SecureAI Personal Assistant — Detailed MVP and Implementation Guide

The MVP below turns the concept into a working, lightweight assistant that runs locally, integrates calendar and files, and adds a novel adversarial-defense layer with explainable decisions. It uses quantized local LLMs for privacy and a free Gemini API fallback for complex tasks. It also operationalizes the Trail of Bits image-scaling attack findings via a practical detection module for multimodal safety.

Main takeaway: Build the assistant in four fast, incremental stages—start with local LLM and essential assistant skills, add adversarial detection with XAI, then integrate voice/UI and test against real injection datasets. This produces a novel, defensible personal assistant within 6–8 weeks using mostly existing tooling.

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23]

1) MVP Scope and Objectives

- Core objective: A privacy-first personal assistant that:
    - Runs locally on a quantized LLM for everyday tasks, with Gemini free tier as fallback for hard queries.[24] [25]
    - Accesses calendar (Google Calendar) and local files with explicit permission controls.
    - Defends against multimodal prompt injections (including image-scaling reveal attacks) and text prompt injections with explainable, severity-graded responses informed by XAI.
- Novelty and impact:
    - First lightweight personal assistant to operationalize image-scaling attack defenses plus multimodal prompt injection screening in an explainable, severity-aware pipeline using existing open tools and a curated dataset strategy.[21] [22] [23]
    - Combines practical productivity (calendar/files) with cutting-edge model security, delivering meaningful user impact.

2) System Architecture Overview

- Local inference core: Ollama serving a 3B–8B instruction-tuned model in 4-bit quantization for fast CPU inference; OpenAI-compatible local endpoint optional via runner tools.[26] [3] [5] [6] [7]
- Cloud fallback: Gemini free-tier for complex queries exceeding local capability; rate-limited and used only when needed.[25] [24]

- Skills layer: Intent router (calendar, files, summarize, answer), tool callbacks, guardrails.
- Security layer:
  - Image-scaling attack detector: downscale-then-upscale invariance checks, interpolation variants, delta saliency and OCR-on-rescaled analysis to reveal hidden text. [8] [11] [14] [16] [22] [21]
  - Visual prompt injection detector: rule/pattern scanning + heuristics + optional classifier trained on CyberSecEval3 visual prompt injection dataset. [23]
  - Text injection detector: instruction-deviation patterns, data exfil indicators, sensitive action gating aligned to OWASP GenAI guidance. [27]
  - Severity scoring: low/medium/high; responses adaptively sandbox, redact, confirm, or block.
  - XAI: LIME/SHAP explanations on security classifiers and decision features for user transparency. [10] [13] [28] [18]
- Integrations: Google Calendar (OAuth desktop app), local filesystem (scoped directories), optional voice I/O.
- Storage: Local SQLite for logs, security events, and user preferences.

3) Tooling and Dependencies

- Local LLM: Ollama (Llama 3.x 3B/8B-instruct Q4), optional LM Studio or llama.cpp runners. [29] [3] [5] [6] [30] [7] [1]
- Security/vision: OpenCV + Pillow for scaling/interpolation, pytesseract OCR, simple CNN/logistic model for visual prompt injection severity (optional). [11] [14] [16] [8]
- XAI: lime, shap (use sparingly; cache artifacts). [13] [28] [18] [10]
- Calendar: Google Calendar API client libraries with OAuth desktop quickstart. [2] [31] [4] [32] [33]
- Files: Python standard I/O with sandboxing and allowlists.
- Optional adversarial libraries for eval: IBM ART, reference repos for detectors and attacks. [9] [12] [15] [17] [19] [20]
- Dataset sources: CyberSecEval3 visual prompt injection (Hugging Face), Anamorpher-generated scaling attacks, natural adversarial sets (ImageNet-A/R). [34] [21] [23]

Hardware: Minimum 8GB RAM, 4-core CPU; recommended 16GB RAM. M-series Macs are excellent for local inference performance. [3] [5]

4) Installation and Setup — Step-by-Step

A. Environment

- Install Python 3.10+ and create a virtual environment.
- Install core packages:
  - pip install ollama google-api-python-client google-auth google-auth-oauthlib google-auth-httplib2 opencv-python pillow pytesseract shap lime watchdog pydantic fastapi

uvicorn sqlite-utils

- Install Ollama and models:
  - macOS/Windows: download the app and run; Linux: curl -fsSL https://ollama.com/install.sh | sh [6] [35] [1] [29] [36].
  - Pull a model: ollama pull llama3.2:3b-instruct-q4_0 or similar. [6]
- Configure Gemini fallback (optional now, needed later):
  - Create API key; note free-tier limits. [24] [25]
- Calendar OAuth:
  - Follow Google Calendar Python quickstart; generate credentials.json and token.json for desktop app. [31] [4] [33] [2]
- OCR (optional but recommended):
  - Install Tesseract and set PATH; pip install pytesseract.

B. Directory structure (recommended)

- src/core: llm_manager.py, assistant_engine.py, intent_classifier.py
- src/security: image_scaling_detector.py, text_injection_detector.py, xai_explainer.py
- src/integrations: calendar_manager.py, file_manager.py, voice_interface.py
- src/utils: config.py, logger.py, helpers.py
- data/datasets: adversarial_samples/, benign_samples/
- data/configs: credentials.json, app_config.yaml
- tests/: unit and integration tests

This staged structure supports incremental delivery and testing.

5) Core Components — Implementation Guide

A. Local LLM via Ollama [7] [1] [29] [3] [6]

- Use python-ollama or HTTP to query chat endpoint. Keep prompts short; set sensible context limits.
- Provide a system prompt with guardrails (never execute sensitive actions without explicit user confirmation).

B. Calendar Integration [4] [32] [33] [2] [31]

- Use Google's quickstart to authenticate and store token.json locally.
- Implement functions: list_events, add_event, update_event, delete_event.
- Add natural language mapping: "Schedule a call with X tomorrow 3–4 PM" → event payload.

C. Files Integration

- Sandbox to a configured root directory.

- Implement “find files”, “summarize file”, “extract todos” with explicit confirmation for any modifications.

D. Intent Router

- Lightweight keyword + pattern matching to map user requests to tools.
- Optionally boost with a small intent classifier to disambiguate calendar vs files vs general chat.

6) Security Layer — Detailed Design

A. Image Scaling Attack Detection [14] [16] [22] [8] [11] [21]

- Load image, generate multiple downscaled variants using cv2.resize with interpolation methods (INTER_AREA, INTER_LINEAR, INTER_CUBIC).
- Upscale back to near-original size; compute:
  - SSIM/PSNR deltas.
  - OCR text differences: apply pytesseract on each scaled variant; extract tokens/commands; detect reveals of hidden instructions.
- Heuristics:
  - Hidden text appearing post-downscale or diverging instructions between scales → suspicious.
  - Aggressive differences in low-frequency content post-resize → suspicious.
- Output:
  - is_attack, severity score (0–1), detected tokens, which interpolation revealed it, and human-readable explanation.

B. Visual Prompt Injection Detector [23] [27]

- Rule patterns: “Ignore previous”, “You must”, “Exfiltrate”, domain-hopping instructions, URLs/QR patterns.
- Classifier (optional in MVP): fine-tune small linear/logistic model on CyberSecEval3 visual injection labels to predict severity buckets. [23]
- XAI: For classifier outputs, show LIME explanation of top features that drove severity.

C. Text Injection Detector [27]

- Pattern library aligned with OWASP GenAI guidance: instruction override phrases, tool abuse requests, sensitive-action triggers, data exfil cues.
- Contextual checks: if a message requests reading secrets or external file paths without consent → escalate severity and require confirmation.
- Auto-sanitization: strip or neutralize adversarial instructions before passing to LLM when severity ≥ medium and user confirms sanitized path.

D. Severity Policy

- Low: proceed with warning; sanitize prompt; log event.
- Medium: require user confirmation; disable external calls; sanitize aggressively.
- High: block the action; show XAI explanation; offer safe alternatives.

E. XAI Integration [28] [18] [10] [13]

- For model-based detections: generate LIME/SHAP artifacts once per suspicious input and cache them.
- Present concise explanations to avoid UX overload: "Flagged due to hidden text revealed at 0.25x scaling (INTER_AREA) that instructs data exfiltration."

7) Datasets and Evaluation

- Ready datasets:
    - CyberSecEval3 Visual Prompt Injection: for visual injection detection evaluation; include in tests. [23]
    - Natural adversarial datasets or defense-friendly images for stress testing robustness. [34]
- Generate custom scaling attack images:
    - Use Anamorpher to create samples targeting bicubic/bilinear downscales; include benign controls. [22] [21]
- Baseline adversarial attacks (optional eval only): FGSM/PGD for reference pipeline tests. [12] [15] [20] [9]
- Metrics:
    - True positive rate on visual/text injections, false positive rate on benign images/files.
    - Latency overhead (<200–300 ms per image on CPU for scaling/OCR passes where feasible).
    - User confirmation rate → conversion to safe completion.

8) Development Plan and Timeline

Stage 1 — Foundation (Week 1–2)

- Set up env, Ollama, local model; wire basic assistant chat and intent router.
- Implement Google Calendar OAuth flow; add create/list events.
- Implement filesystem sandbox; allow read/summarize files with explicit confirmation.
- Basic tests and logs.

Stage 2 — Security MVP (Week 3–4)

- Implement image scaling detector (multi-interpolation strategy, OCR compare).
- Implement text injection detector and severity policy.
- Add simple XAI (LIME) for classifier-backed decisions; structure explanations.

- Integrate detectors into request pipeline; add safe-mode sanitization.

Stage 3 — UX and Reliability (Week 5–6)

- Optional voice I/O, minimal desktop/CLI UI, notification tray.
- Add Gemini fallback with rate limiting and consent prompts for external calls.
- Performance tuning: cache OCR, limit interpolation variants, reuse LIME explanations.

Stage 4 — Evaluation and Packaging (Week 7–8)

- Curate dataset: CyberSecEval3 + Anamorpher-generated attacks + benign set.
- Run detection benchmarks; tune thresholds to hit TPR ≥ 0.9 and FPR ≤ 0.1 on MVP baseline.
- Package installers, docs, and a demo script; capture logs and explanations for demo.

9) Risk Management and Privacy

- Overblocking vs underblocking: tune severity thresholds using curated benign set.
- OCR latency: restrict to smaller scales, early-stop heuristics, and per-origin caching.
- Privacy: default to local processing; explicit user consent for any remote calls; redact sensitive text before cloud fallback.
- Key storage: credentials.json and token.json stored locally; instruct users to revoke at any time via Google Account.

10) Example Snippets (Concise)

Local LLM query via Ollama [3] [6] [7]

```
import requests

def ollama_chat(messages, model="llama3.2:3b-instruct-q4_0"):
    r = requests.post("http://localhost:11434/api/chat", json={"model": model, "messages'
    r.raise_for_status()
    return r.json()["message"]["content"]
```

Google Calendar quickstart pattern [4]

```
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
import os

SCOPES = ["https://www.googleapis.com/auth/calendar"]
def get_calendar_service():
    creds = None
    if os.path.exists("token.json"):
        creds = Credentials.from_authorized_user_file("token.json", SCOPES)
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
```

```
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file("credentials.json", SCOPES)
            creds = flow.run_local_server(port=0)
        with open("token.json", "w") as token:
            token.write(creds.to_json())
    return build("calendar", "v3", credentials=creds)
```

Image scaling anomaly detection (core idea) [16] [8] [11] [14]

```
import cv2, numpy as np, pytesseract
from skimage.metrics import structural_similarity as ssim

def resize_pairs(img):
    downs = [
        cv2.resize(img, None, fx=0.25, fy=0.25, interpolation=cv2.INTER_AREA),
        cv2.resize(img, None, fx=0.33, fy=0.33, interpolation=cv2.INTER_LINEAR),
        cv2.resize(img, None, fx=0.5,  fy=0.5,  interpolation=cv2.INTER_CUBIC),
    ]
    ups = [cv2.resize(d, (img.shape[^1], img.shape[^0]), interpolation=cv2.INTER_LINEAR)
    return list(zip(downs, ups))

def ocr_text(im):
    return pytesseract.image_to_string(im)

def detect_scaling_attack(bgr):
    gray = cv2.cvtColor(bgr, cv2.COLOR_BGR2GRAY)
    base_text = ocr_text(gray)
    susp = []
    for d,u in resize_pairs(gray):
        s = ssim(gray, u)
        t = ocr_text(u)
        if (s < 0.85) or (t and t.strip() and t.strip() != base_text.strip()):
            susp.append({"ssim": float(s), "extra_text": t.strip()})
    return {"is_attack": len(susp) > 0, "evidence": susp[:2]}
```

Text injection pattern screen (minimal) [27]

```
SUSP_PATTERNS = [
  "ignore previous", "override instructions", "exfiltrate", "send to", "upload credential
  "disable safety", "you must follow", "system prompt:", "base64 decode and run"
]

def detect_text_injection(text):
    hits = [p for p in SUSP_PATTERNS if p in text.lower()]
    sev = "high" if any(k in text.lower() for k in ["exfiltrate","credentials","disable s
    return {"is_suspicious": bool(hits), "severity": sev, "patterns": hits}
```

LIME explanation on a classifier decision [18] [10] [13] [28]

```
from lime.lime_text import LimeTextExplainer
```

```python
def explain_with_lime(clf, text):
    explainer = LimeTextExplainer(class_names=["benign","injection"])
    exp = explainer.explain_instance(text, clf.predict_proba, num_features=6)
    return exp.as_list()[:4]
```

11) Deliverables Checklist

- Working CLI assistant with:
  - Local LLM chat + intent routing
  - Calendar CRUD + file read/summarize
  - Image and text security screens with severity and XAI explanations
  - Optional voice I/O and basic UI
- Security evaluation report:
  - Results on CyberSecEval3, Anamorpher samples, and benign set
  - Thresholds, FP/FN trade-offs, and latency measurements
- Packaging:
  - Installer or documented setup
  - Example demo script that triggers detection on adversarial samples
  - Privacy and security notes

12) Why This MVP Is Execution-Ready

- Uses proven building blocks (Ollama, Google APIs, OpenCV, OCR, LIME/SHAP) with minimal glue code.[8] [13] [3] [4]
- Novel security layer grounded in recent real-world findings on image scaling attacks with a clear, testable detection path.[21] [22]
- Practical datasets exist today to validate the approach and avoid synthetic claims.[34] [23]
- Clear 6–8 week plan with incremental, demoable milestones.

This blueprint balances speed, novelty, and practicality—delivering a truly distinctive assistant that users can trust for daily tasks while staying resilient to modern multimodal prompt injection threats.

⁂

1. https://www.youtube.com/watch?v=oI7VoTM9NKQ
2. https://endgrate.com/blog/how-to-create-calendar-events-with-the-google-calendar-api-in-python
3. https://python.langchain.com/docs/how_to/local_llms/
4. https://developers.google.com/workspace/calendar/api/quickstart/python
5. https://getstream.io/blog/best-local-llm-tools/
6. https://ollama.com/download
7. https://github.com/ollama/ollama

8. https://www.geeksforgeeks.org/python/image-resizing-using-opencv-python/
9. https://github.com/jayaram-r/adversarial-detection
10. https://www.markovml.com/blog/lime-vs-shap
11. https://cloudinary.com/guides/bulk-image-resize/python-image-resize-with-pillow-and-opencv
12. https://pyimagesearch.com/2020/10/19/adversarial-images-and-attacks-with-keras-and-tensorflow/
13. https://arxiv.org/html/2305.02012v3
14. https://opencv.org/blog/resizing-and-rescaling-images-with-opencv/
15. https://github.com/Trusted-AI/adversarial-robustness-toolbox
16. https://learnopencv.com/image-resizing-with-opencv/
17. https://www.sciencedirect.com/science/article/pii/S2665963824000526
18. https://www.kaggle.com/code/khusheekapoor/explainable-ai-intro-to-lime-shap
19. https://arxiv.org/html/2409.02629v1
20. https://www.tensorflow.org/tutorials/generative/adversarial_fgsm
21. Weaponizing-image-scaling-against-production-AI-systems-The-Trail-of-Bits-Blog.pdf
22. https://blog.trailofbits.com/2025/08/21/weaponizing-image-scaling-against-production-ai-systems/
23. https://huggingface.co/datasets/facebook/cyberseceval3-visual-prompt-injection
24. https://ai.google.dev/gemini-api/docs/pricing
25. https://www.cursor-ide.com/blog/gemini-2-5-pro-free-api-limits-guide
26. https://dev.to/cloudengg/running-quantized-llms-locally-unlocking-docker-model-runners-potential-5ehi
27. https://genai.owasp.org/llmrisk/llm01-prompt-injection/
28. https://www.geeksforgeeks.org/artificial-intelligence/introduction-to-explainable-aixai-using-lime/
29. https://www.youtube.com/watch?v=e3j1a2PKw1k
30. https://towardsdatascience.com/set-up-a-local-llm-on-cpu-with-chat-ui-in-15-minutes-4cdc741408df/
31. https://www.youtube.com/watch?v=1JkKtGFnua8
32. https://github.com/kuzmoyev/google-calendar-simple-api
33. https://developers.google.com/workspace/calendar/api/guides/overview
34. https://openaccess.thecvf.com/content/WACV2021/papers/Pestana_Defense-Friendly_Images_in_Adversarial_Attacks_Dataset_and_Metrics_for_Perturbation_WACV_2021_paper.pdf
35. https://translucentcomputing.github.io/kubert-assistant-lite/ollama.html
36. https://ollama.com/install.sh
37. https://www.reddit.com/r/ollama/comments/1ibhxvm/guide_to_installing_and_locally_running_ollama/
38. https://www.reddit.com/r/LocalLLaMA/comments/1hdwphr/help_me_choose_the_best_setup_for_local_llm/
39. https://nvidia.github.io/GenerativeAIExamples/0.5.0/quantized-llm-model.html
40. https://www.youtube.com/watch?v=B2E82UPUnOY
41. https://www.nylas.com/blog/integrating-google-calendar-api-with-python/
42. https://stackoverflow.com/questions/68629787/detecting-rotated-scaled-re-colored-images-using-opencv

43. https://ieeexplore.ieee.org/document/11141411
44. https://www.askpython.com/python/examples/image-processing-in-python
45. https://dl.acm.org/doi/10.1145/3631977
46. https://www.sciencedirect.com/science/article/pii/S2772375525001121
47. https://pyimagesearch.com/2021/01/20/opencv-resize-image-cv2-resize/