System State Forecast

Predicting User State via Resource Fingerprinting

January 31, 2026

System Forecast Team

Contents

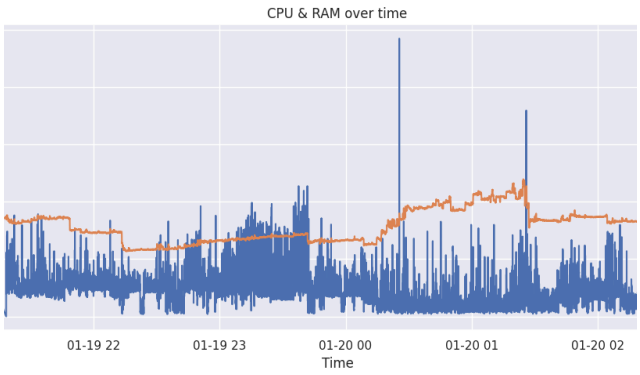
1 Project Overview

The Vision & Pivot

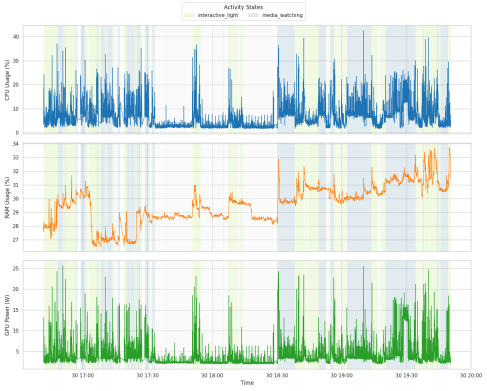
- **The Goal:** Classify user state (**Idle**, **Light Interactive**, **Media**) using only non-invasive hardware metrics.
- **The Pivot:** Originally tried predicting raw CPU usage.
  - **Failed because:** 99.9% of usage was 5-15% (The “10% guessing trap”).
  - **Result:** We cannot predict **when** a user starts a task, but we can identify **what** they are doing by the hardware reaction.

The Vision & Pivot (ii)

- **The Dataset:**
  - User A (Linux): 42,000 lines | User B: 6,000 lines | User C (Windows): 20,000 lines.
  - **Discarded:** All 68,000 lines because they lacked GPU metrics, which proved essential.



Old Dataset (CPU/RAM)



New Dataset (GPU included)

The Vision & Pivot (ii) (ii)

New Dataset (Network and Disk)

Data Collection Architecture

- **Environment:** Arch Linux + Niri Window Manager (deeply scriptable).
- **Sources:**
  - `psutil`: CPU, RAM, Disk, Network.
  - `intel_gpu_top`: The “Smoking Gun” for state detection.
  - `libinput`: Used for calculating ground-truth WPM/Idle for training.
- **Technical Hurdle:** To get GPU data without `sudo`, we applied:

```
sudo setcap cap_perfmon+ep /usr/bin/intel_gpu_top
```

Data Collection Architecture (ii)

- **Validation:** Real-time notification alerts (`watch_metrics.nu`).

```
# watch_metrics.nu
def main [...params: string] {
  loop {
    open comprehensive_activity_log.csv | last | select ...$params
    | notify-send -t 1000 $"($in)"
    sleep 1sec
  }
}
```

2 Engineering & Labeling

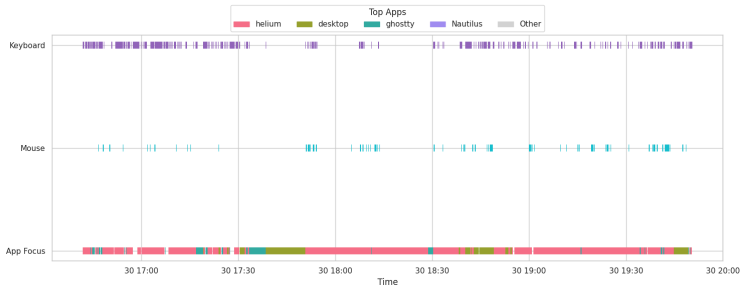
Labeling Strategy: Manual-but-Correct

1. **Manual Tagging:** Niri keyboard shortcuts bound to `current_state.txt`.
2. **Heuristic Idle:** Post-processed labels.

```
# live_inference.py (Heuristic Override)
if prediction == 'interactive_light':
    # If GPU is in deep sleep (RC6) and CPU is very low,
    # or if the GPU engines are essentially dead (< 1%):
    if (rc6_mean_5s > 99.0 and cpu_mean_5s < 5.0) or (max_gpu_raw < 1.0):
        prediction = 'Idle'
```

Labeling Strategy: Manual-but-Correct (ii)

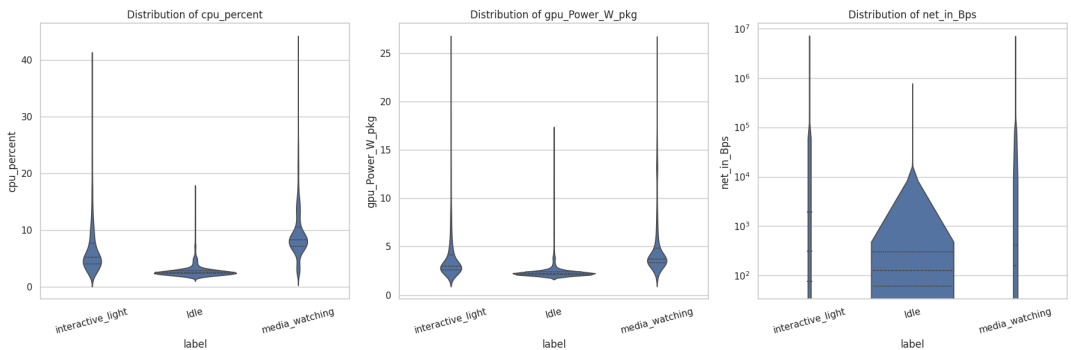
3. **Final Training Set:** 11,000 lines of high-quality labeled data.



Feature Engineering: Rolling Windows

Raw metrics are too noisy. We engineered features over two windows:

- **Short (5s):** Captures sudden bursts (e.g., page loads).
- **Long (30s):** Captures sustained patterns (e.g., 24fps video stream).
- **The "Max GPU" Insight:** Discovered that `max_gpu` (highest load on any single engine) is the 1 discriminator between "Idle" and "Static Reading."



3 The Machine Learning Model

Model Evolution & The Leakage Trap

- **Algorithm:** Random Forest Classifier (handles non-linear spikes).
- **The "Leaky" Model (v1):** Achieved 87% accuracy, but we realized it was reading `idle_time_sec`. Since we can't use input logs in production, we retrained.
- **The System-Only Model (v2):** Accuracy dropped to 82%, but performed significantly better in "live" inference scenarios by relying purely on resource patterns.

Top 5 Features (v2):

1. `gpu_RC6_pct_mean_5s`
2. `max_gpu_mean_5s`
3. `cpu_percent_mean_30s`
4. `cpu_percent_mean_5s`
5. `gpu_RC6_pct_std_5s`

v1 (Leaky) Performance:

Class	Precision	Recall	F1-Score
Idle	0.70	1.00	0.82
Interactive (Light)	0.83	0.97	0.89
Media Watching	0.97	0.77	0.86

The Inference Engine (Hybrid Logic)

To solve the “Idle vs. Light Reading” overlap (where metrics are nearly identical), we implemented a **Hardware-Aware Override**:

```
if prediction == 'interactive_light':
    # If GPU is in deep sleep (RC6) and CPU is very low,
    # or if the GPU engines are essentially dead (< 1%):
    if (rc6_mean_5s > 99.0 and cpu_mean_5s < 5.0) or (max_gpu_raw < 1.0):
        prediction = 'Idle'
```

Result: High-precision Idle detection without needing a mouse/keyboard hook.

4 Results & Conclusion

Model Performance (v2 System-Only)

Class	Precision	Recall	F1-Score
Idle	0.53	0.59	0.56
Interactive (Light)	0.76	0.94	0.84
Media Watching	0.97	0.76	0.85

Note: Media Watching precision is exceptional (97%) due to the unique signature of the Video Command Streamer (VCS) engine.

Outcomes & Future

Privacy: State detection is achieved without ever logging keystrokes or sensitive window titles in production.

Responsiveness: 1-second sampling rate with minimal overhead.

Future Work:

Hysteresis: Implement a state-switch delay to prevent “flickering.”

Include interactive-heavy activities (e.g., compiling code, running compute-intensive ML workloads, gaming) and background downloads (e.g., torrenting). Since I often download dozens of gigabytes of media, I believe this category is worth including.