Project Description – System Telemetry Monitor (Python + Tkinter Desktop App)

□ Overview

This project is a lightweight **System Telemetry Monitoring Application** built using **Python** and displayed through a **Tkinter GUI desktop interface**. The application shows **real-time system health metrics**, including:

- **CPU Usage (%)**
- **Memory Usage (%)**
- **Number of Running Processes**
- **Count of Open Network Ports**
- **Auto-refreshing live telemetry every few seconds**

The project is bundled into an **EXE desktop application**, allowing it to run without Python installed.

□ What Makes This Project Unique?

□ **The entire project (logic, UI, structure) was created using ChatGPT.**
□ Small modifications, UI adjustments, and packaging into EXE were done in **VS Code**.
□ Instead of showing data in a browser, the project uses a **Tkinter desktop window** to show live system telemetry.

This demonstrates how AI + developer refinement can create a fully functioning desktop tool.

□□ How the Project Works – Step-by-Step

□ Step 1 – Telemetry Data Collection (telemetry.py)

A Python script gathers system statistics using:

✔ **`psutil`**

Used for reading:

- CPU usage (`psutil.cpu_percent()`)
- Memory usage (`psutil.virtual_memory().percent`)
- Running processes (`len(psutil.pids())`)

✔ Open Ports Detection

A system command is executed:

```
netstat -ano | findstr LISTENING
```

The output is parsed and **only the count of ports** is extracted (not the full list), giving you:

```
Number of open network ports
```

All values are returned as a dictionary:

```
{
  "cpu": 8.5,
  "memory": 64.3,
  "processes": 297,
  "ports": 30
}
```

□ Step 2 – Tkinter Desktop Display (app.py)

Instead of using a browser, the project uses a **Tkinter GUI window** to display telemetry in real time.

The GUI shows:

- CPU Usage: XX%
- Memory Usage: XX%
- Running Processes: XX
- Open Network Ports: XX

Features:

- Clean, simple UI
- Auto-refresh (e.g., every 2 seconds)
- Large readable labels
- Optionally color-coded (green/yellow/red) based on usage
- Runs as a standalone `.exe`

□ Step 3 – Auto-Refreshing Live Data

The Tkinter window schedules updates using:

```
root.after(2000, refreshData)
```

This means every 2 seconds the app:

1. Calls `getSystemSnapshot()`
2. Updates all labels
3. Displays fresh results in the GUI

This makes the app behave like a **live system dashboard**.

□ Step 4 – Packaging as a Desktop EXE

The final application is compiled into an EXE using:

```
pyinstaller --onefile --noconsole app.py
```

This lets you run the app on any Windows machine without installing Python.

□ Development Process (Important Note)

This entire project — from system design to telemetry logic, Tkinter UI, live refreshing, and EXE instructions — was:

□ Fully generated using ChatGPT

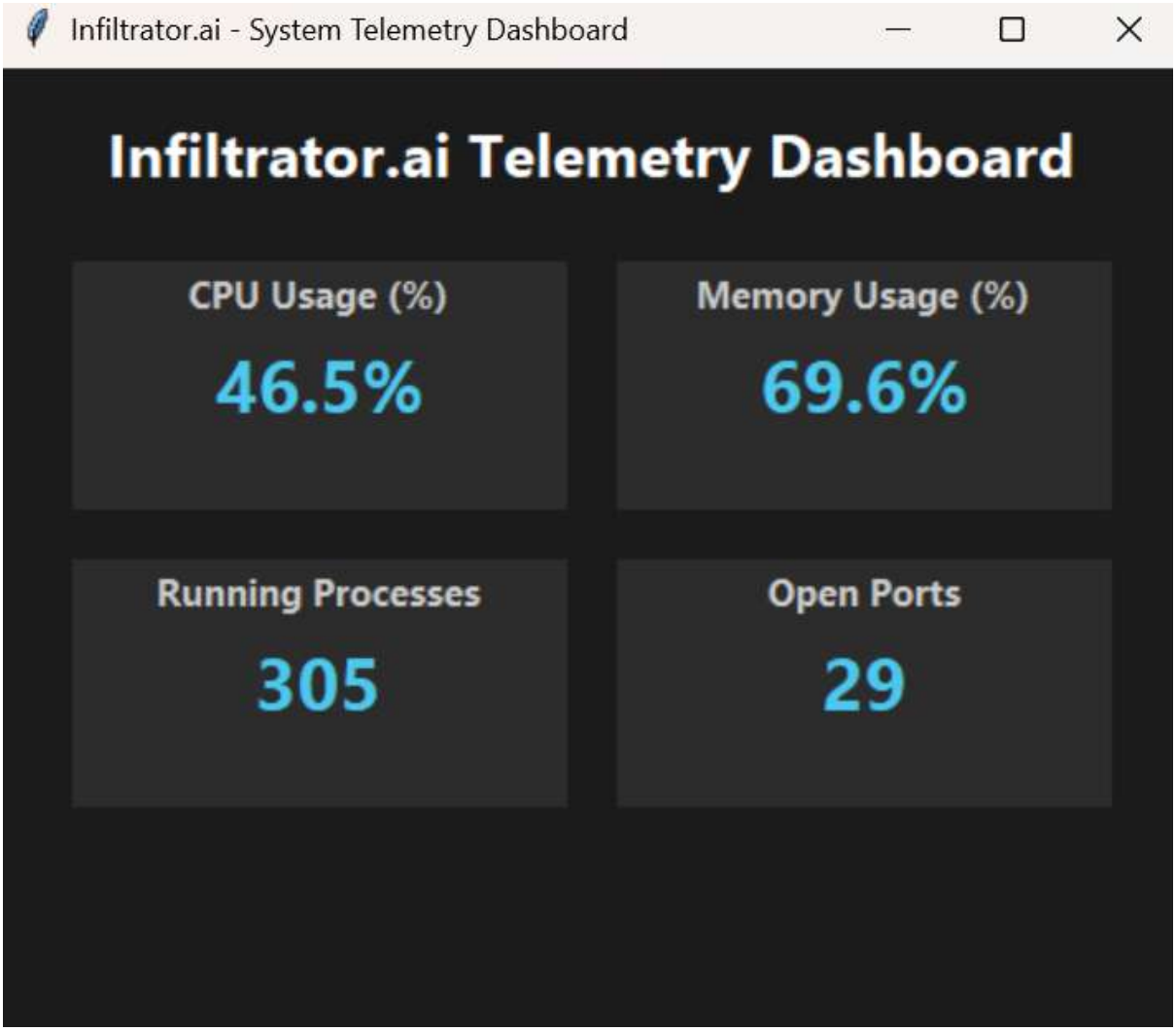
ChatGPT produced:

- Code structure
- Telemetry logic
- Tkinter UI
- Live data update mechanism
- Error handling
- Exe packaging steps

✏□ Your contributions (using VS Code):

- Minor formatting changes
- Fixing small indentation issues
- Adjustments to refresh timing
- Fine-tuning UI text
- Packing to EXE

Screenshot


Infiltrator.ai - System Telemetry Dashboard
Infiltrator.ai Telemetry Dashboard
CPU Usage (%)
46.5%
Memory Usage (%)
69.6%
Running Processes
305
Open Ports
29