ChatGPT

# AI-Powered Mobile Disinformation Tracker (Google Cloud)

## Overview

We propose a system that detects fake vs. real news on a user's phone by analyzing the current screen content when the **Home** button is pressed. The mobile app (Android or iOS) captures the screen and sends the data to a Google Cloud backend. There, Vision AI and NLP services extract and analyze text/images, possibly cross-checking with fact-check databases. The user then gets a verdict or warning. Key Google Cloud components include Cloud Vision API (for OCR and image analysis), Cloud Natural Language API or Vertex AI (for text classification), Cloud Functions/Run (to orchestrate processing), and Firebase (for messaging/state). The architecture leverages pre-trained models and datasets (e.g. BERT-based fake-news classifiers, FactCheck API) to flag disinformation. Below we detail each component, trigger mechanism, available datasets, and restrictions.

## Mobile Platform Integration

- **Android:** We implement an Android app with an **AccessibilityService** that has permission to read screen content [1]. On Android 11+ the service can call `AccessibilityService.takeScreenshot()` to capture the display [2]. The app must request the user to enable this service in Settings. The service listens for UI events (e.g. window changes) and can detect when the Home button is pressed by checking for the system UI view ID (`com.android.systemui:id/home_button`) [3]. Upon detection, it immediately captures the screen bitmap. (See ScreenshotTile for an example: it uses an AccessibilityService to take screenshots on Android 11+ [4].)

- **iOS:** Apple does *not* allow arbitrary background screen capture or hooking into the Home button. There is **no public API** for capturing another app's screen content without user consent. Attempts to do so require private APIs (e.g. IOKit/IOSurface) and result in App Store rejection [5] [6]. Therefore, on iOS the app cannot auto-capture on Home press. A possible workaround is to use a **Share Extension** or manual flow: the user explicitly shares a screenshot or URL to our app, which then performs analysis. We note this severe platform restriction and focus on Android for automated triggering.

- **Trigger on Home Button:** On Android, the AccessibilityService can detect the Home button event (via the view ID hack [3]) or listen to `onAccessibilityEvent`. It then calls `takeScreenshot(Display.DEFAULT_DISPLAY, executor, callback)` (API 30+) to grab the screen [2]. This runs as a background service. On iOS, since this trigger isn't possible, one can only rely on user-initiated actions (not automated).

## Screen Capture and Content Extraction

- **Android Capture:** The AccessibilityService (running in foreground/background) invokes `takeScreenshot()`, yielding a `ScreenshotResult` bitmap [7]. This requires Android 11+ and the user's explicit permission to enable the accessibility service [4]. The screenshot is then forwarded (e.g. via HTTPS) to the cloud. Note that some apps set `FLAG_SECURE`, which prevents screenshots [8]; such screens will appear blank or will not be captured, a known limitation.

- **Image/Text Extraction (Cloud Vision):** Once the image is in the cloud (e.g. uploaded to a Cloud Storage bucket or sent in a Cloud Function request), we use Google Cloud Vision API. First, Vision's **OCR (TEXT_DETECTION or DOCUMENT_TEXT_DETECTION)** extracts textual content from the image [9]. This handles both short text (e.g. a headline) and dense text (e.g. full article on screen) [9] [10]. In parallel, Vision's **Web Detection** can find related web references and similar images for any graphics on screen [11]. For example, it returns "pages with matching images" or "visually similar images" which might indicate if the image (or meme) is known or manipulated [12] [13].

- **Natural Language Processing:** The extracted text is then analyzed. We can use Cloud Natural Language API or a custom model via Vertex AI. The NLP pipeline may include:

- *Named Entity Recognition* (identify people, organizations, locations),
- *Sentiment or content analysis* (to gauge sensationalism), and
- *Claim detection/classification* (evaluating factual vs. misleading statements).
  While Google's NL API does not natively classify "fake news," we can use it to structure the text and possibly train an AutoML model. Alternatively, a pre-trained language model (e.g. BERT) can be deployed on Vertex AI to classify the text.

- **Fact-Check Integration:** We can optionally query Google's Fact Check Tools API. The **Claim Search API** lets us search for fact-checks related to a query (e.g. a key phrase from the text) [14]. This retrieves known debunks or fact-check articles. Combining our prediction with external fact-check results can improve accuracy (flag known misinformation).

## Disinformation Models and Datasets

Existing datasets and models can bootstrap the system:

- **Datasets:** We would train on labeled fake-news corpora such as *WELFake* (72k articles, roughly half fake/real) [15], *FakeNewsNet* (with Politifact and GossipCop real/fake pairs) [16] [17], and *LIAR* (12.8K PolitiFact statements labeled by truthfulness) [18]. Kaggle also hosts Politifact and ISOT fake news datasets. These provide text (headlines and articles) plus a label.

- **Pretrained Models:** Transformer models fine-tuned on fake-news data can be used. For example, a BERT-based model from Hugging Face is explicitly trained to classify news as real/fake [19]. Such models (e.g. `bert-base-uncased` fine-tuned on news datasets) often achieve high accuracy [20]. We could deploy a Hugging Face model via Vertex AI or call the HF Inference API. Several open-source projects exist (e.g. a GitHub repo using Scikit-learn on labelled news [21] [22]). By leveraging these resources, the system's core classifier can run either in a Cloud Function/Run or even on-device via ML Kit if needed.

## Security, Privacy, and OS Restrictions

- **User Consent & Privacy:** Reading the screen is highly sensitive. Our app must clearly request permission (accessibility or share access) and explain why. Any captured image/text should be processed securely: ideally encrypted in transit and discarded after use. Store minimal data on Google Cloud (e.g. no personal PII). The user should have control (e.g. opt-out or disable).

- **Android Policy:** Google Play's policies strictly restrict AccessibilityService use. An app may only use it for *accessibility functions*, not for unrelated tasks. Our justification would be that we assist users by identifying harmful content. Nonetheless, misuse could lead to rejection. Also, many apps (banking, video players) set `FLAG_SECURE` to block screenshots [8], so our system will simply skip or get blank data for those screens.

- **iOS Policy:** As noted, background capture is not permitted. The workaround (user-initiated share) complies with iOS sandbox rules. There's no violation of privacy if the user explicitly triggers analysis.

- **Performance/Security:** Since the system captures potentially sensitive screen data, it should do so *only on user action* (Home press) and *not continuously*, minimizing privacy exposure. All cloud communication should use HTTPS and authentication (e.g. Firebase Auth) so that only authorized devices send data. We must follow GDPR-like practices if personal data is involved.

## System Architecture and Flow

1. **Mobile Trigger:** The user presses Home. On Android, our AccessibilityService intercepts this and takes a screenshot [3] [4]. On iOS, this auto-trigger is not possible (user must manually share).

2. **Data Upload:** The screenshot (image) is sent from the app to Google Cloud. This could be a Cloud Function HTTP endpoint or upload to a Cloud Storage bucket.

3. **OCR Extraction:** A Cloud Function (or Cloud Run container) calls Cloud Vision's OCR (`DOCUMENT_TEXT_DETECTION`) to extract text [9]. It also optionally calls Vision's Web Detection to find related images/pages [11].

4. **Text Analysis:** The extracted text is fed into NLP. A Vertex AI endpoint (e.g. BERT classifier) predicts "fake vs. real" or "likely false" score. We might also call Cloud Natural Language API to extract entities/sentiment as features.

5. **Fact-Check Lookup:** The text or its claims are queried via Google's Fact Check Claim Search API [14]. If matches are found, they influence the final verdict (e.g. "Claim already debunked as false").

6. **Result Aggregation:** The outputs (predicted probability of fake news, confidence, relevant fact-check excerpts) are combined into a response. This response could be stored in Firestore or sent immediately via Firebase Cloud Messaging (push notification) back to the device.

7. **User Notification:** On the phone, the app receives the analysis result and alerts the user (e.g. a notification or overlay pop-up) indicating whether the content appears fake or real, possibly linking to fact-check sources.

8. **Feedback Loop:** Optionally, the user can correct the result (flag misclassification) to improve the model over time. Cloud Firestore can log examples for retraining.

```
Mobile (Android App with AccessibilityService)
       │
       ├─[Home press]→ AccessibilityService.takeScreenshot() [2]
       └─[HTTPS/Storage Upload]→ Cloud Function (Vision OCR & Analysis)
                 ↓
        ┌─────────┴─────────┐
        │      Vision OCR     │→ text output [9]
        │    Web Detection    │→ image context [11]
        └─────────┬─────────┘
                    ↓
             Cloud NLP/ML (e.g. Vertex)
                    ↓
        Google FactCheck API lookup [14]
                    ↓
              Verdict (fake/real)
                    ↓
  Store/Return result → Firebase Push/DB → Mobile UI
```

**Cloud Components:**
- *Cloud Functions or Cloud Run* containers implement the above pipeline. They use Google-managed AI APIs (Vision, Language) and possibly call custom Vertex AI endpoints.
- *Cloud Storage* may temporarily hold screenshots (triggering Cloud Functions via Pub/Sub notifications).
- *Firebase Auth* can secure the endpoints, ensuring only our app sends data.
- *Firestore* can log scan history, models' output, and user feedback.
- *Cloud Monitoring* tracks usage and latency.

**Example:** AI Seer's Facticity.AI uses a similar approach: it runs inference containers on Cloud Run and Vertex AI (even using Google's Gemini LLM) to fact-check statements [23] [24]. They achieved <20 ms latency per request by scaling with Cloud Run and multi-LLM models [23] [24]. We can adopt a lighter version of this: a Cloud Function or AutoML model for fast classification.

## Key References and Tools

- **Accessibility Service:** Android's `AccessibilityService` can read screen text and capture screenshots [2] [1], but use it only with user consent.
- **Vision API:** For OCR and image context: "Vision API can detect and extract text from images" [9], and its Web Detection finds related web matches [11].

- **Fact Check API:** Google's Fact Check Tools (Claim Search API) lets us query existing fact-checks by text query [14].
- **Models/Datasets:** Use BERT-based fake-news classifiers [19], trained on datasets like WELFake (72k labeled articles) [15] or FakeNewsNet (PolitiFact/GossipCop) [16] [17] and LIAR (PolitiFact statements) [18].
- **Example Code:** Many open-source projects (e.g. a GitHub fake-news detector) illustrate training classifiers on news text [21] [22]. The ScreenshotTile app demonstrates the takeScreenshot API on Android [4].

By integrating these components, the proposed system continuously monitors the screen (Android) or accepts shared content (iOS), analyzes it via Google Cloud's AI services, and warns the user when disinformation is detected. The design respects platform privacy rules and uses scalable cloud ML for robust fake-news detection.

**Sources:** Authoritative documentation and examples are cited above for all components [1] [4] [9] [11] [14] [20] [18] [16] [15] [23] [21] [8] [6].

---

[1] Impact of Accessibility Permission in Android Apps | BrowserStack
https://www.browserstack.com/guide/accessibility-permission-in-android

[2] [4] GitHub - cvzi/ScreenshotTile: Screenshot Tile for Android without Root
https://github.com/cvzi/ScreenshotTile

[3] accessibilityservice - (Android) How to detect the press of the virtual back button, home button and overview button - Stack Overflow
https://stackoverflow.com/questions/73347131/android-how-to-detect-the-press-of-the-virtual-back-button-home-button-and-ov

[5] [6] iphone - How does the iOS app Display Recorder record the screen without using private API? - Stack Overflow
https://stackoverflow.com/questions/11090184/how-does-the-ios-app-display-recorder-record-the-screen-without-using-private-ap

[7] AccessibilityService.TakeScreenshot Method (Android.AccessibilityServices) | Microsoft Learn
https://learn.microsoft.com/en-us/dotnet/api/android.accessibilityservices.accessibilityservice.takescreenshot?view=net-android-35.0

[8] Secure sensitive activities | Fraud prevention | Android Developers
https://developer.android.com/security/fraud-prevention/activities

[9] [10] Detect text in images | Cloud Vision API | Google Cloud
https://cloud.google.com/vision/docs/ocr

[11] [12] [13] Detect Web entities and pages | Cloud Vision API | Google Cloud
https://cloud.google.com/vision/docs/detecting-web

[14] Fact Check Tools API | Google for Developers
https://developers.google.com/fact-check/tools/api

[15] Dataset Search
https://datasetsearch.research.google.com/search?query=Fake+Content+Detection

[16] [17] GitHub - KaiDMML/FakeNewsNet: This is a dataset for fake news detection research
https://github.com/KaiDMML/FakeNewsNet

[18] ucsbnlp/liar · Datasets at Hugging Face
https://huggingface.co/datasets/ucsbnlp/liar

[19] [20] Pulk17/Fake-News-Detection · Hugging Face
https://huggingface.co/Pulk17/Fake-News-Detection

[21] [22] GitHub - kapilsinghnegi/Fake-News-Detection: This project detects whether a news is fake or not using machine learning.
https://github.com/kapilsinghnegi/Fake-News-Detection

[23] [24] AI Seer case study | Google Cloud
https://cloud.google.com/customers/ai-seer