$) determines the final verdict label.

* **DEEPFAKE:** $P\_{fake} > 0.90$
* **SUSPICIOUS:** $0.50 \le P\_{fake} \le 0.90$ (Requires human review).
* **AUTHENTIC:** $P\_{fake} < 0.50$

### 3.4 Output Schema

Return a strict JSON object.

{  
 "verdict": "string", // AUTHENTIC | SUSPICIOUS | DEEPFAKE  
 "confidence": float, // 0.0 to 1.0  
 "flags": [string], // List of strings: e.g. ["FFT Artifacts", "Metadata: Photoshop", "Metadata: Stripped"]  
 "analysis": "string" // The Chain-of-Thought text generated by the VLM  
}

## 4. Classification Math (The A/B Logic)

To be implemented in the post-processing of the API response:

1. **Extract Logprobs:** Get log-probability values for token "A" and token "B".
2. **Convert to Linear:** $Val\_A = e^{logprob\_A}$, $Val\_B = e^{logprob\_B}$
3. Normalize:  
   $$ Score\_{fake} = \frac{Val\_B}{Val\_A + Val\_B} $$