Virtual Assistant

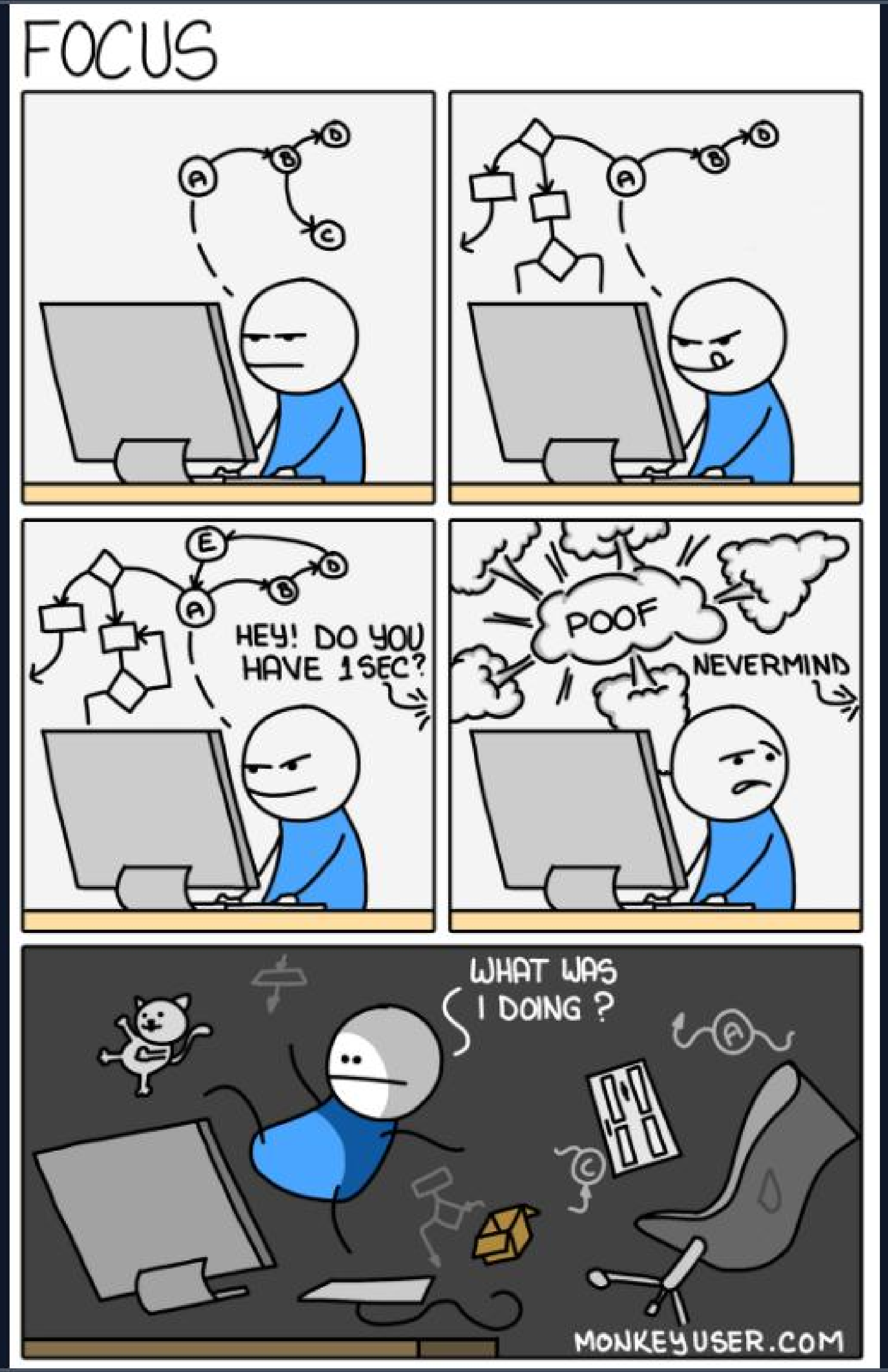
FOR DEVELOPER DEPENDENCY & CODE VULNERABILITY

PROJECT PROPOSAL

The Challenge

Modern software development relies heavily on open-source packages, leading to complex dependency trees and hidden security risks.

- **Dependency Hell:** Conflicting versions and transitive dependencies.
- **Hidden Vulnerabilities:** CVEs buried deep in the dependency graph.
- **Manual Overhead:** Hours spent cross-referencing advisory databases.



Use Case Overview

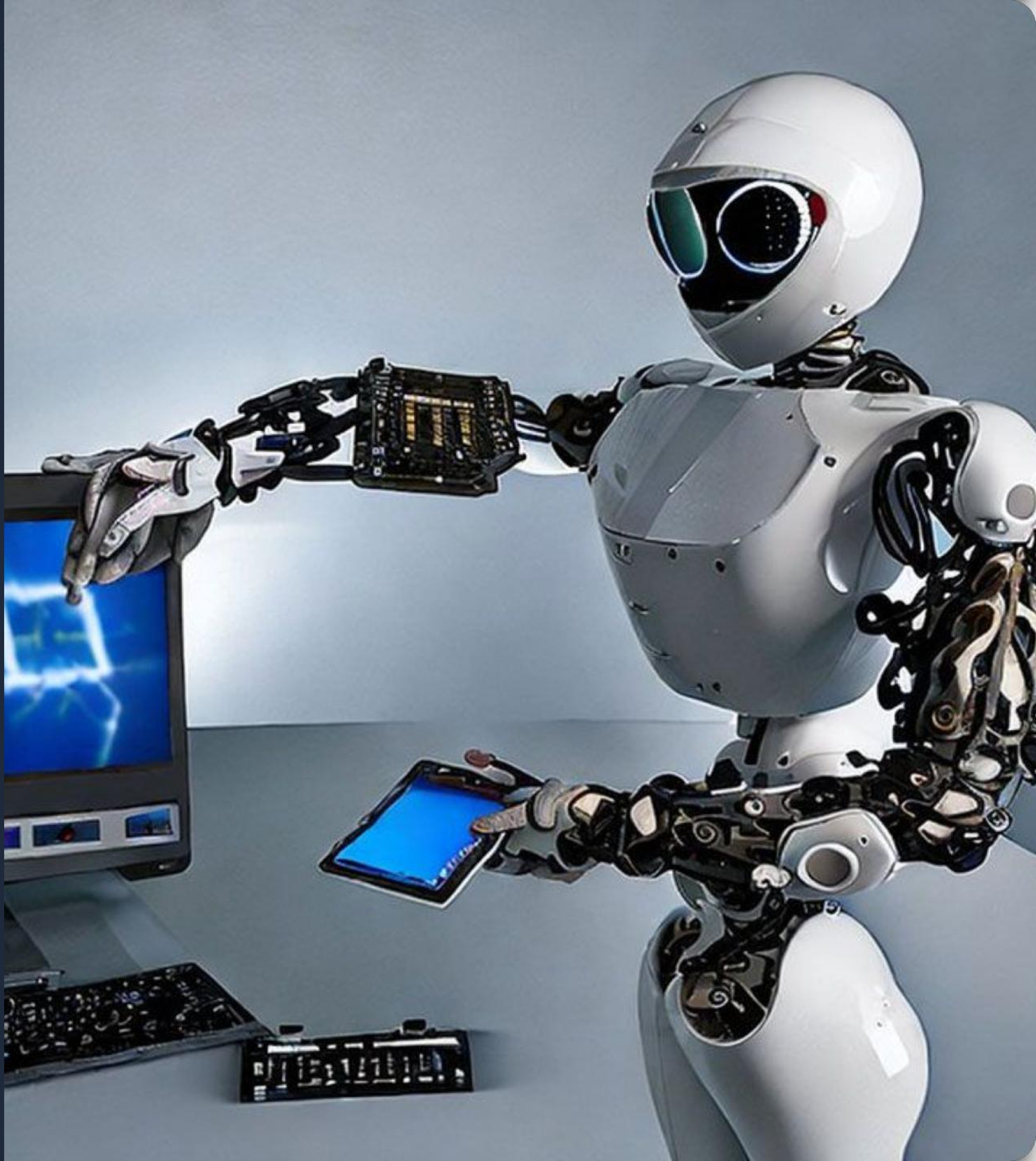
A specialized Virtual Assistant designed to help developers secure their Python projects effortlessly.

Core Functionality:

Scans dependency manifests (e.g., `requirements.txt`, `poetry.lock`) to identify known vulnerabilities and recommend safe versions.

Key Output:

Provides actionable insights, safe pins, and remediation commands via natural language chat.



System Architecture

1. User Query

Developer asks natural language questions about security.

2. LLM Reasoning

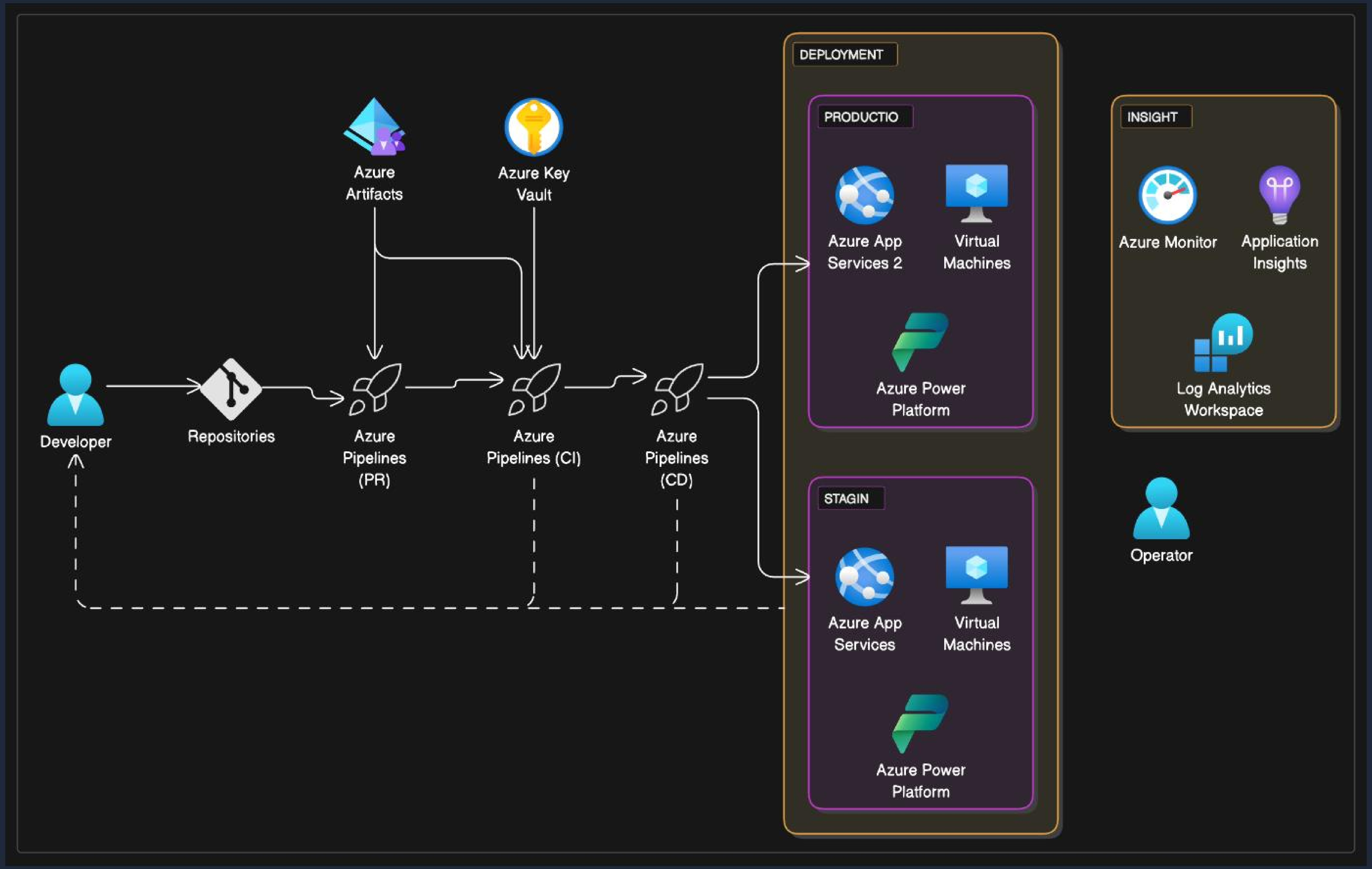
Model interprets intent and selects appropriate tools.

3. Tool Execution

Fetches real-time data from OSV, PyPI, and Scanners.

4. Response

Synthesizes data into actionable advice.



Data Sources & Knowledge Base

OSV API

A distributed vulnerability database for Open Source Projects. The primary source for CVEs and advisories.

PyPI JSON

Provides critical metadata: releases, `requires_python`, distribution URLs, and project links.

Standards

Enforces deterministic version handling via **PEP 440** and dependency specs via **PEP 508**.

Tools & Comparators

Execution Tools

The assistant leverages command-line tools to perform the heavy lifting:

- **pip-audit:** Audits Python environments for packages with known vulnerabilities.
- **OSV Scanner:** General purpose vulnerability scanner using the OSV database.

LLM Backend

Selected models for reasoning and tool orchestration:

- **Phi-3.5-mini-instruct:** Lightweight, efficient.
- **Llama-2-13B-GGUF:** Robust reasoning capabilities.

Sample User Queries: Basic

- **Vulnerability Scan**

  > "Scan my requirements.txt for vulns; propose minimal safe pins for Python 3.11."

- **Specific Package Check**

  > "Is urllib3 == 1.25.8 vulnerable? Show CVEs, CVSS, and first fixed version."

- **Advisory Lookup**

  > "For Django, list all advisories affecting my version, earliest patched version, and links."

Sample User Queries: Advanced

- **Conflict Resolution**

  > "Urllib3 fix conflicts with requests<2.31. Explain and give two alternatives."

- **Policy Enforcement**

  > "Apply policy: no pre-releases, min CVSS 5.0. What pins do you recommend?"

- **Report Generation**

  > "Export the scan as JSON and a Markdown summary."

- **CI/CD Integration**

  > "Give me a GitHub Actions step to run pip-audit on every push."

Why This Architecture?

Privacy First

Utilization of local-capable models (GGUF format) ensures sensitive dependency trees are processed securely.

Deterministic Accuracy

LLMs leverage **Tools** for facts. We rely on OSV/PyPI for the "truth" rather than LLM training data.

Real-Time Data

Direct API calls mean the assistant knows about vulnerabilities discovered **today**, not just in the training set.

Questions?

Open Floor for Discussion

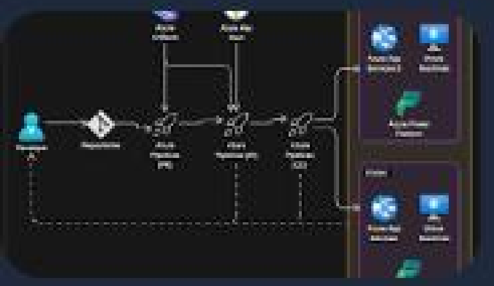
Image Sources

https://preview.redd.it/o1gnotfxi4k01.png?width=1080&crop=smart&auto=webp&s=b478b5bc496fd38c0e423c0db69869bbc96d120f

Source: www.reddit.com

https://cdn.britannica.com/47/246247-050-F1021DE9/AI-text-to-image-photo-robot-with-computer.jpg

Source: www.britannica.com



https://cdn.prod.website-files.com/62d58a323cbc396f06a780aa/687ef4976555b76d51d47f7b_68379b49e4b72a73c2992bd2_diagram-export-5-28-2025-4_24_52-PM.png

Source: www.eraser.io

https://www.exascaleproject.org/wp-content/uploads/2019/11/Facility-Resource-Utilization-scaled.jpg

Source: www.exascaleproject.org