Slop Mop

Team 7 : Aryan Rakshit, Dhiya Pereira, Jack White, Seung Hyeon (Leah) Lee, Raj Penmetcha, Seth DeWhitt

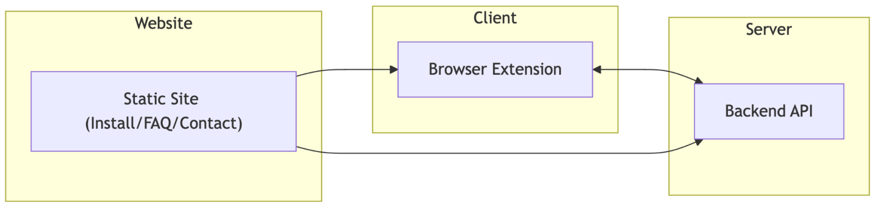
Index

Purpose (0.5 points)

The system being designed is a browser extension-based AI Content detector that analyzes content, specifically on social media/news sites, and indicates whether that content was likely generated by AI or human-made. Unlike current tools, which require users to copy/paste text or upload images to external sites, this extension performs that analysis directly within the users' feed or timeline. The extension would ingest all content that is visible and route it through an AI detection model in the backend (a Transformer-based model for text and a CNN/vision transformer model for images). This should all be done in a way to not affect the overall user experience (ex. not routing users to alternate webpages, affecting the functionality of the current site, etc). This project addresses a growing issue, which is users online being misled by AI-generated content. By giving a way for real-time, accurate detection, the aim is to reduce the exposure to misinformation and also create a clean and seamless user experience.

Design Outline (3.0 points)

A. Outline your design decisions (for example, client-server model), identify the components of your system, and describe the purpose of each component



Key Design Decisions

1. Client-Server - heavy inference off of the device and making sure models are consistent
2. Asynchronous detection + UI Overlay - User scrolling should not affecting, show loading sign
3. Two-layer caching: client cache for speed and server cache for popular posts
4. Privacy Policy enforcement: TTL deletion and anonymized metadata; local history only

a. Client-server model
   i. Client:
      1. Content Script
         a. Scrapes media feed, extracting posts.
         b. Overlays the classification and confidence percentage to the user in their social media feed.
         c. Popup menu
            i. Shows user options and customization controls
               1. site whitelist, privacy settings, accessibility settings
            ii. Toggles extension on and off
      2. Background Script:
         a. Sends media to the server to be classified
            i. includes auth logic, retry logic, rate limiting
         b. Caches past media and classification for 24 hours
         c. Stores detection history in the local database

            i. Displays user statistics, detection history/cache
    3. Local Storage
        a. stores detection history/cache and user settings
    4. Client follows privacy policy:
        a. Store past detections for 24hrs. Longer periods can be chosen by the user.
        b. Metadata sent to the server is anonymized. Images are beyond scope of anonymization.

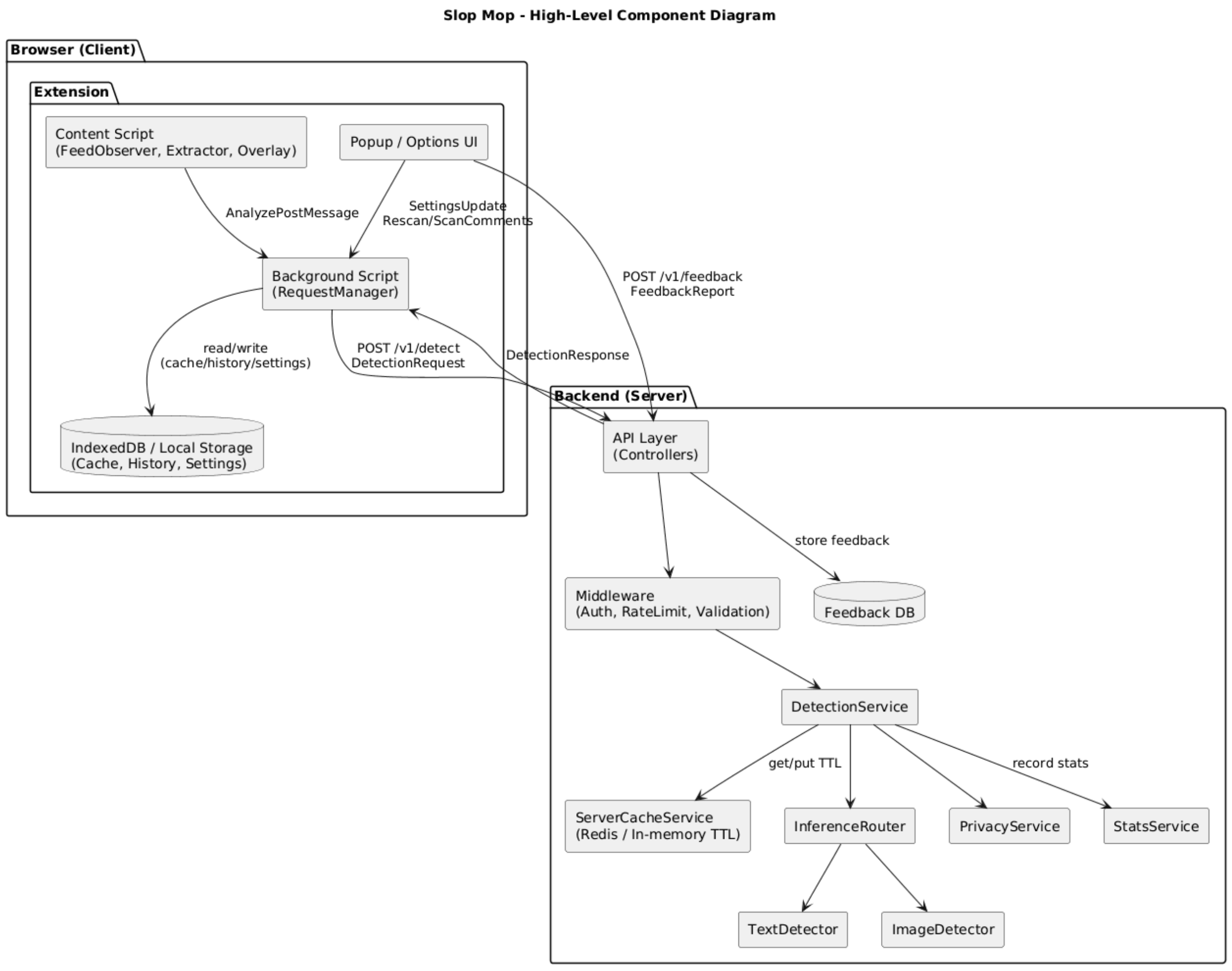
ii. Server:
    1. API Layer
        a. Accept requests with media from clients (instances of the extension)
            i. Checks for authentication (implemented as API keys per install), check cache, send to Detection Module
            ii. Returns the result of the classification to the browser extension client
    2. Detection Module
        a. Receives media from the extension server. Routes to the proper classification system for each image.
        b. Classification model returns a classification with confidence percentage to the API layer
    3. Database
        a. Cache
            i. Server cache stores only with TTL, does not store user identity
            ii. User detection history is stored locally by default and then expires based on settings (ex. 24 horus)
        b. User feedback
            i. Stores user feedback and user statistics
    4. Server follows privacy policy:
        a. Delete media within 24hrs
        b. Delete cache within 24hrs

iii. Website:
    1. HTML server
        a. Serves a webpage containing
            i. Information about our extension
            ii. Installation instructions
            iii. Download link
            iv. FAQ
            v. Contact / submit bugs
            vi. Submit review/comments
            vii. Privacy Policy

B. Describe the interactions between individual system components.

a. User scrolls through social media feed
b. Extension detects new posts
c. Extension extracts content and sends a request to the backend
d. Backend authenticates the request and checks the cache
e. If cached results exist, return immediately
f. Otherwise, if not cached, send content to the AI detection module.
g. AI module analyzes content and returns the confidence score
h. Backend returns the result to extension
i. Extension overlays confidence score and the explanation onto the social media feed
j. Timeout/network error
k. Unsupported content type or disabled site -> skips detection

C.


Slop Mop - High-Level Component Diagram
Browser (Client)
Extension
Content Script (FeedObserver, Extractor, Overlay)
Popup / Options UI
AnalyzePostMessage
SettingsUpdate Rescan/ScanComments
Background Script (RequestManager)
read/write (cache/history/settings)
POST /v1/detect DetectionRequest
DetectionResponse
POST /v1/feedback FeedbackReport
IndexedDB / Local Storage (Cache, History, Settings)
Backend (Server)
API Layer (Controllers)
store feedback
Middleware (Auth, RateLimit, Validation)
Feedback DB
DetectionService
get/put TTL
record stats
ServerCacheService (Redis / In-memory TTL)
InferenceRouter
PrivacyService
StatsService
TextDetector
ImageDetector

Design Issues (2.0 points)

Functional Issues

1. Scope of Detection : Which content should we consider for AI detection?
    a. Potential solutions
        i. We will start with text detection, then move to image detection. If time permits, we will attempt video detection.
        ii. Displaying clear notices of what our model is detecting on so the user does not assume everything is being checked for AI.
    b. Justification
        i. Text gives the best speed/accuracy tradeoff early and is easier to extract from websites. This reduces risk and helps us ship an MVP that is stable. Clear UI labeling prevents users from assuming that everything is being scanned
2. Presentation of Detection : How do we represent results in the UI?
    a. Potential Solutions
        i. To ensure there is not a large pop-up on the screen when the model signals detection, we can have an alert icon appear so the user can take note of it, but also still see what exact content is being flagged. For example, a small badge with a hover description.
        ii. Inline badge, expandable panel with details, no popups
    b. Justification: Inline badges are non-intrusive and work well while scrolling, and still allow deeper details when the user wants them
3. Detection History and Caching : How do we approach caching?
    a. Potential Solutions
        i. When faced with whether/how to store past detections, we can explore the options of storing the history locally, on a server, or not at all.
        ii. Another potential storage option to consider is local browser storage, but with a time limit.
        iii. Store user history locally only and then keep server cache only for TTL
    b. Justification
        i. Local history gives lots of user value without collecting personal browsing data. The server caching also reduces compute for reposted content while still limiting privacy risk
4. Content type detection : How do we detect a post's media type?
    a. Potential Solutions
        i. Not only to prevent potential hard crash, but also to make sure that the detection results are accurate (if text > image model or image/video > text model, there's a high change for 0% ai detection), the model will check the HTML tags for <img>, <video>, <p> or <span> tags to identify which model (image or text) it will use/send to.
        ii. Site-specific adapters (RedditAdapter, TwitterAdapter, etc.)

        iii. Mix of both DOM parsing and adapters
    b. Justification
        i. Using a hybrid of generic DOM heuristics and site specific adapters. Adapters can improve accuracy without hardcoding and heuristics work almost everywhere

Non-Functional Issues

1. Lengthy detection times : How do we balance and communicate latency to the user?
    a. Potential Solution
        i. Display “loading” symbols and properly update when the user returns to the specific post
    b. Justification
        i. This solution keeps the user informed while maintaining high accuracy.
2. Consistently inaccurate detections : How do we balance between accuracy and performance?
    a. Potential Solutions
        i. Prioritize accuracy over speed, use a more heavy-duty detection system
        ii. Specialize the system to raise the average* accuracy. For example, have the text-detection system be specialized at detecting text from popular models like GPT-5 or Gemini as opposed to generally detecting text from any LLM
    b. Justification
        i. Most text will be generated by the most popular models. We will never classify 100% of AI-generated text.
3. User social media data is being tracked due to caching detections : How do we balance caching performance vs user privacy?
    a. Potential Solutions
        i. Delete all cached detections after a short window (2 - 24 hours)
        ii. Delete all cached detections upon inactivity being detected
        iii. Delete all cached detections upon exiting the tab
        iv. Allow users to delete info as desired
    b. Justification
        i. We choose to delete all cached detections after 24hours. This allows a balance between user privacy and performance.
4. Scalability : How do we design our server for massive amounts of user requests?
    a. Potential Solutions
        i. Since our model will handle up to 1000 simultaneous detection requests, we can try the solution of having a single backend server.
        ii. Another solution is to have a serverless architecture or a horizontally scalable architecture.
    b. Justification

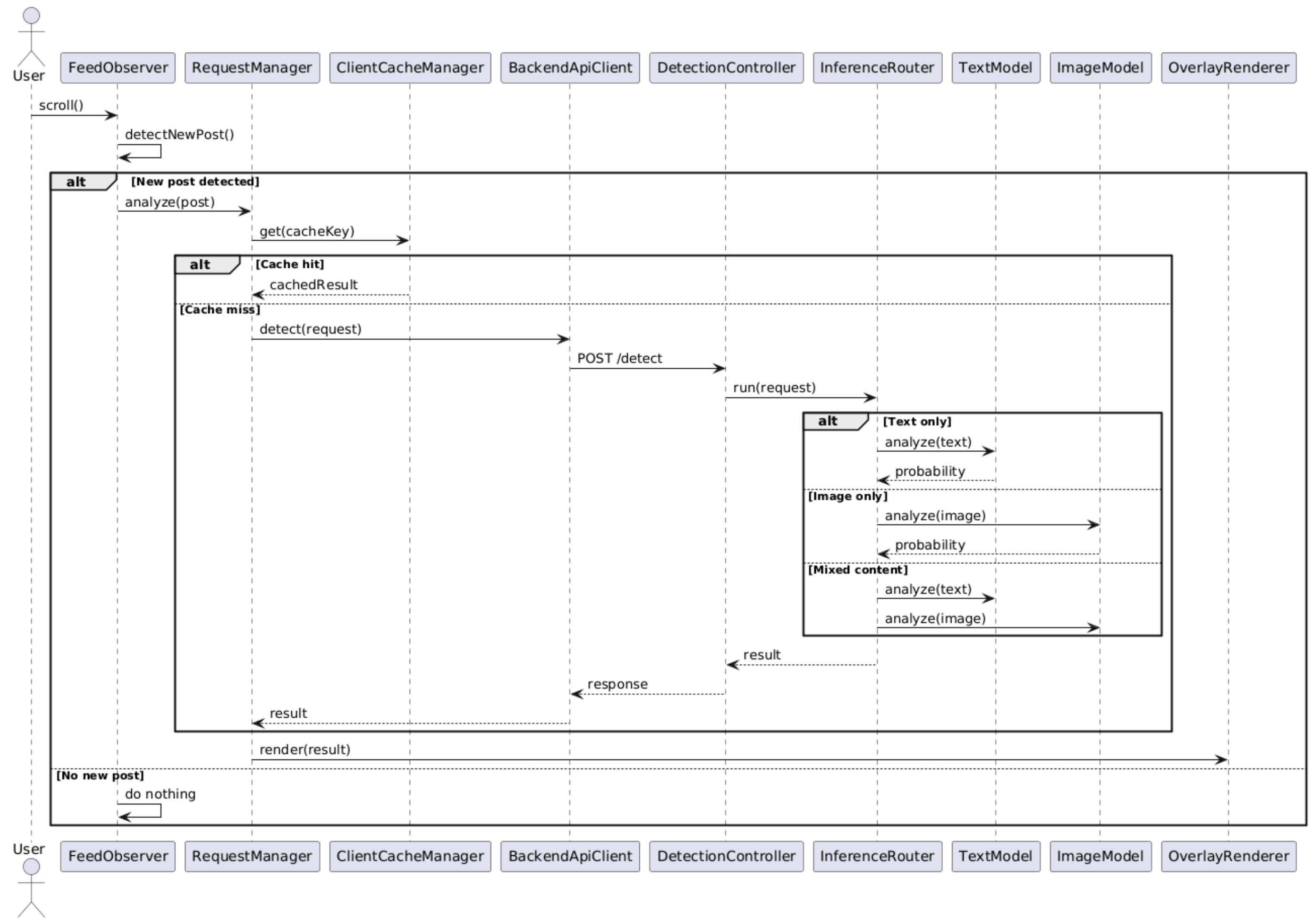
        i. For simplicity, we choose a single backend server. If latency becomes a problem, we will try hosting multiple servers with mutexes on the database. If that doesn't work, we can reevaluate and generate a new strategy.

5. Accessibility and Usability : How do we design UX for a diverse audience?
    a. Potential Solutions
        i. Ensuring we have a standard UI that is easy to navigate.
        ii. Having high contrast and large text modes.
        iii. Having a "simple" mode with less complexity.
    b. Justification
        i. Expect that an older audience will require larger text size, simpler menus. Little to no downside to implementing this with an option to enable.

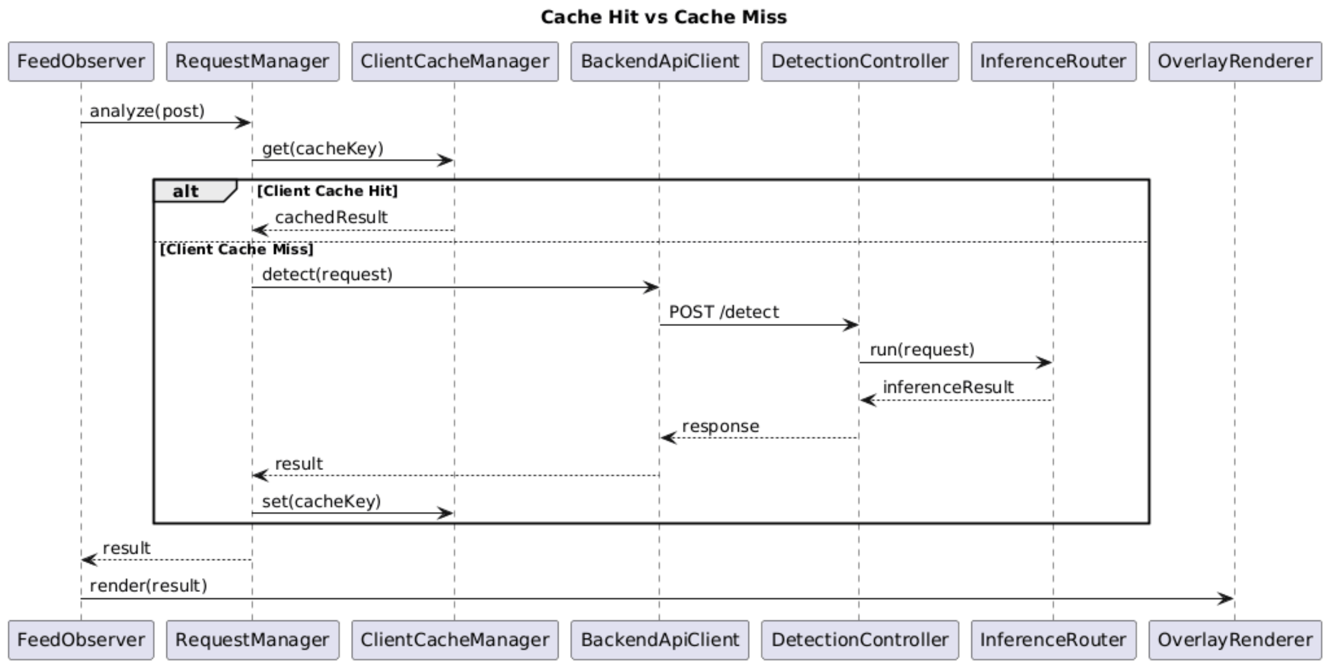
Design Details (7.5 points)

1. User scrolls through their feed
2. FeedObserver/ScrollDetector detects new post
3. FeedObserver/ScrollDetector sends the post to RequestManager
4. RequestManager sends the list of posts (data) to AnalysisHandler
5. AnalysisHandler sends the post to appropriate subclass of DetectorModel (TextModel or ImageModel)
6. DetectorModel (TextModel or ImageModel) returns probability to RequestManager
7. RequestManager sends the result to FeedObserver/ScrollDetector
8. FeedObserver/ScrollDetector updates the CSS to display result to the user

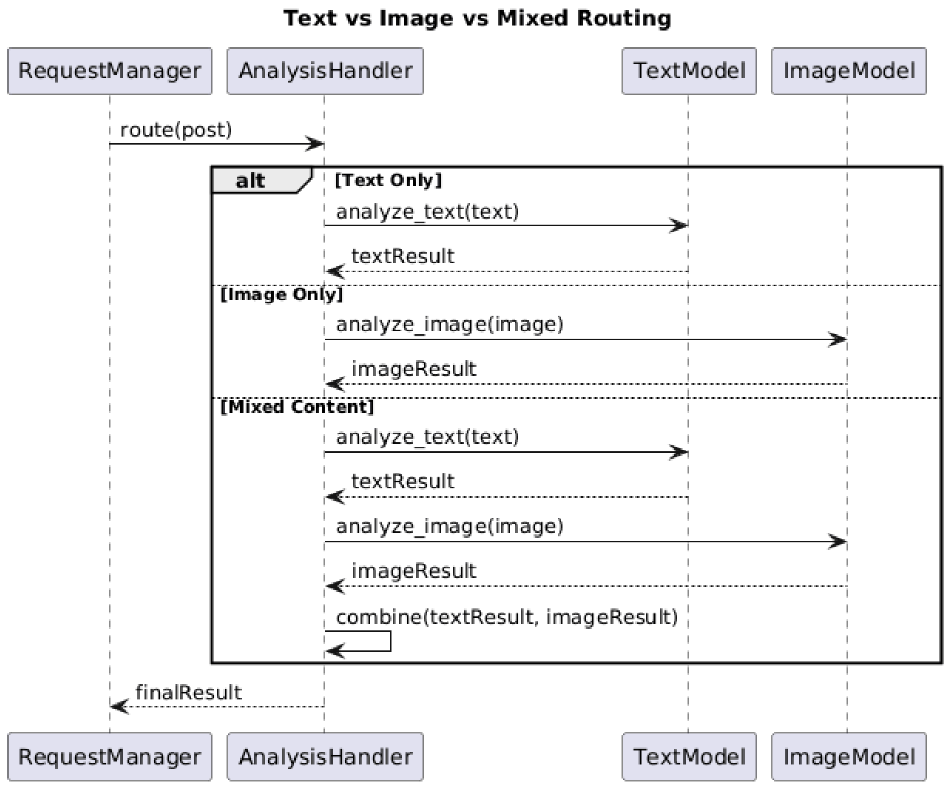
Scroll detection:



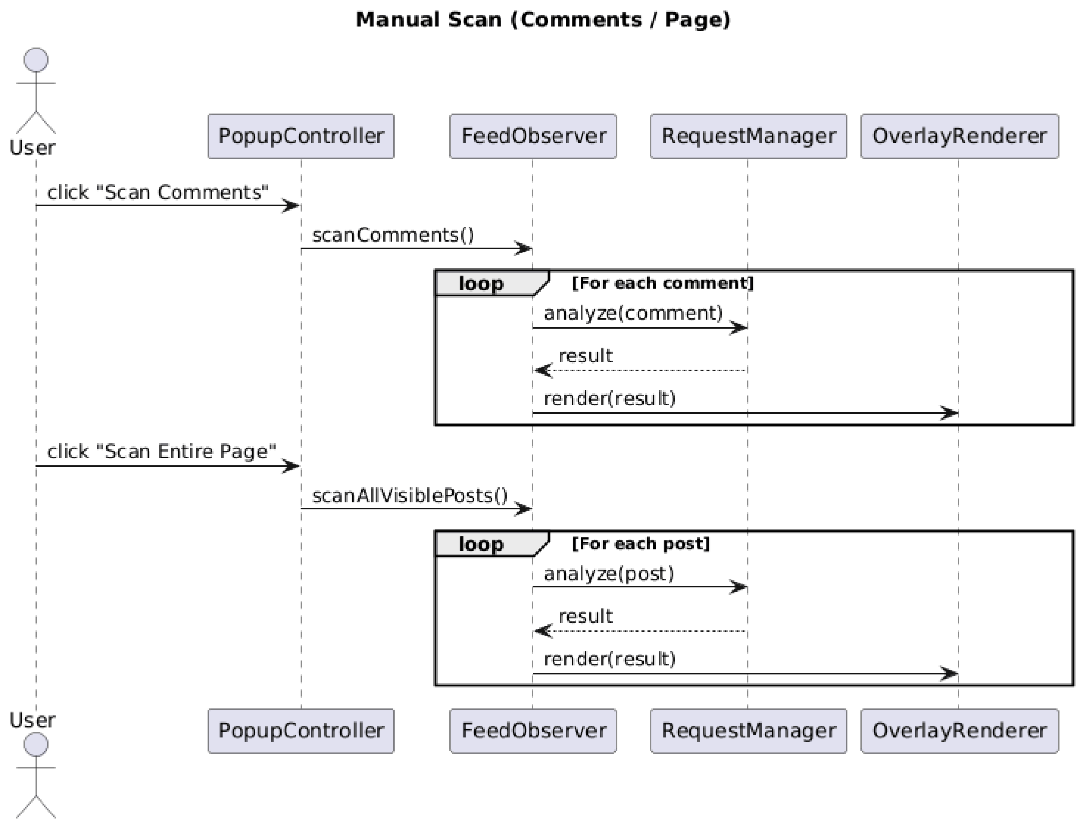
Cache hit vs. Cache miss:



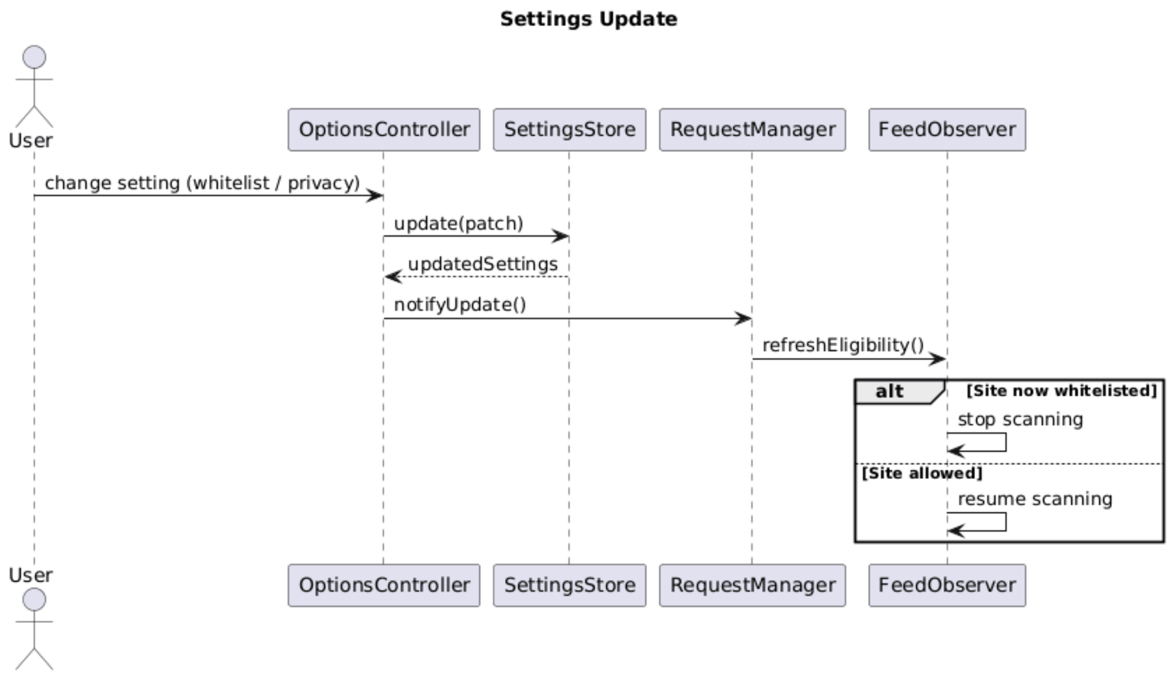
Text vs Image vs Mixed content:



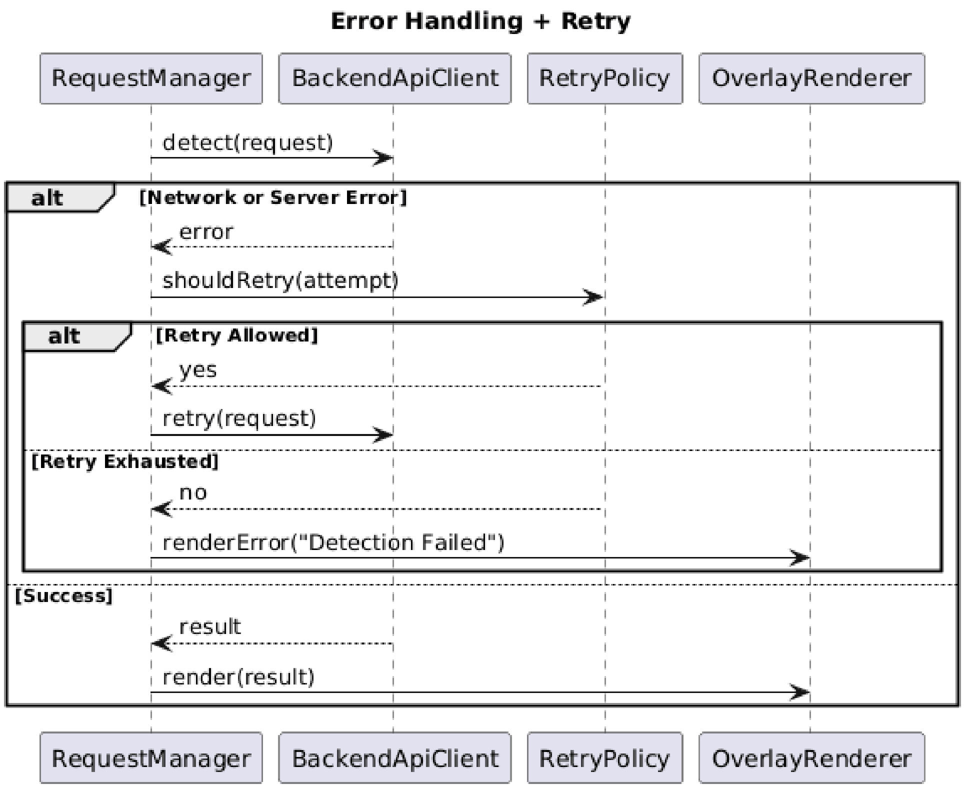
Manual scan for comments and rescanning:



Settings change flow:



Error handling and retry:



A. Data Types

- **ContentType** (enum)
    - TEXT
    - IMAGE
    - MIXED
    - UNSUPPORTED
- **SiteId**: string (e.g., "reddit.com")
- **PostId**: string
- **DetectionMode** (enum)
    - FAST
    - ACCURATE
- **UserSettings**
    - enabled: boolean
    - whitelist: SiteId[]
    - retainHistoryHours: number (default 24)
    - cacheTtlHours: number (default 24)
    - scanText: boolean (default true)
    - scanImages: boolean (default false initially)
    - scanComments: "off" | "user_triggered" | "auto_top_n"
    - uiMode: "simple" | "detailed"
    - accessibility:
        - highContrast: boolean;

        - largeText: boolean
    - privacy:
        - sendRawText: boolean;
        - sendImages: boolean;
        - allowTelemetry: boolean
    - rateLimit:
        - {maxRequestsPerMinute: number }
- **NormalizedPostContent**
    - site: SiteId
    - postId: PostId
    - url: string
    - capturedAtMs: number
    - contentType: ContentType
    - text:
        - plain: string;
        - languageHint: string
    - images[]:
        - imageId: string;
        - bytesBase64: string;
        - srcUrl: string;
        - mimeType: string
    - domContext:
        - authorHandle: string;
        - timestampText: string
- **DetectionRequest**
    - installId: string (UUID per install)
    - requestId: string (UUID per request)
    - site: SiteId
    - postId: PostId
    - contentType: ContentType
    - mode: DetectionMode
    - text:
        - plain: string
    - images[]:
        - imageId: string;
        - bytesBase64: string;
        - mimeType: string
    - clientHints:
        - extensionVersion: string;
        - platform: string
- **DetectionResponse**
    - requestId: string
    - postId: PostId
    - verdict: "likely_ai" | "likely_human" | "unknown"

    - confidence: float (0.0–1.0)
    - explanation:
        - summary: string;
        - highlights[] (optional):
            - start: number;
            - end: number;
            - reason: string
    - model:
        - name: string;
        - version: string
    - cache:
        - hit: boolean;
        - ttlRemainingMs: number
    - timing:
        - totalMs: number;
        - inferenceMs: number

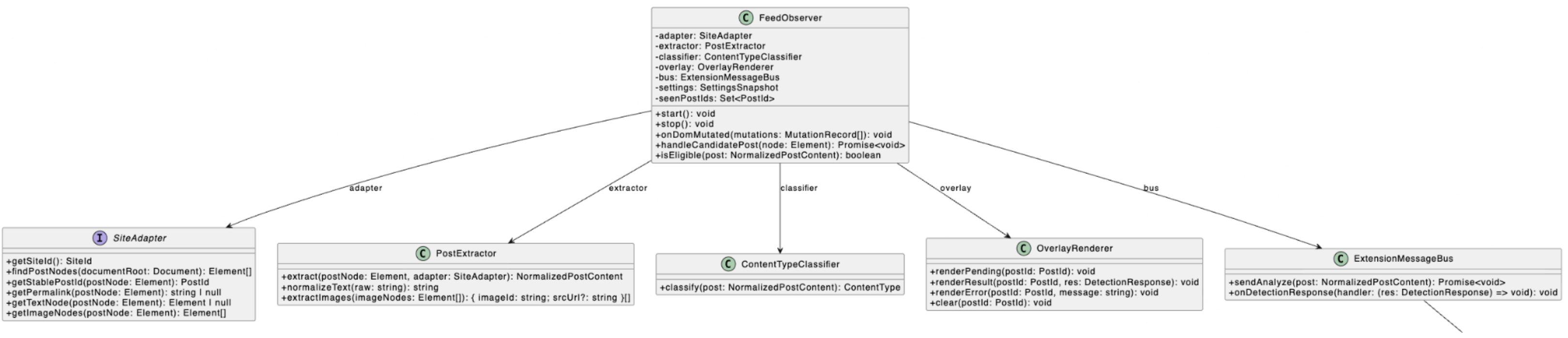
- **FeedbackReport**
    - installId: string
    - reportId: string
    - postId: PostId
    - site: SiteId
    - shownVerdict: "likely_ai" | "likely_human" | "unknown"
    - userLabel: "ai" | "human" | "not_sure"
    - notes: string
    - submittedAtMs: number
    - consent:
        - allowStoreContentSnippet: boolean
    - contentSnippet: string

B. Extension / Client

B1. Content Script

- **FeedObserver**
    - **Fields**
        - adapter: SiteAdapter
        - extractor: PostExtractor
        - classifier: ContentTypeClassifier
        - overlay: OverlayRenderer
        - bus: ExtensionMessageBus
        - settings: SettingsSnapshot
        - seenPostIds: Set<PostId>
    - **Methods**

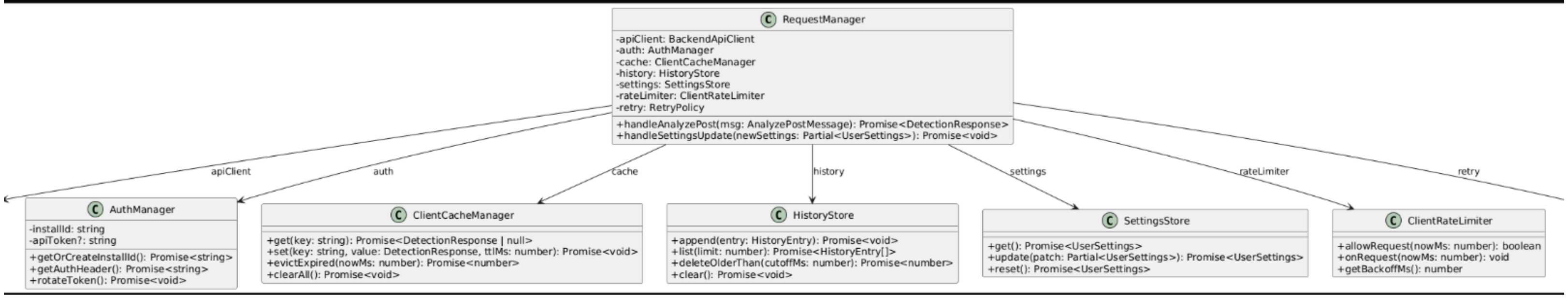
        - start(): void
        - stop(): void
        - onDomMutated(mutations: MutationRecord[]): void
        - handleCandidatePost(node: Element): Promise<void>
        - isEligible(post: NormalizedPostContent): boolean
- **SiteAdapter** (interface)
    - **Methods**
        - getSiteId(): SiteId
        - findPostNodes(documentRoot: Document): Element[]
        - getStablePostId(postNode: Element): PostId
        - getPermalink(postNode: Element): string | null
        - getTextNode(postNode: Element): Element | null
        - getImageNodes(postNode: Element): Element[]
- **PostExtractor**
    - **Methods**
        - extract(postNode: Element, adapter: SiteAdapter): NormalizedPostContent
        - normalizeText(raw: string): string
        - extractImages(imageNodes: Element[]): -> { imageId: string; srcUrl?: string }[]
- **ContentTypeClassifier**
    - **Methods**
        - classify(post: NormalizedPostContent): ContentType
- **OverlayRenderer**
    - **Methods**
        - renderPending(postId: PostId): void
        - renderResult(postId: PostId, res: DetectionResponse): void
        - renderError(postId: PostId, message: string): void
        - clear(postId: PostId): void
- **ExtensionMessageBus** (content-script side)
    - **Methods**
        - sendAnalyze(post: NormalizedPostContent): Promise<void>
        - onDetectionResponse(handler: (res: DetectionResponse) => void): void


FeedObserver
-adapter: SiteAdapter
-extractor: PostExtractor
-classifier: ContentTypeClassifier
-overlay: OverlayRenderer
-bus: ExtensionMessageBus
-settings: SettingsSnapshot
-seenPostIds: Set<PostId>
+start(): void
+stop(): void
+onDomMutated(mutations: MutationRecord[]): void
+handleCandidatePost(node: Element): Promise<void>
+isEligible(post: NormalizedPostContent): boolean
adapter
extractor
classifier
overlay
bus
SiteAdapter
+getSiteId(): SiteId
+findPostNodes(documentRoot: Document): Element[]
+getStablePostId(postNode: Element): PostId
+getPermalink(postNode: Element): string | null
+getTextNode(postNode: Element): Element | null
+getImageNodes(postNode: Element): Element[]
PostExtractor
+extract(postNode: Element, adapter: SiteAdapter): NormalizedPostContent
+normalizeText(raw: string): string
+extractImages(imageNodes: Element[]): { imageId: string; srcUrl?: string }[]
ContentTypeClassifier
+classify(post: NormalizedPostContent): ContentType
OverlayRenderer
+renderPending(postId: PostId): void
+renderResult(postId: PostId, res: DetectionResponse): void
+renderError(postId: PostId, message: string): void
+clear(postId: PostId): void
ExtensionMessageBus
+sendAnalyze(post: NormalizedPostContent): Promise<void>
+onDetectionResponse(handler: (res: DetectionResponse) => void): void

B2. Background Script (service worker)

- **RequestManager**
    - **Fields**
        - apiClient: BackendApiClient
        - auth: AuthManager
        - cache: ClientCacheManager
        - history: HistoryStore
        - settings: SettingsStore
        - rateLimiter: ClientRateLimiter
        - retry: RetryPolicy
    - **Methods**
        - handleAnalyzePost(msg: AnalyzePostMessage): Promise<DetectionResponse>
        - handleSettingsUpdate(newSettings: Partial<UserSettings>): Promise<void>
- **BackendApiClient**
    - **Fields**
        - baseUrl: string
        - timeoutMs: number
    - **Methods**
        - detect(req: DetectionRequest, authHeader: string): Promise<DetectionResponse>
        - submitFeedback(report: FeedbackReport, authHeader: string): Promise<{ ok: boolean }>
        - health(): Promise<{ ok: boolean; version: string }> (optional)
- **AuthManager**
    - **Fields**
        - installId: string
        - apiToken?: string
    - **Methods**
        - getOrCreateInstallId(): Promise<string>
        - getAuthHeader(): Promise<string>
        - rotateToken(): Promise<void> (optional)
- **ClientCacheManager** (IndexedDB)
    - **Methods**
        - get(key: string): Promise<DetectionResponse | null>
        - set(key: string, value: DetectionResponse, ttlMs: number): Promise<void>
        - evictExpired(nowMs: number): Promise<number>
        - clearAll(): Promise<void>
    - **Key derivation**
        - key = hash(site + postId + contentFingerprint + mode + modelPolicyVersion)
        - contentFingerprint = hash(normalize(text)) for text; for images may start with hash(srcUrl) and later use hash(bytes)

- **HistoryStore** (IndexedDB)
    - **Types**
        - HistoryEntry: { site: SiteId; postId: PostId; timestampMs: number; verdict: string; confidence: number; mode: DetectionMode }
    - **Methods**
        - append(entry: HistoryEntry): Promise<void>
        - list(limit: number): Promise<HistoryEntry[]>
        - deleteOlderThan(cutoffMs: number): Promise<number>
        - clear(): Promise<void>
- **SettingsStore**
    - **Methods**
        - get(): Promise<UserSettings>
        - update(patch: Partial<UserSettings>): Promise<UserSettings>
        - reset(): Promise<UserSettings>
- **ClientRateLimiter**
    - **Methods**
        - allowRequest(nowMs: number): boolean
        - onRequest(nowMs: number): void
        - getBackoffMs(): number
- **RetryPolicy**
    - **Methods**
        - shouldRetry(err: unknown, attempt: number): boolean
        - nextDelayMs(attempt: number): number


RequestManager
-apiClient: BackendApiClient
-auth: AuthManager
-cache: ClientCacheManager
-history: HistoryStore
-settings: SettingsStore
-rateLimiter: ClientRateLimiter
-retry: RetryPolicy
+handleAnalyzePost(msg: AnalyzePostMessage): Promise<DetectionResponse>
+handleSettingsUpdate(newSettings: Partial<UserSettings>): Promise<void>
apiClient
auth
cache
history
settings
rateLimiter
retry
AuthManager
-installId: string
-apiToken?: string
+getOrCreateInstallId(): Promise<string>
+getAuthHeader(): Promise<string>
+rotateToken(): Promise<void>
ClientCacheManager
+get(key: string): Promise<DetectionResponse | null>
+set(key: string, value: DetectionResponse, ttlMs: number): Promise<void>
+evictExpired(nowMs: number): Promise<number>
+clearAll(): Promise<void>
HistoryStore
+append(entry: HistoryEntry): Promise<void>
+list(limit: number): Promise<HistoryEntry[]>
+deleteOlderThan(cutoffMs: number): Promise<number>
+clear(): Promise<void>
SettingsStore
+get(): Promise<UserSettings>
+update(patch: Partial<UserSettings>): Promise<UserSettings>
+reset(): Promise<UserSettings>
ClientRateLimiter
+allowRequest(nowMs: number): boolean
+onRequest(nowMs: number): void
+getBackoffMs(): number

B3. Popup UI + Options UI

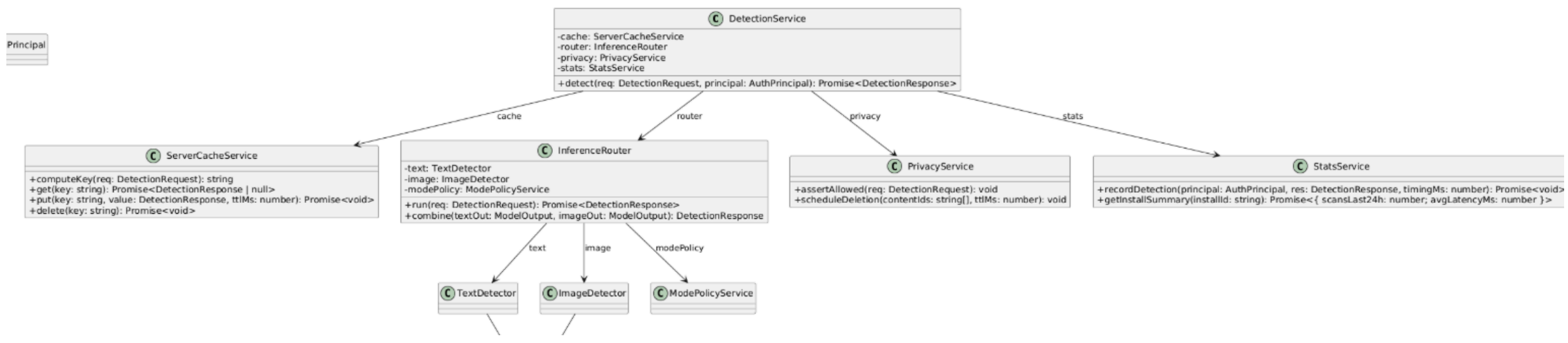
- **PopupController**
    - **Methods**
        - init(): Promise<void>
        - toggleEnabled(enabled: boolean): Promise<void>
        - renderRecentHistory(): Promise<void>
        - rescanCurrentPage(): Promise<void>
- **OptionsController**
    - **Methods**

- init(): Promise<void>
- updateWhitelist(sites: SiteId[]): Promise<void>
- updatePrivacySettings(patch: UserSettings["privacy"]): Promise<void>
- updateAccessibility(patch: UserSettings["accessibility"]): Promise<void>
- clearHistory(): Promise<void>

C. Backend / Server

C1. API layer (HTTP)

- **DetectionController**
  - **Endpoint**
    - POST /v1/detect → DetectionResponse
  - **Method**
    - detect(req: DetectionRequest, ctx: RequestContext): Promise<DetectionResponse>
- **FeedbackController**
  - **Endpoint**
    - POST /v1/feedback → { ok: boolean }
  - **Method**
    - submit(report: FeedbackReport, ctx: RequestContext): Promise<{ ok: boolean }>
- **AuthMiddleware**
  - authenticate(headers: Record<string,string>): Promise<AuthPrincipal>
- **RateLimitMiddleware**
  - check(principal: AuthPrincipal, nowMs: number): Promise<void>
- **ValidationMiddleware**
  - validateDetectRequest(req: DetectionRequest): void


Principal
DetectionService
-cache: ServerCacheService
-router: InferenceRouter
-privacy: PrivacyService
-stats: StatsService
+detect(req: DetectionRequest, principal: AuthPrincipal): Promise<DetectionResponse>
cache
router
privacy
stats
ServerCacheService
+computeKey(req: DetectionRequest): string
+get(key: string): Promise<DetectionResponse | null>
+put(key: string, value: DetectionResponse, ttlMs: number): Promise<void>
+delete(key: string): Promise<void>
InferenceRouter
-text: TextDetector
-image: ImageDetector
-modePolicy: ModePolicyService
+run(req: DetectionRequest): Promise<DetectionResponse>
+combine(textOut: ModelOutput, imageOut: ModelOutput): DetectionResponse
PrivacyService
+assertAllowed(req: DetectionRequest): void
+scheduleDeletion(contentIds: string[], ttlMs: number): void
StatsService
+recordDetection(principal: AuthPrincipal, res: DetectionResponse, timingMs: number): Promise<void>
+getInstallSummary(installId: string): Promise<{ scansLast24h: number; avgLatencyMs: number }>
text
image
modePolicy
TextDetector
ImageDetector
ModePolicyService

-

C2. Detection + cache

- **DetectionService**

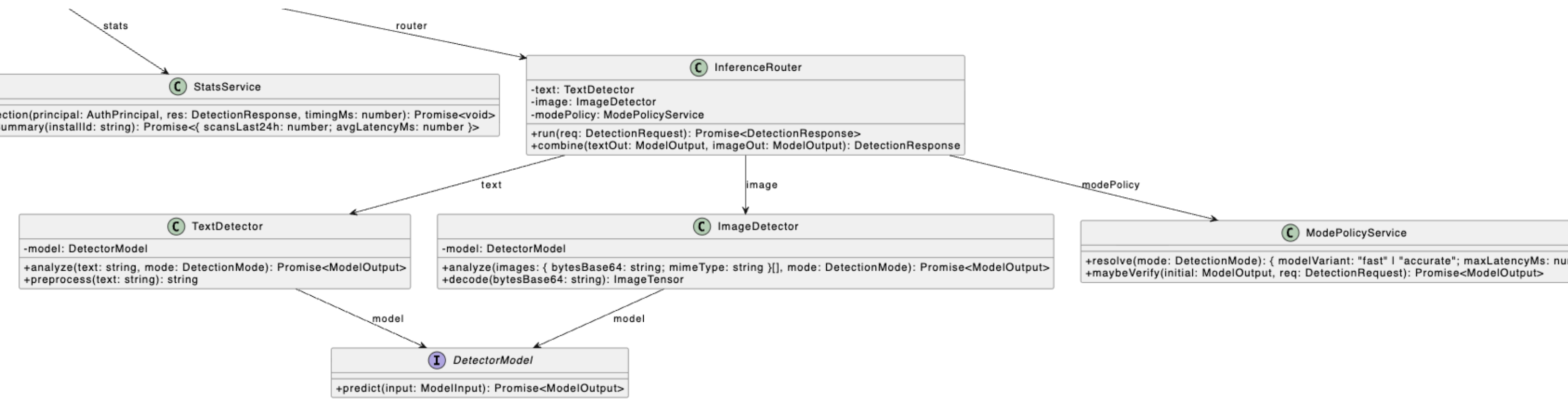
- **Fields**
    - cache: ServerCacheService
    - router: InferenceRouter
    - privacy: PrivacyService
    - stats: StatsService
- **Methods**
    - detect(req: DetectionRequest, principal: AuthPrincipal): Promise<DetectionResponse>

- **ServerCacheService** (Redis or in-memory + TTL)
    - **Methods**
        - computeKey(req: DetectionRequest): string
            - key = HMAC_SHA256(serverSecret, normalizedContent + mode + modelVersion)
        - get(key: string): Promise<DetectionResponse | null>
        - put(key: string, value: DetectionResponse, ttlMs: number): Promise<void>
        - delete(key: string): Promise<void>
- **PrivacyService**
    - **Methods**
        - assertAllowed(req: DetectionRequest): void
        - scheduleDeletion(contentIds: string[], ttlMs: number): void

C3. Detection module (routing + models)

- **InferenceRouter**
    - **Fields**
        - text: TextDetector
        - image: ImageDetector
        - modePolicy: ModePolicyService
    - **Methods**
        - run(req: DetectionRequest): Promise<DetectionResponse>
        - combine(textOut: ModelOutput, imageOut: ModelOutput): DetectionResponse
- **DetectorModel** (interface)
    - predict(input: ModelInput): Promise<ModelOutput>
- **TextDetector**
    - **Fields**
        - model: DetectorModel
            - Note: preliminary plan is to use desklib/ai-text-detector-v1.01 on HuggingFace
    - **Methods**
        - analyze(text: string, mode: DetectionMode): Promise<ModelOutput>

        - preprocess(text: string): string
- **ImageDetector**
    - **Fields**
        - model: DetectorModel
            - Note: preliminary plan is to use capcheck/ai-image-detection model on HuggingFace
    - **Methods**
        - analyze(images: { bytesBase64: string; mimeType: string }[], mode: DetectionMode): Promise<ModelOutput>
        - decode(bytesBase64: string): ImageTensor
- **ModePolicyService**
    - **Methods**
        - resolve(mode: DetectionMode): { modelVariant: "fast" | "accurate"; maxLatencyMs: number }
        - maybeVerify(initial: ModelOutput, req: DetectionRequest): Promise<ModelOutput>
- **Model I/O**
    - ModelInput (union)
        - { kind: "text"; text: string }
        - { kind: "image"; tensor: ImageTensor }
    - ModelOutput
        - { verdict: "likely_ai" | "likely_human" | "unknown"; confidence: number; explanation (optional): { summary: string; highlights?: ... } }


stats
router
StatsService
ction(principal: AuthPrincipal, res: DetectionResponse, timingMs: number): Promise<void>
ummary(installId: string): Promise<{ scansLast24h: number; avgLatencyMs: number }>
InferenceRouter
-text: TextDetector
-image: ImageDetector
-modePolicy: ModePolicyService
+run(req: DetectionRequest): Promise<DetectionResponse>
+combine(textOut: ModelOutput, imageOut: ModelOutput): DetectionResponse
text
image
modePolicy
TextDetector
-model: DetectorModel
+analyze(text: string, mode: DetectionMode): Promise<ModelOutput>
+preprocess(text: string): string
ImageDetector
-model: DetectorModel
+analyze(images: { bytesBase64: string; mimeType: string }[], mode: DetectionMode): Promise<ModelOutput>
+decode(bytesBase64: string): ImageTensor
ModePolicyService
+resolve(mode: DetectionMode): { modelVariant: "fast" | "accurate"; maxLatencyMs: nu
+maybeVerify(initial: ModelOutput, req: DetectionRequest): Promise<ModelOutput>
model
model
DetectorModel
+predict(input: ModelInput): Promise<ModelOutput>

C4. Persistence (feedback + aggregate stats)

- **FeedbackRepository**

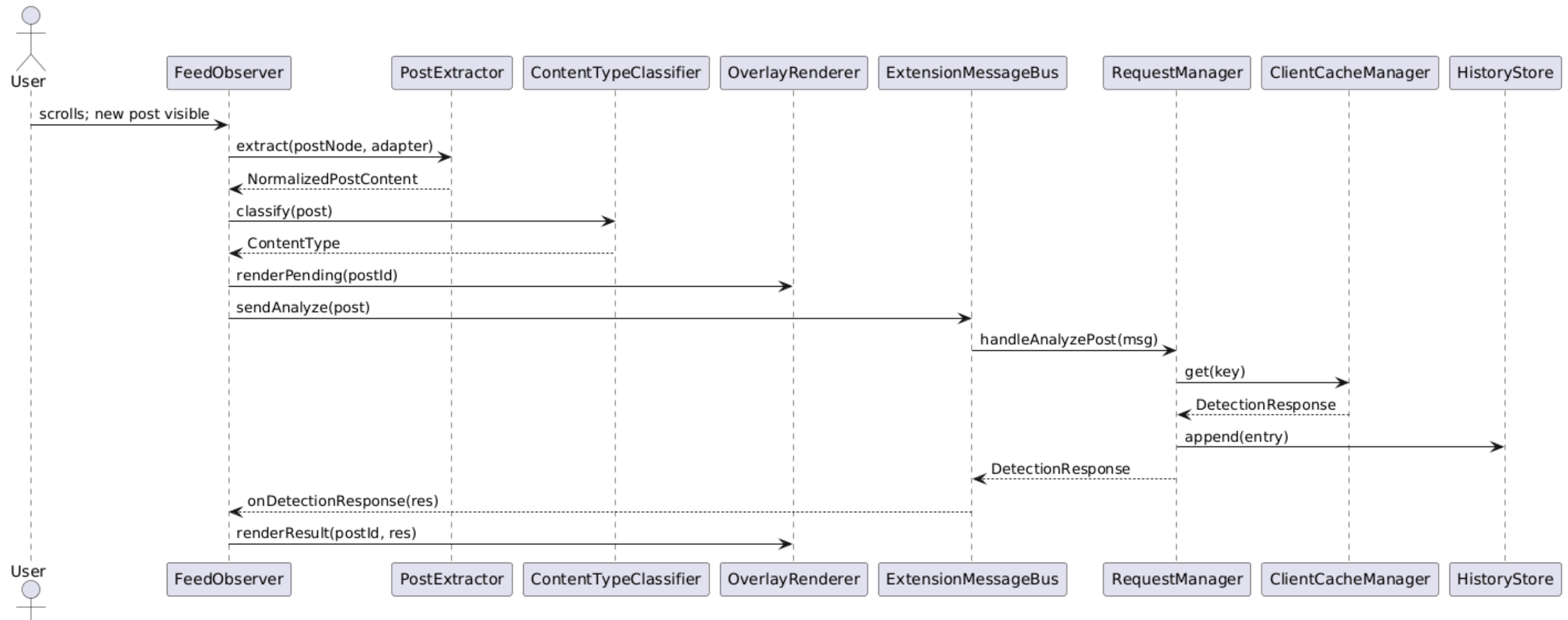
        - **Methods**
            - insert(report: FeedbackReport): Promise<void>
            - listRecent(limit: number): Promise<FeedbackReport[]>
- **StatsService**
    - **Methods**
        - recordDetection(principal: AuthPrincipal, res: DetectionResponse, timingMs: number): Promise<void>
        - getInstallSummary(installId: string): Promise<{ scansLast24h: number; avgLatencyMs: number }> (optional)

D. Website

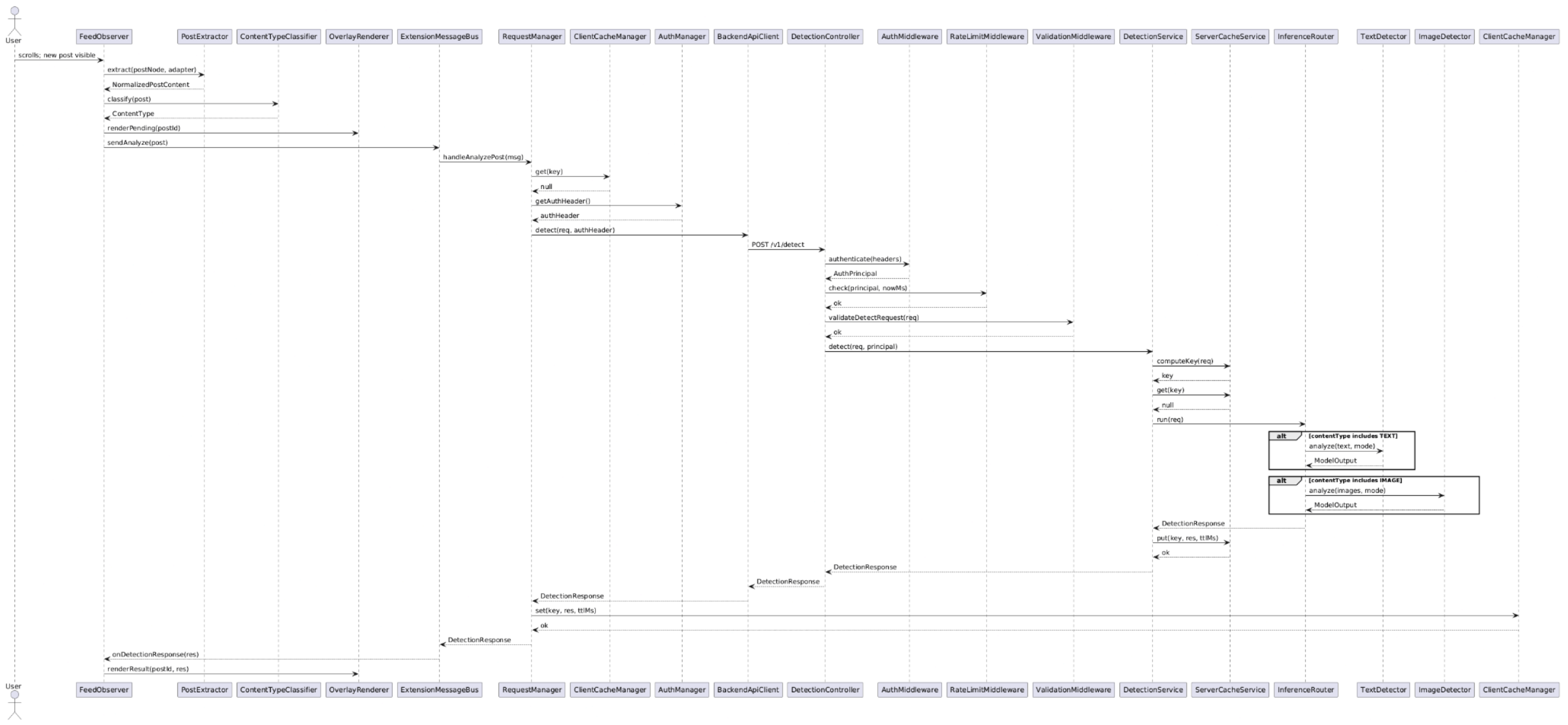
- **StaticSiteServer**
    - **Routes**
        - GET /
        - GET /install
        - GET /faq
        - GET /privacy
        - GET /download
        - POST /contact (optional)
    - **Methods**
        - serveStatic(path: string): HttpResponse
        - submitContact(form: ContactForm): Promise<{ ok: boolean }> (optional)

E. Runtime Interaction Walkthrough

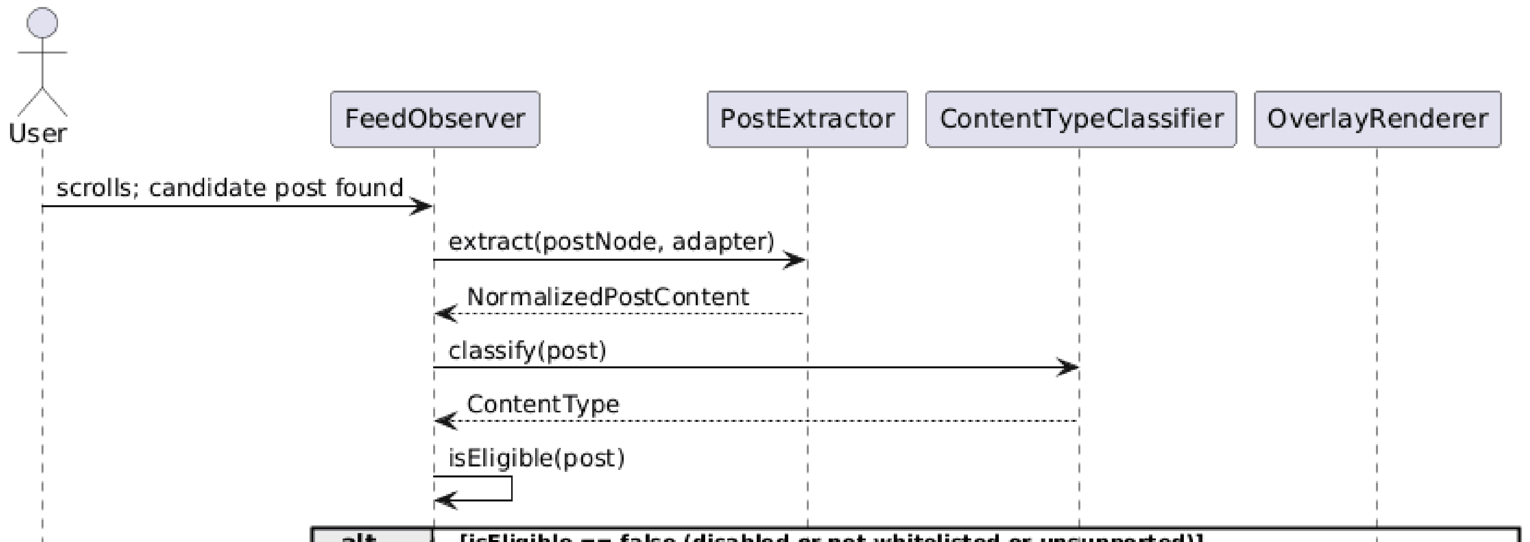
- **E1. User scrolls, detects new post, extension loads cached result**
    - Content script: FeedObserver → PostExtractor → OverlayRenderer.renderPending → ExtensionMessageBus.sendAnalyze
    - Background: RequestManager.handleAnalyzePost → ClientCacheManager.get → HistoryStore.append → response
    - Content script: OverlayRenderer.renderResult



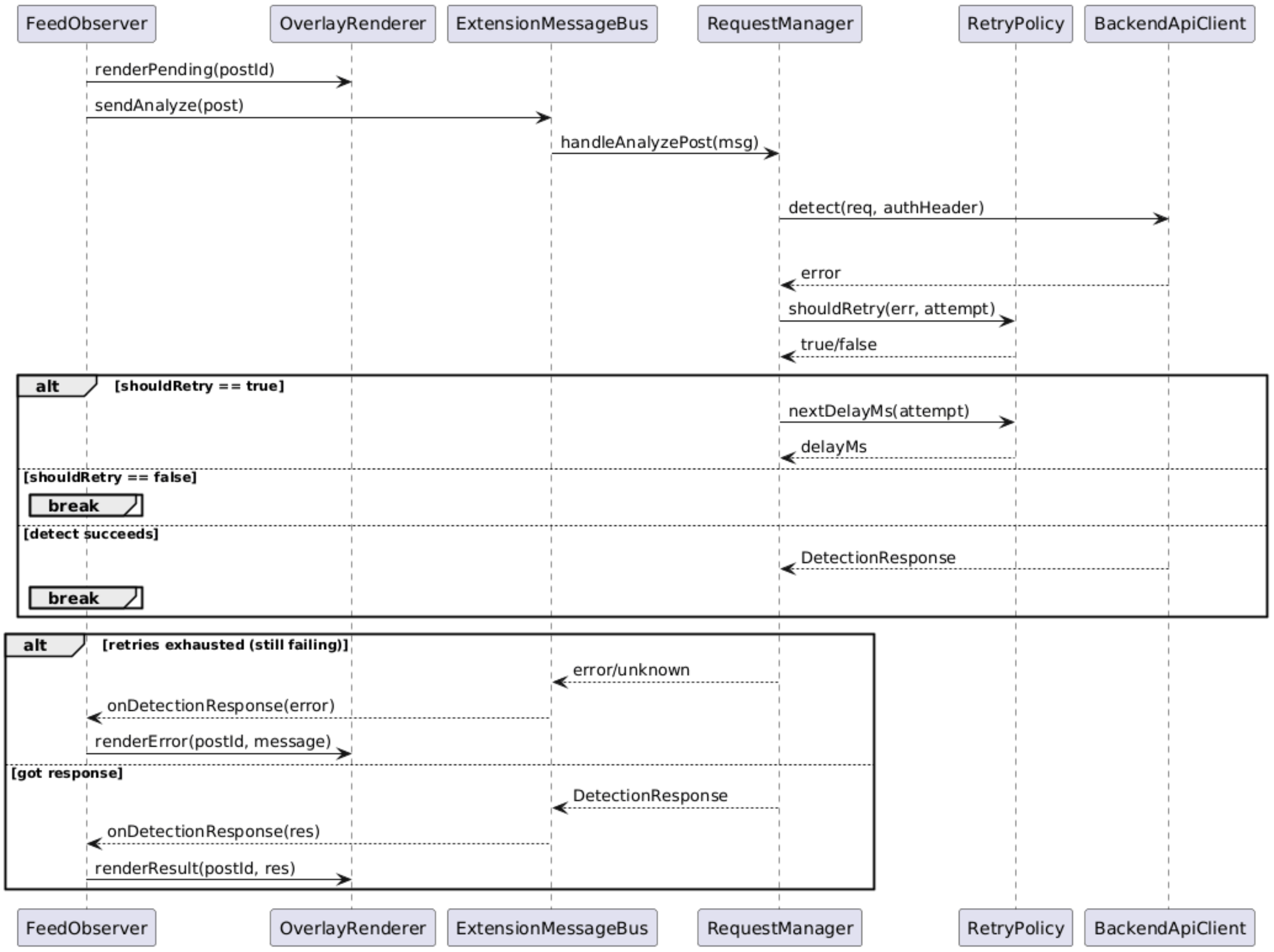
-
- **E2. User scrolls, detects new post, client and cache miss, then inference**
    - Background: AuthManager.getAuthHeader → BackendApiClient.detect
    - Server: DetectionController.detect → auth/rate-limit/validate → DetectionService.detect
    - Server: ServerCacheService.get miss → InferenceRouter.run → TextDetector.analyze / ImageDetector.analyze → ServerCacheService.put
    - Background: ClientCacheManager.set → HistoryStore.append
    - Content script: OverlayRenderer.renderResult



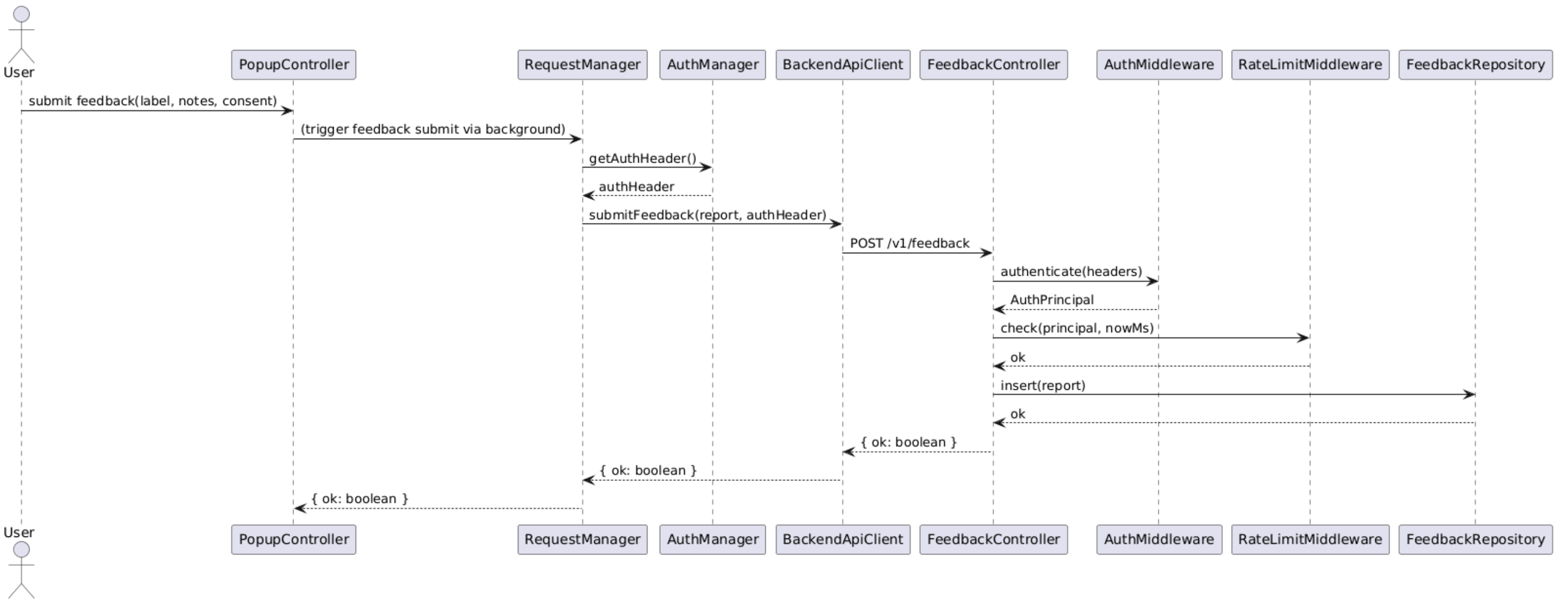
-
- **E3. Extension Disabled or website not whitelisted**
    - Content script: FeedObserver.isEligible false
    - No request sent because handleCandidatePost() returns if not FeedObserver.isEligible



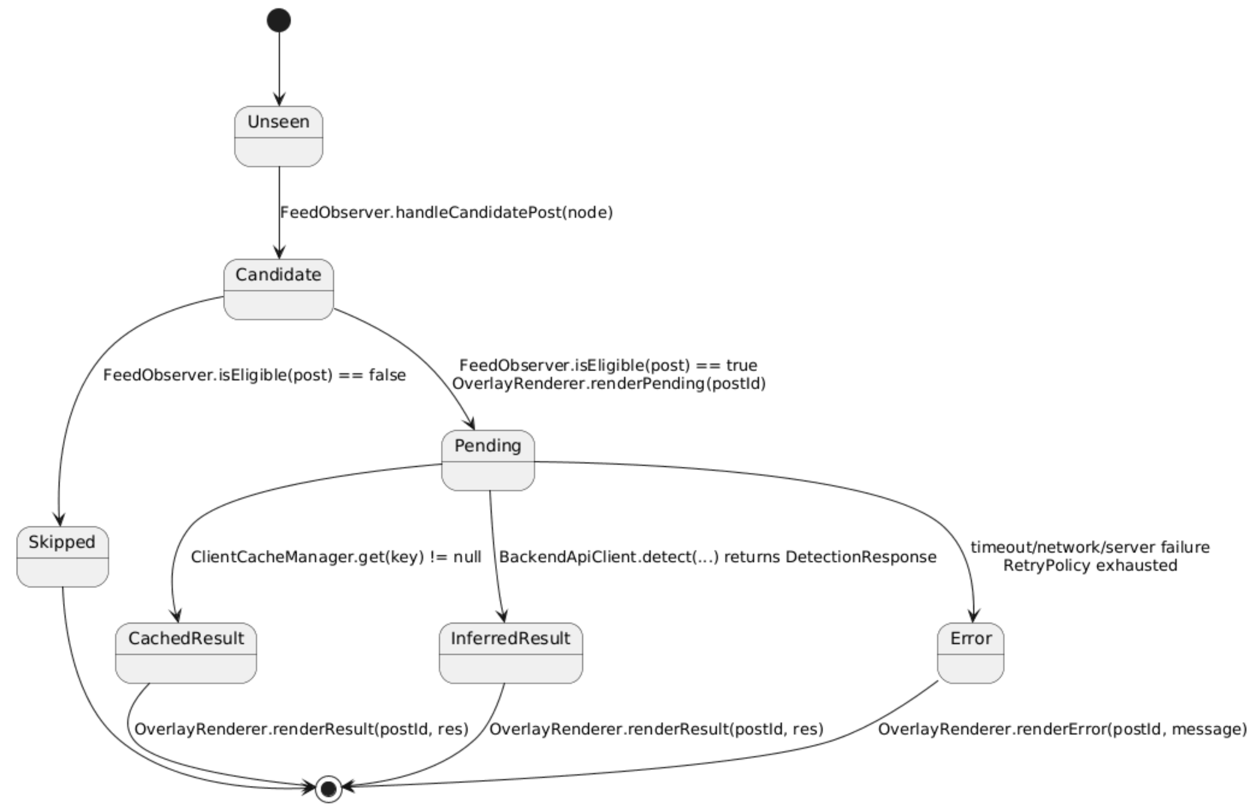
-
- **E4. Error occurs**
    - Background: BackendApiClient.detect throws error → RetryPolicy backoff → if still failing return error/unknown
    - Content script: OverlayRenderer.renderError



- **E5. User submits feedback**
    - UI: collect label/notes/consent → background BackendApiClient.submitFeedback
    - Server: FeedbackController.submit → FeedbackRepository.insert

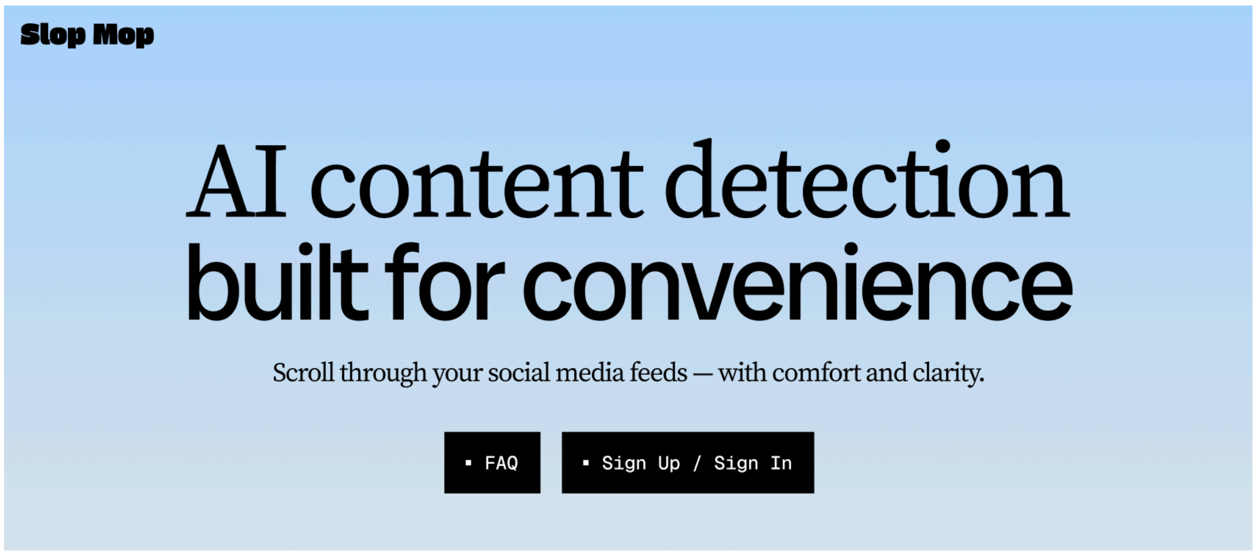

User
PopupController
RequestManager
AuthManager
BackendApiClient
FeedbackController
AuthMiddleware
RateLimitMiddleware
FeedbackRepository
submit feedback(label, notes, consent)
(trigger feedback submit via background)
getAuthHeader()
authHeader
submitFeedback(report, authHeader)
POST /v1/feedback
authenticate(headers)
AuthPrincipal
check(principal, nowMs)
ok
insert(report)
ok
{ ok: boolean }
{ ok: boolean }
{ ok: boolean }

F. State Diagram

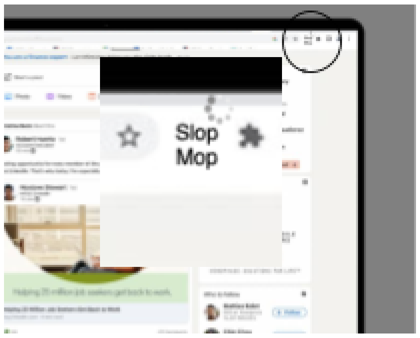

Unseen
FeedObserver.handleCandidatePost(node)
Candidate
FeedObserver.isEligible(post) == false
FeedObserver.isEligible(post) == true
OverlayRenderer.renderPending(postId)
Pending
Skipped
ClientCacheManager.get(key) != null
BackendApiClient.detect(...) returns DetectionResponse
timeout/network/server failure
RetryPolicy exhausted
CachedResult
InferredResult
Error
OverlayRenderer.renderResult(postId, res)
OverlayRenderer.renderResult(postId, res)
OverlayRenderer.renderError(postId, message)

G. UI Mockups

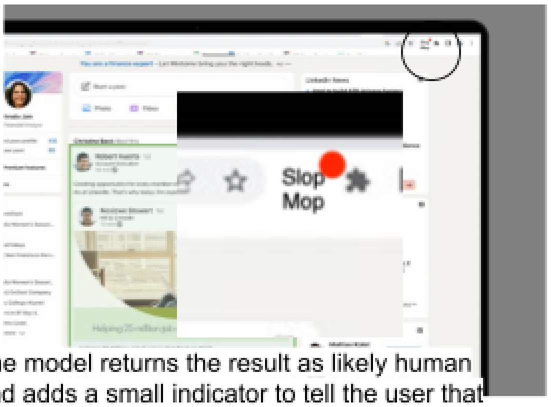
Basic homepage mockup:



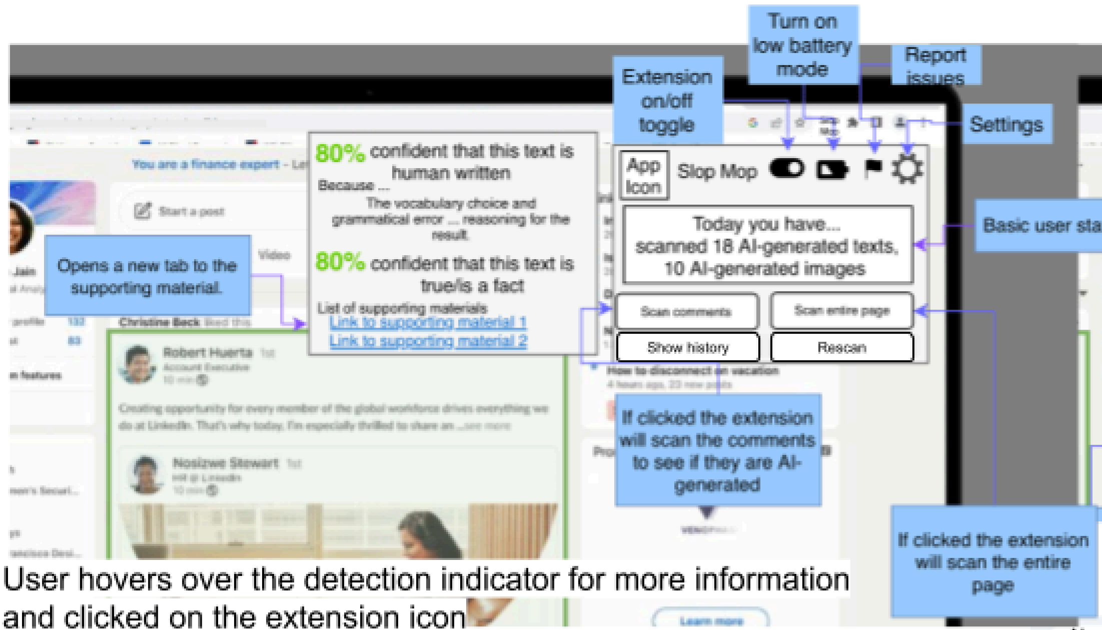
This is the homepage that users will see when first navigating to the website. Skeptical users can then navigate to the FAQs page and others can sign up to use the extension



User is scrolling through their feed. The model detects new post and is analysing it.



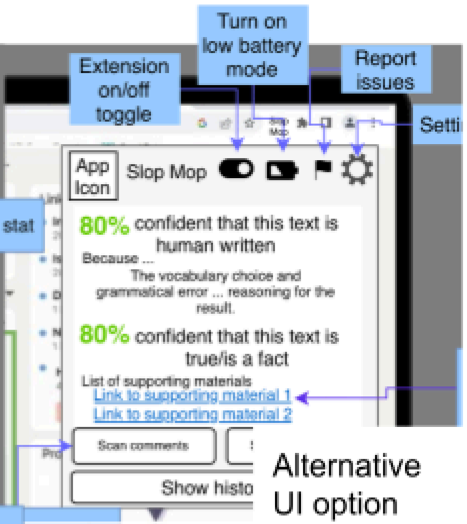
The model returns the result as likely human and adds a small indicator to tell the user that the analysis has been done.


Turn on low battery mode
Report issues
Extension on/off toggle
Settings
App Icon
Slop Mop
Basic user stat
Opens a new tab to the supporting material.
Scan comments
Scan entire page
Show history
Rescan
If clicked the extension will scan the comments to see if they are AI-generated
If clicked the extension will scan the entire page

80% confident that this text is human written
Because ...
The vocabulary choice and grammatical error ... reasoning for the result.
80% confident that this text is true/is a fact
List of supporting materials
Link to supporting material 1
Link to supporting material 2

Today you have... scanned 18 AI-generated texts, 10 AI-generated images

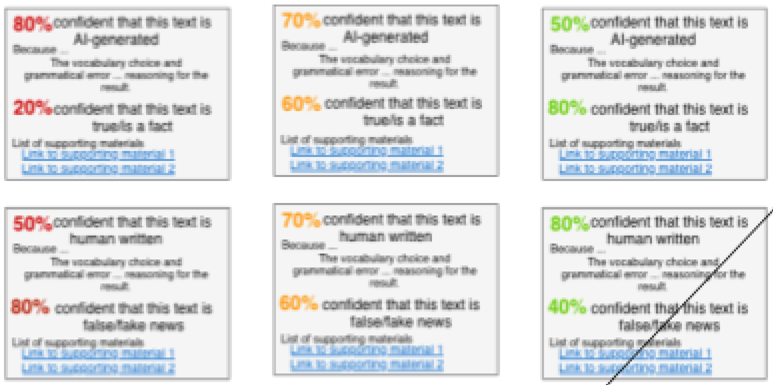
User hovers over the detection indicator for more information and clicked on the extension icon



80% confident that this text is human written
Because ...
The vocabulary choice and grammatical error ... reasoning for the result.
80% confident that this text is true/is a fact
List of supporting materials
Link to supporting material 1
Link to supporting material 2

Alternative UI option

Example results



80% confident that this text is AI-generated
Because ...
The vocabulary choice and grammatical error ... reasoning for the result.
20% confident that this text is true/is a fact
List of supporting materials
Link to supporting material 1
Link to supporting material 2

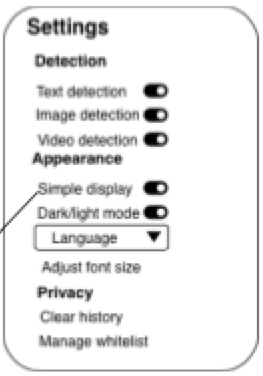
70% confident that this text is AI-generated
Because ...
The vocabulary choice and grammatical error ... reasoning for the result.
60% confident that this text is true/is a fact
List of supporting materials
Link to supporting material 1
Link to supporting material 2

50% confident that this text is AI-generated
Because ...
The vocabulary choice and grammatical error ... reasoning for the result.
80% confident that this text is true/is a fact
List of supporting materials
Link to supporting material 1
Link to supporting material 2

50% confident that this text is human written
Because ...
The vocabulary choice and grammatical error ... reasoning for the result.
80% confident that this text is false/fake news
List of supporting materials
Link to supporting material 1
Link to supporting material 2

70% confident that this text is human written
Because ...
The vocabulary choice and grammatical error ... reasoning for the result.
60% confident that this text is false/fake news
List of supporting materials
Link to supporting material 1
Link to supporting material 2

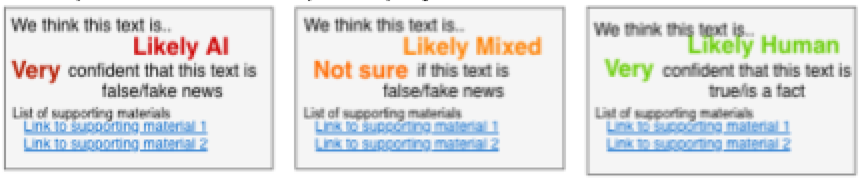
80% confident that this text is human written
Because ...
The vocabulary choice and grammatical error ... reasoning for the result.
40% confident that this text is false/fake news
List of supporting materials
Link to supporting material 1
Link to supporting material 2



Settings

Detection
Text detection
Image detection
Video detection
Appearance
Simple display
Dark/light mode
Language
Adjust font size
Privacy
Clear history
Manage whitelist

Example results with simple display turned on

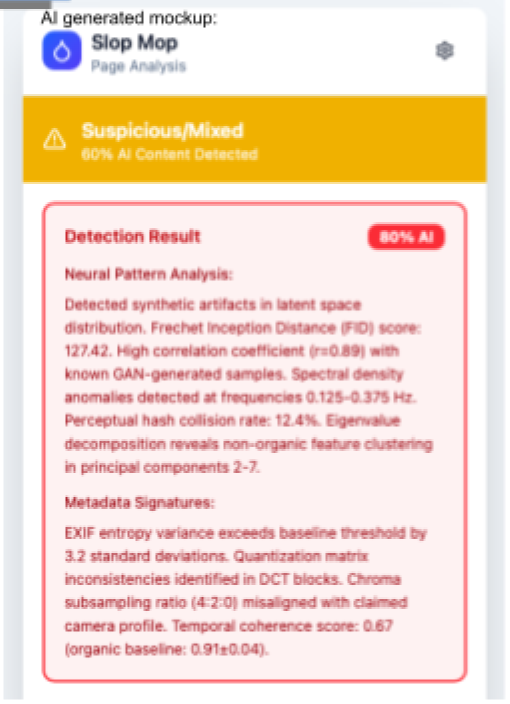


We think this text is.. **Likely AI**
Very confident that this text is false/fake news
List of supporting materials
Link to supporting material 1
Link to supporting material 2

We think this text is.. **Likely Mixed**
Not sure if this text is false/fake news
List of supporting materials
Link to supporting material 1
Link to supporting material 2

We think this text is.. **Likely Human**
Very confident that this text is true/is a fact
List of supporting materials
Link to supporting material 1
Link to supporting material 2

AI generated mockup:



Slop Mop
Page Analysis

Suspicious/Mixed
60% AI Content Detected

Detection Result 80% AI

Neural Pattern Analysis:

Detected synthetic artifacts in latent space distribution. Frechet Inception Distance (FID) score: 127.42. High correlation coefficient (r=0.89) with known GAN-generated samples. Spectral density anomalies detected at frequencies 0.125-0.375 Hz. Perceptual hash collision rate: 12.4%. Eigenvalue decomposition reveals non-organic feature clustering in principal components 2-7.

Metadata Signatures:

EXIF entropy variance exceeds baseline threshold by 3.2 standard deviations. Quantization matrix inconsistencies identified in DCT blocks. Chroma subsampling ratio (4:2:0) misaligned with claimed camera profile. Temporal coherence score: 0.67 (organic baseline: 0.91±0.04).