Agentic AI Network & System Health Monitor

# 1. Introduction

The Agentic AI Network & System Health Monitor is a Python-based tool designed to provide real-time monitoring of key system and network performance metrics. It leverages locally running Ollama language models to analyze these metrics, identify potential anomalies, and provide concise, color-coded status updates or alerts directly in the terminal.

This tool acts as a simple "agent" operating on your local machine. It perceives the system's state (CPU load, GPU load, network traffic, latency, packet loss), reasons about this state using a local AI model, and acts by presenting an analysis and highlighting potential issues requiring user attention. Its goal is to offer a proactive and intelligent overview of system health without relying on external cloud services for analysis, ensuring data privacy.

The project acts as an intelligent Digital Shadow of your computer's network and system health. It mirrors the real-time status of the physical system through automated data collection and uses AI to interpret that status, but it doesn't interact back to control the system.

**What is Digital Shadow?**

1. **Digital Model:** A Digital Model is often a static representation or a model used for simulation *without* a live, automatic data feed from the physical asset. This project *is* connected to the live system and continuously pulls data. So, it's more than just a Digital Model.
2. **Digital Shadow:** This concept involves a digital representation that **automatically receives data** from its physical counterpart. The data flow is primarily **one-way** (physical system -> digital representation). The shadow reflects the *current* and *past* state of the physical asset based on sensor data. This project perfectly fits this:
   * It automatically pulls real-time data (CPU, GPU, Network stats, Ping results) *from* the physical computer.
   * The data flow is one-way: The script reads system state but *doesn't* send commands back to control or change the system's operation.
   * Its purpose is to monitor and reflect the *current* operational state and use AI to analyze that state.
3. **Digital Twin:** A Digital Twin implies a more sophisticated, **bidirectional link**. Data flows both ways (physical <-> digital). A Digital Twin can often be used for simulation ("what-if" scenarios), prediction, and potentially sending control commands back to optimize the physical asset. This project lacks:
   * **Bidirectional Control:** It doesn't modify the system based on its analysis.
   * **Predictive Simulation:** It analyzes the current state but doesn't simulate future outcomes.

# 2. Features

* **Real-time Monitoring:** Continuously gathers data on:
  + CPU Utilization (%)
  + NVIDIA GPU Utilization (%) (if available and pynvml installed)
  + Network Bandwidth (Download/Upload Mbps) for active interfaces.
  + Network Latency (ms) via ping to a specified target.
  + Network Packet Loss (%) via ping.
* **Local AI Analysis:** Utilizes a user-specified Ollama model running locally to interpret the collected metrics and provide health assessments.
* **Agentic Behavior:**
  + **Perception:** Senses CPU, GPU, Network I/O, Ping results.
  + **Reasoning:** Uses the Ollama model to analyze perceived data against normal parameters.
  + **Action:** Displays formatted metrics and color-coded AI analysis/alerts.
  + **Goal:** Monitor system health and alert on abnormalities.
  + **Environment:** The local host machine and its network connection.
* **Automatic Network Interface Detection:** Identifies active, non-loopback network interfaces with IP addresses and network traffic.
* **Configurable:** Allows setting the Ollama model, ping target, and update interval via command-line arguments.
* **Color-Coded Output:** Presents normal AI status reports in green and potential issues/alerts identified by the AI in red for quick visual identification. Internal errors or warnings are shown in yellow.
* **Cross-Platform Compatibility:** Designed to run on Windows, Linux, and macOS (requires platform-specific ping command availability).
* **Privacy-Focused:** All data collection and AI analysis happen locally; no system data is sent to external servers (besides the optional ping requests).

# 3. How it Demonstrates Agentic AI

While relatively simple, this tool embodies key characteristics of an AI agent:

1. **Perception:** It uses libraries like psutil, pynvml, and subprocess calls (ping) to perceive its environment – the state of the host system's CPU, GPU, and network interfaces, as well as the quality of its connection to an external target.
2. **Reasoning/Decision Making:** Instead of relying solely on predefined, rigid thresholds, it passes the perceived data to a Large Language Model (Ollama). The LLM performs a form of reasoning, comparing the current state against its general knowledge (or specifically guided knowledge via the prompt) of what constitutes "normal" or "problematic" system behavior.
3. **Action:** Based on the LLM's analysis, the agent acts by displaying the information on the console. Crucially, it modifies its output (color-coding) based on the AI's judgment, highlighting alerts distinctly from routine status updates.
4. **Goal-Oriented:** The agent has a clear objective: monitor system health and notify the user of potential issues indicated by the metrics.
5. **Autonomy:** Once launched, it operates continuously without direct user input for each cycle, gathering data, invoking the AI, and presenting results autonomously at the specified interval.

It differs from a basic monitoring script by incorporating the LLM's analytical capability to interpret the *combination* of metrics and make a judgment call, moving beyond simple threshold breaches.

# 4. Dependencies

## 4.1. Python Libraries

* **psutil:** Required for accessing system details like CPU usage, network interface information, and network I/O counters.
* **requests:** Required for making API calls to the locally running Ollama instance.
* **pynvml (Optional):** Required for monitoring NVIDIA GPU utilization. If not installed or if an NVIDIA GPU/driver is not present, GPU monitoring will be disabled gracefully.

## 4.2. External Software

* **Ollama:** Must be installed and running on the local machine. ([https://ollama.com/](https://www.google.com/url?sa=E&q=https%3A%2F%2Follama.com%2F))
* **An Ollama Model:** You need to have pulled at least one model compatible with Ollama (e.g., llama3, mistral, phi3).
* **ping utility:** The standard operating system ping command must be available in the system's PATH.

# 5. Installation

1. **Install Python:** Ensure you have Python 3.6 or newer installed.
2. **Install Ollama:** Follow the instructions on the [Ollama website](https://www.google.com/url?sa=E&q=https%3A%2F%2Follama.com%2F) to download and install Ollama for your operating system.
3. **Pull an Ollama Model:** Open your terminal and pull a model you want to use. For example:

ollama pull llama3 # Or mistral, phi3, etc.

1. **Install Python Dependencies:** Open your terminal and run:

pip install psutil requests pynvml

1. **Get the Script:** Get the Python code as network\_health.py
2. **Ensure Ollama is Running:** Start the Ollama application or service if it's not already running.

# 6. Usage

Run the script from your terminal using python:

python network\_health.py -m <model\_name> [options]

## 6.1. Command-Line Arguments

* -m MODEL\_NAME, --model MODEL\_NAME (**Required**): Specifies the name of the Ollama model to use for analysis (e.g., llama3, mistral:7b, llama3.2:3b). Make sure this model has been pulled via ollama pull <model\_name>.
* -t TIME\_INTERVAL, --time TIME\_INTERVAL (*Optional*): Sets the update interval in seconds between fetching metrics and getting AI analysis.
  + *Default:* 5 seconds.
* -p PING\_TARGET, --ping-target PING\_TARGET (*Optional*): Sets the hostname or IP address to ping for latency and packet loss measurements.
  + *Default:* 8.8.8.8 (Google Public DNS).

## 6.2. Examples

* **Run with Llama 3 model and default settings (5s interval, ping 8.8.8.8):**

python network\_health.py -m llama3.2:3b

* **Run with Mistral model, update every 10 seconds:**

python network\_health.py -m mistral-small3.1 -t 10

* **Run with Phi-3 model, update every 3 seconds, ping Cloudflare DNS:**

python network\_health.py -m phi4 -t 3 -p 1.1.1.1

# 7. How It Works (Internal Logic)

1. **Initialization:**
   * Parses command-line arguments.
   * Checks for essential Python libraries (psutil, requests).
   * Optionally attempts to import pynvml and initialize the NVML library for NVIDIA GPU monitoring. Sets a flag (nvml\_initialized) based on success.
   * Detects active network interfaces using psutil.
   * Prints startup configuration.
2. **Main Loop (Repeats every TIME\_INTERVAL seconds):**
   * Clears the terminal screen.
   * Prints the current timestamp and configuration header.
   * **Get Metrics:**
     + Fetches current overall CPU utilization using psutil.cpu\_percent().
     + If NVML was initialized, fetches GPU name(s) and utilization using pynvml.
     + Retrieves current network I/O counters (bytes\_sent, bytes\_recv) for monitored interfaces using psutil.net\_io\_counters().
     + Calculates Download/Upload speed (Mbps) by comparing current I/O counters with the previous loop's counters and the time elapsed. Handles counter rollovers simplistically.
     + If the ping thread is not running, starts a new background thread to execute the OS ping command against the target.
     + Checks a queue for results from the completed ping thread (latency, packet loss). Uses the last known results if the current test isn't finished.
   * **Display Metrics:** Prints the gathered CPU, GPU, Bandwidth, and Ping status to the console.
   * **Prepare AI Prompt:** Constructs a detailed prompt for the Ollama model, including all gathered metrics (CPU, GPU, Bandwidth per interface, Latency, Packet Loss), formatted clearly. It also includes instructions for the AI on how to analyze the data and the desired output format ("ALERT:" or "Status:").
   * **Call Ollama:** Sends the prompt to the running Ollama instance via an HTTP POST request to its API endpoint (/api/generate).
   * **Display AI Analysis:**
     + Receives the AI's response.
     + Checks if the response indicates an internal script/API error (starts with "Error:") and prints it in yellow.
     + Checks if the response starts with "ALERT:" and prints it in red.
     + Otherwise, assumes it's a normal status report, ensures it starts with "Status:", and prints it in green.
   * **Wait:** Calculates the time taken for the loop and sleeps for the remaining duration to maintain the target TIME\_INTERVAL.
3. **Shutdown:**
   * The loop continues until interrupted (Ctrl+C).
   * On interruption or error, it performs cleanup.
   * If NVML was initialized, it calls pynvml.nvmlShutdown().
   * Prints an exit message.

# 8. Output Explanation

The terminal output refreshes at the specified interval and typically shows:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--- Network & System Health Monitor (YYYY-MM-DD HH:MM:SS) ---

--- Author: Prashant Saxena (https://github.com/p3rcyshots) ---

Model: <model\_name> | Ping Target: <target\_ip> | Update Time: <interval>s

------------------------------------------------------------

CPU Usage: XX.X%

GPU Status:

GPU 0 (NVIDIA GeForce RTX XXXX): YY.Y% Util # (Shown in Blue if available)

N/A (pynvml not installed or NVML init failed) # (If GPU monitoring disabled)

Network Bandwidth:

Interface: <iface\_name\_1> # (Shown in Blue)

Download: DD.DD Mbps

Upload: UU.UU Mbps

Interface: <iface\_name\_2> # (If multiple active)

Download: DD.DD Mbps

Upload: UU.UU Mbps

(Waiting for first interval to calculate...) # (On first run)

(No I/O data reported...) # (If errors occur)

Ping Status:

Target: <target\_ip>

Avg Latency: LL.LL ms

Packet Loss: P.P%

(Waiting for current ping test to complete...) # (If ping thread running)

------------------------------------------------------------

AI Analysis:

Status: System metrics appear normal. Network latency and packet loss are low. CPU and GPU usage are within expected ranges. # (Shown in Green)

# OR

AI Analysis:

ALERT: High network latency (XXXms) detected to <target\_ip>. Packet loss is also elevated (X%). This may impact connectivity. # (Shown in Red)

# OR

AI Analysis:

Error: Could not connect to Ollama API at http://localhost:11434/api/generate. Is Ollama running? # (Shown in Yellow)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* **Green AI Analysis:** Indicates the AI model determined the current metrics are within normal operational parameters based on the prompt's guidance.
* **Red AI Analysis:** Indicates the AI model detected one or more metrics falling outside normal parameters (high latency/loss, high CPU/GPU, etc.) and flagged it as requiring attention.
* **Yellow AI Analysis:** Indicates an error within the monitoring script itself (e.g., cannot connect to Ollama, error parsing data).

# 9. Test Cases / Verification

1. **Baseline Run:**
   * Command: python network\_health.py -m <your\_model>
   * Expected: Script runs, displays metrics similar to the host's current state. AI Analysis should show "Status:" in green (assuming no actual system issues).
2. **Ollama Connection Failure:**
   * Action: Stop the Ollama service/application.
   * Command: python network\_health.py -m <your\_model>
   * Expected: Script runs, metrics display, AI Analysis shows a connection error message in yellow.
3. **Invalid Model Name:**
   * Command: python network\_health.py -m non\_existent\_model
   * Expected: Script runs, metrics display, AI Analysis shows an Ollama error (likely model not found) in yellow.
4. **High CPU Simulation:**
   * Action: Run a CPU stress test tool (e.g., stress on Linux, Prime95, or a heavy computational task).
   * Command: python network\_health.py -m <your\_model>
   * Expected: CPU Usage metric should rise significantly. AI Analysis may change to "ALERT:" (in red) mentioning high CPU usage.
5. **High GPU Simulation (NVIDIA Only):**
   * Action: Run a GPU benchmark (e.g., FurMark, Unigine Heaven) or a demanding game.
   * Command: python network\_health.py -m <your\_model>
   * Expected: GPU Status metric should show high utilization. AI Analysis may change to "ALERT:" (in red) mentioning high GPU usage.
6. **Network Latency/Loss Simulation:**
   * Action: Use network simulation tools (like tc on Linux, Clumsy on Windows) or physically degrade the connection (e.g., run over a poor WiFi signal, saturate bandwidth with large downloads/uploads). Alternatively, change the ping target (-p) to a known slow or unreliable host.
   * Command: python network\_health.py -m <your\_model> [-p <slow\_host>]
   * Expected: Ping Status should show increased Latency and/or Packet Loss. AI Analysis may change to "ALERT:" (in red) mentioning the network issues.
7. **Interface Change Simulation:**
   * Action: Disable the primary network interface while the script is running, then re-enable it or enable a different one (e.g., switch from Ethernet to WiFi).
   * Expected: The script should initially show errors calculating bandwidth or finding I/O data for the disabled interface. It should then attempt to re-detect interfaces and either recover monitoring on the original interface or switch to the new one, possibly resetting bandwidth stats.

# 10. Potential Improvements / Future Work

* **Historical Data & Trend Analysis:** Store metrics over time (e.g., in memory using collections.deque or to a file/database) and include averages, min/max, or deviations in the prompt for more sophisticated AI analysis.
* **Explicit Thresholds:** Add optional user-defined thresholds for basic metrics (e.g., latency > 200ms) to provide immediate visual warnings *in addition* to the AI analysis.
* **Configuration File:** Move settings like Ollama URL, default target, thresholds, etc., to a config file (e.g., YAML, INI) instead of hardcoding defaults.
* **Per-Process Monitoring:** Extend psutil usage to identify specific processes consuming high CPU. Correlate PIDs using GPU compute (via pynvml.nvmlDeviceGetComputeRunningProcesses()) with psutil to name high GPU consumers.
* **Broader GPU Support:** Integrate libraries for AMD (pyamdgpuinfo, rocm-smi) and potentially Intel GPU monitoring.
* **Error Recovery:** Implement more robust error handling, such as automatic retries for Ollama API calls or temporary fallback to simpler analysis if the AI fails repeatedly.
* **Logging:** Add logging to a file to record metrics, AI responses, and errors over time.
* **GUI:** Develop a graphical user interface (e.g., using Tkinter, PyQt, Kivy, or a web framework like Flask/Streamlit) for a more user-friendly display.
* **Structured AI Output:** Modify the prompt to request JSON output from the AI (e.g., {"status": "alert", "reason": "High Latency", "details": "..."}) for more reliable parsing in the script.

# 11. Troubleshooting & Notes

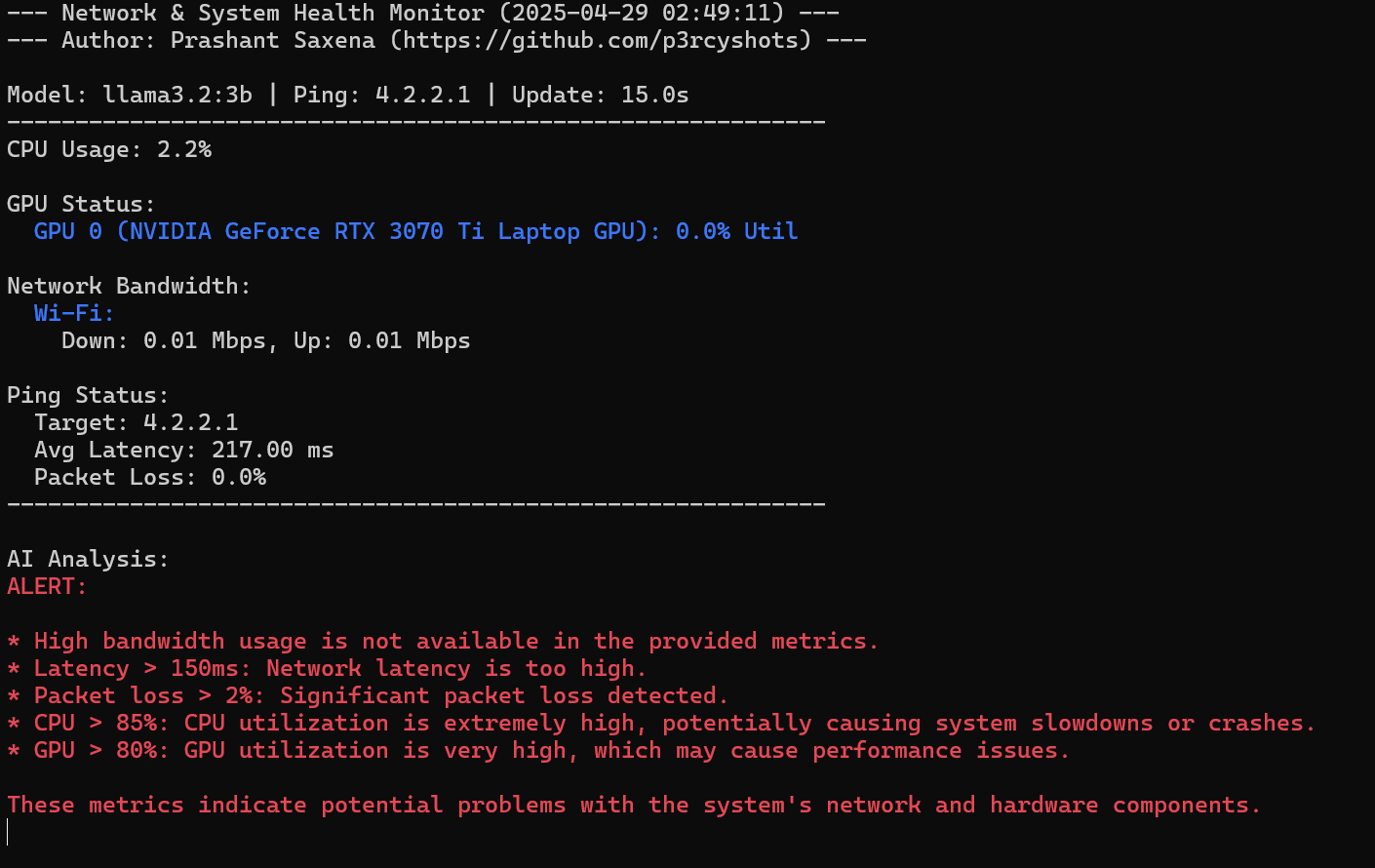
* **pynvml Issues:** Ensure you have the correct NVIDIA drivers installed. pynvml relies on the NVML library included with the drivers. On Linux, you might sometimes need sudo for NVML access.
* **ping Permissions:** On some systems, running ping might require specific permissions, especially if using options requiring raw sockets (though the basic usage here is usually fine). If ping fails with permission errors, try running the script with sudo (Linux/macOS) or as Administrator (Windows).
* **Ollama Not Found/Running:** Double-check that the Ollama service is running and accessible at http://localhost:11434. Ensure the specified model (-m) has been downloaded (ollama list).
* **Firewall:** Ensure no local firewall rules are blocking the script from accessing http://localhost:11434 or sending ICMP echo requests (ping).
* **Resource Usage:** The script itself consumes some CPU and memory. The Ollama model will consume significant resources (CPU, RAM, potentially GPU) when performing analysis. Monitor the script's own impact if necessary.
* **AI Accuracy:** The quality and relevance of the AI analysis depend heavily on the chosen Ollama model, the quality of the prompt, and the complexity of the system state. Smaller models might give less insightful or occasionally incorrect analyses.

# 12. License

*This project is licensed under GNU GPLv3*

# 12. Screenshots

## Screenshot 1



## Screenshot 2

