PC Bottleneck Analyzer — Claude Code Session Prompts

Personal Tool First, Monetize Later

This is a "build it because I need it" project. The idea: a desktop or web app that scans your PC, shows you exactly what's holding you back, checks your BIOS/settings against what your hardware can actually do, and gives AI-powered recommendations to squeeze more performance out of what you already own.

Why this could become a business:

- PC building/optimization is a $50B+ market and growing
- r/buildapc has 7M+ members, r/pcmasterrace has 7M+, r/overclocking has 500K+
- Existing tools (UserBenchmark, HWiNFO, Speccy) show stats but DON'T tell you what to DO
- Nobody combines hardware detection + BIOS audit + AI recommendations in one place
- Monetization angle: free scan, pay for deep analysis / optimization guide / real-time monitoring
- Even if you never charge, this is a portfolio piece that shows serious technical ability

Architecture Decision: Desktop App (Electron/Tauri) vs Web App We need to READ local hardware data (CPU, GPU, RAM, BIOS settings, temps, clock speeds). This requires system-level access. Two approaches:

Option A: **Lightweight local agent + web dashboard** (RECOMMENDED)

- Small Python/Rust script that runs locally, collects hardware data, sends to your web app
- Web app does the analysis, shows the dashboard, runs AI recommendations
- Why: you already know web dev (Next.js), and the local agent is simple
- User flow: download agent → run it → it opens a browser tab with your results

Option B: **Full desktop app (Tauri or Electron)**

- Everything runs locally, no server needed
- More complex to build and distribute
- Better for privacy-sensitive users

We'll go with **Option A —** it plays to your strengths and ships faster.

SESSION 1: Local System Scanner (Python Agent)

Build a Python script that scans a Windows PC and collects all relevant hardware and performance data. This is the data collection layer — it needs to be thorough and accurate.

1. Hardware Detection:
Use these Python libraries:
- psutil (cross-platform system info)
- GPUtil (GPU info)
- wmi (Windows-specific deep hardware info)
- cpuinfo (detailed CPU info)
- subprocess to run Windows commands for data these libraries miss

Install: pip install psutil GPUtil py-cpuinfo wmi

Collect this data and output as structured JSON:

a. **CPU:**
   - Model name (e.g., "AMD Ryzen 7 5800X" or "Intel Core i7-13700K")
   - Architecture (x86_64, etc.)
   - Core count (physical cores)
   - Thread count (logical cores)
   - Base clock speed (GHz)
   - Current clock speed (GHz) — are all cores boosting correctly?
   - Max turbo/boost clock (GHz) — from spec sheet comparison
   - Cache sizes (L1, L2, L3)
   - Current temperature (°C)
   - Current usage per core (%)
   - Power draw if available (W)

b. **GPU:**
   - Model name (e.g., "NVIDIA RTX 4070 Ti")
   - VRAM total (GB)
   - VRAM used (GB)
   - GPU clock speed (MHz) — current vs base vs boost
   - Memory clock speed (MHz)
   - Temperature (°C)
   - Fan speed (%)
   - Driver version
   - GPU utilization (%)
   - PCIe generation and lane width (is it running x16 Gen4 like it should?)

c. **RAM:**
   - Total installed (GB)
   - Speed (MHz) — ACTUAL speed, not just what the stick is rated for
   - Number of sticks and which slots they're in
   - Channel mode (single/dual/quad) — THIS IS A HUGE BOTTLENECK IF WRONG
   - Timings (CL, tRCD, tRP, tRAS) if accessible

  - Form factor (DDR4/DDR5)
  - Current usage (GB used / available)

d. **Storage:**
  - For each drive:
    - Model name
    - Type (NVMe SSD, SATA SSD, HDD)
    - Capacity (GB)
    - Used space / free space
    - Interface (PCIe Gen3/Gen4, SATA III)
    - Health status (SMART data if accessible)
    - Read/write speeds (sequential) — run a quick benchmark or pull from SMART
    - Is the OS on this drive?
    - Is it the boot drive?

e. **Motherboard:**
  - Model name
  - Chipset
  - BIOS version
  - BIOS date (is it outdated?)

f. **Operating System:**
  - Windows version and build number
  - Is it up to date?
  - Power plan setting (Balanced, High Performance, Ultimate Performance)
  - Game Mode status (on/off)
  - Hardware-accelerated GPU scheduling (on/off)
  - Virtual memory / page file settings

g. **Network (bonus):**
  - Connection type (Ethernet/WiFi)
  - Current speed/bandwidth
  - Latency to common servers (ping google.com)

2. BIOS/Firmware Settings Detection:
This is the hard part and the most valuable. Many settings require reading from WMI or registry.

Try to detect:
- XMP/EXPO/DOCP profile: is it ENABLED? (This is the #1 missed setting — people buy 3600MHz RAM and it runs at 2133MHz because XMP isn't enabled)
  - Compare actual RAM speed vs rated speed to detect this
  - If actual speed < rated speed: FLAG THIS as critical finding
- Resizable BAR / Smart Access Memory: enabled or disabled?
  - Check via GPU driver or registry key

- TPM status
- Virtualization (VT-x / AMD-V): enabled? (matters for WSL, Docker, etc.)
- Secure Boot: enabled?
- Fast Boot: enabled? (can cause issues with dual boot or hardware changes)
- CSM (Compatibility Support Module): enabled/disabled? (should be disabled for modern systems)
- Above 4G Decoding: enabled? (required for Resizable BAR)

For settings we CAN'T read programmatically, note them as "Unable to detect — check BIOS manually" and provide instructions for where to find them.

3. Output Format:
- Save all collected data as a JSON file: system_scan.json
- Include a timestamp for when the scan was run
- Include a unique scan ID (UUID)
- Pretty-print the JSON for readability
- Also print a human-readable summary to the console:

  === PC Bottleneck Analyzer — System Scan ===
  CPU: AMD Ryzen 7 5800X (8C/16T, 3.8GHz base, currently at 4.5GHz, 62°C)
  GPU: NVIDIA RTX 4070 Ti (12GB VRAM, driver 551.23, 45°C, PCIe Gen4 x16)
  RAM: 32GB DDR4-3600 (Dual Channel, XMP ENABLED ✓)
  Storage: Samsung 980 Pro 1TB NVMe (PCIe Gen4, 45% used, healthy)
  OS: Windows 11 23H2 (up to date, High Performance power plan)

  ⚠ POTENTIAL ISSUES DETECTED: 2
  1. RAM running at 2133MHz but rated for 3600MHz — XMP may not be enabled
  2. Power plan set to "Balanced" — switch to "High Performance" for gaming

4. Make it runnable:
- Single file: scanner.py (or a small package with main.py)
- Run with: python scanner.py
- Should complete in under 30 seconds
- Handle errors gracefully — if a sensor isn't available, skip it and note "unavailable"
- Must work on Windows 10 and 11
- Should NOT require admin privileges for basic info (note which features need admin for elevated access)

TESTING:
- [ ] Run the scanner on your actual PC — does it detect your CPU correctly?
- [ ] GPU detected with correct model, VRAM, and driver version?
- [ ] RAM speed detected — is it the ACTUAL speed or the rated speed? (They may differ — this is intentional)
- [ ] Storage drives all listed with correct types and capacities?
- [ ] Motherboard model and BIOS version correct?
- [ ] Windows version correct?
- [ ] Power plan correctly identified?

- [ ] JSON output is valid (paste into a JSON validator)
- [ ] Console summary is readable and accurate
- [ ] Scanner completes in under 30 seconds
- [ ] Run without admin — what data is missing? Run WITH admin — what extra data appears?
- [ ] Run on a second PC if available — does it still work?
- [ ] If GPU is not NVIDIA (AMD or Intel), does it handle gracefully?

Show me the full JSON output and console summary from your actual scan.

SESSION 2: Web Dashboard (Display & Analysis)

Build the web dashboard that takes the scan data and presents it beautifully with analysis. This is a Next.js app — same stack as your other projects.

1. Project Setup:
- Next.js 14+ with App Router, TypeScript, Tailwind CSS v4
- shadcn/ui components: card, badge, tabs, progress, tooltip, separator, alert, dialog
- Dark theme by default (PC enthusiasts live in dark mode)
- Color scheme: dark slate background, neon accents (cyan #22D3EE for good, amber #F59E0B for warnings, red #EF4444 for problems, green #10B981 for optimal)
- The vibe: think HWiNFO meets a modern dashboard — technical but clean

2. Data Input:
Two ways to get scan data into the dashboard:

a. **File upload:** Drag-and-drop the system_scan.json file onto the page
   - Parse and validate the JSON
   - Show an error if the format is wrong

b. **API endpoint:** POST /api/scan with the JSON body
   - The Python scanner can POST results directly to the dashboard
   - Return a unique URL: /scan/[scanId] where the user can view their results
   - Store scan data (in database or local storage for now)

c. **Demo mode:** Include a sample scan (your actual PC data) so people can see the dashboard without scanning their own PC. Show it at /demo

3. Dashboard Layout:

Main page after scan upload shows tabs:

Tab 1: System Overview
- Hardware summary cards in a grid:
  - CPU card: model, cores/threads, current clock vs max boost, temp with color (green <70°C, yellow 70-85°C, red >85°C)
  - GPU card: model, VRAM (used/total bar), clock speed, temp, PCIe info
  - RAM card: total, speed, channel mode, usage bar
  - Storage card(s): each drive with type badge (NVMe/SSD/HDD), capacity bar, health status
  - Motherboard card: model, chipset, BIOS version + date
  - OS card: version, power plan, key settings status

- For each card: green checkmark badge if optimal, yellow warning if suboptimal, red alert if a problem is detected

Tab 2: Bottleneck Analysis (THE KEY FEATURE)
This is where we tell the user what's actually holding them back.

Analyze the hardware combination and identify bottlenecks:

a. **CPU vs GPU Balance:**
   - Compare CPU and GPU tier (using a hardcoded lookup table of common CPUs/GPUs ranked by performance tier)
   - If the CPU is significantly weaker than the GPU: "Your CPU (Ryzen 5 3600) may bottleneck your GPU (RTX 4090) in CPU-intensive games. Expect 15-30% lower FPS than this GPU is capable of at 1080p."
   - If the GPU is significantly weaker: "Your GPU is the limiting factor. Your CPU (i9-13900K) has plenty of headroom — upgrading your GPU would give the biggest performance boost."
   - If balanced: "Your CPU and GPU are well-matched. No significant bottleneck detected."
   - Show this as a visual balance scale or bar chart

b. **RAM Bottleneck:**
   - If single channel: "⚠ CRITICAL: Your RAM is running in single channel mode. This cuts memory bandwidth in half. If you have one stick, add a matching second stick. If you have two sticks, make sure they're in the correct slots (usually slots 2 and 4)."
   - If speed is below rated: "⚠ Your RAM is running at [actual]MHz but is rated for [rated]MHz. Enable XMP/DOCP in your BIOS to get the speed you paid for. This is a FREE performance boost."
   - If DDR4-2133 or DDR4-2400 with a modern CPU: "Your RAM speed is limiting your CPU performance, especially with Ryzen processors which are very sensitive to RAM speed."
   - If 8GB total: "8GB of RAM is below the recommended minimum for modern gaming and multitasking. You may experience stuttering when multiple applications are open."

c. **Storage Bottleneck:**
   - If OS is on an HDD: "⚠ Your operating system is on a hard drive. This is the single biggest upgrade you can make — moving to an SSD will make your PC feel brand new. Boot times will drop from 60+ seconds to under 15."
   - If games are on HDD: "Loading times in games could be improved by moving them to an SSD."
   - If NVMe drive is in a SATA slot or running at Gen3 when Gen4 is available: flag it

d. **Thermal Bottleneck:**

- If CPU temp >85°C at load: "Your CPU is thermal throttling. It's reducing clock speed to avoid overheating. Consider: better cooler, reapplying thermal paste, or improving case airflow."
- If GPU temp >85°C: similar warning

e. **Settings Bottleneck:**
- Power plan not on High Performance: "Switching to High Performance power plan can improve CPU responsiveness by 5-10% in gaming."
- XMP not enabled: flag with detailed instructions
- Resizable BAR not enabled: "Enabling Resizable BAR (free BIOS setting) can improve GPU performance by 5-15% in supported games."
- Old GPU drivers: "Your GPU driver is [X] months old. Update to the latest version for bug fixes and game optimizations."
- Old BIOS: "Your BIOS is from [date]. Check [motherboard manufacturer] for updates — newer BIOS versions often improve performance and compatibility."

Show each bottleneck as a card with:
- Severity badge: 🔴 Critical / 🟡 Warning / 🟢 Optimal
- What the issue is (plain English, not jargon)
- How much performance is being left on the table (estimate: "~15-30% FPS improvement possible")
- Exactly how to fix it (step-by-step instructions)
- Difficulty rating: Easy (change a Windows setting) / Medium (BIOS change) / Hard (hardware upgrade)

Tab 3: Recommendations
Prioritized list of actions:
1. Free fixes (settings changes, driver updates, BIOS tweaks)
2. Cheap fixes ($0-50: thermal paste, case fans, RAM stick)
3. Upgrade recommendations ($50-500: based on which component is the bottleneck)

For upgrade recommendations, suggest SPECIFIC parts:
- "Your best GPU upgrade path: RTX 4070 Super ($599) — removes GPU bottleneck entirely with your current CPU"
- "RAM upgrade: add a matching 16GB stick (~$35) for dual channel"
- Make these based on the user's current parts, not generic advice

Tab 4: Raw Data
- Collapsible JSON tree view of all collected data
- "Download Report" button: export as PDF or JSON
- "Copy to Clipboard" for sharing in forums (formatted for Reddit)

4. Performance Score:
Give the system an overall score out of 100:
- CPU performance: /25 (based on tier, thermals, clock speed vs expected)
- GPU performance: /25 (based on tier, thermals, VRAM utilization)
- RAM: /20 (speed, capacity, channel mode)
- Storage: /15 (type, health, is OS on SSD)

- Settings optimization: /15 (power plan, XMP, drivers, Resizable BAR)

Show as a big circular progress gauge at the top of the dashboard with a letter grade:
- 90-100: A+ "Your system is optimized. You're getting everything out of your hardware."
- 80-89: A "Great setup. A few tweaks could squeeze out more performance."
- 70-79: B "Good, but you're leaving performance on the table."
- 60-69: C "Several bottlenecks detected. Easy fixes available."
- Below 60: D "Significant issues found. Major free performance gains possible."

TESTING:
- [ ] Upload your actual system_scan.json — does it parse and display correctly?
- [ ] Every hardware card shows accurate data from the scan
- [ ] Bottleneck analysis identifies real issues (check if YOUR PC has any known bottlenecks and verify the tool finds them)
- [ ] If your RAM is running at the correct speed, the tool doesn't false-flag it
- [ ] Recommendations make sense for your actual hardware
- [ ] Performance score seems reasonable for your setup
- [ ] Demo mode works with the included sample data
- [ ] Raw data tab shows the complete JSON
- [ ] Mobile responsive (375px) — all cards and tabs work
- [ ] Dark theme is consistent throughout
- [ ] "Download Report" generates a file
- [ ] "Copy to Clipboard" formats nicely for Reddit (monospace, readable)
- [ ] Upload an invalid JSON file — shows a friendly error, not a crash
- [ ] Upload a scan from a very old PC (mock data with a Pentium + GT 710) — does the scoring/analysis still make sense?
- [ ] Upload a scan from a high-end PC (mock data with i9-14900K + RTX 4090) — does it correctly say "no major bottlenecks"?

Show me the full dashboard with your real scan data. Show test results.

SESSION 3: AI-Powered Deep Analysis

Add Claude AI analysis on top of the rule-based bottleneck detection. The AI can catch things the rule engine misses and give personalized, conversational advice.

1. AI System Analysis:
Create a "Get AI Analysis" button on the Bottleneck Analysis tab.

When clicked, send the full scan data to Claude API with this prompt:

"You are an expert PC hardware technician and performance optimization specialist with 15 years of experience building and tuning PCs. You've worked at system integrators, overclocking competitions, and now you help regular users get the most out of their hardware.

A user has scanned their PC. Here is their complete system data:
[full scan JSON]

Provide a thorough analysis in this structure:

Overall Assessment
2-3 sentences about the system as a whole. What kind of PC is this? (Budget gaming rig, workstation, all-rounder, etc.) Is the builder savvy or did they make some rookie mistakes?

The Good Stuff
What's well-configured or well-chosen about this system? Acknowledge what they got right. Be specific — don't just say 'good CPU', say 'the Ryzen 7 5800X is still a strong CPU for gaming in 2026 and pairs well with your GPU.'

Bottlenecks & Issues (ranked by impact)
For each issue found:
- **What's wrong** (one sentence, plain English)
- **How much it matters** (quantify: 'this is costing you roughly X% FPS in games' or 'this adds Y seconds to your boot time')
- **How to fix it** (step-by-step, assume the user knows where their BIOS is but not much more)
- **Difficulty and cost** (Free/Easy, Cheap/$20-50, Moderate/$50-200, Expensive/$200+)

The #1 Thing to Do Right Now
If you could only tell this user ONE thing to do to improve their PC, what would it be? Make it specific and actionable.

If You Had $200 to Upgrade
Given this system, what would give the biggest bang for $200? Be specific about exact parts and expected improvement.

If You Had $500 to Upgrade
Same as above but with $500 budget. Don't recommend replacing everything — focus on the weakest links.

RULES:
- Be conversational and friendly, not robotic
- Use specific numbers and benchmarks where possible
- Don't recommend things they don't need — if the system is fine, say so
- If their RAM speed suggests XMP isn't enabled, make this the FIRST recommendation — it's the most common free performance fix
- If their power plan is on Balanced, mention switching to High Performance
- Don't recommend overclocking unless they specifically have a K/X series CPU and a good cooler
- Be honest about diminishing returns — don't recommend a $500 upgrade that gives 5% improvement"

Display the AI response in a clean, readable card with collapsible sections.
Cache the response (don't re-analyze the same scan).

2. AI Chat Follow-Up:
After the initial analysis, add a chat interface where the user can ask follow-up questions:

- "Would my CPU bottleneck an RTX 4080?"
- "Is it worth upgrading from DDR4 to DDR5?"
- "What's the best cooler for my CPU under $50?"
- "How do I enable XMP in my BIOS? My motherboard is an MSI B550."

Implementation:
- Simple chat UI below the AI analysis (input field + send button + message history)
- Each message sends to Claude API with the system scan as context + conversation history
- System prompt includes: "You have the user's full system specs. Answer questions about their specific hardware. Be concise — most answers should be 2-4 sentences. Reference their specific parts by name."
- Show a "typing..." indicator while waiting for response

3. BIOS Optimization Guide:
Create a dedicated BIOS guide based on the user's specific motherboard.

When the user clicks "BIOS Optimization Guide":
- Detect their motherboard manufacturer (ASUS, MSI, Gigabyte, ASRock)
- Send to Claude API:
  "The user has a [motherboard model] with [CPU] and [RAM]. Generate a step-by-step BIOS optimization guide specific to this motherboard.

  Include:
  1. How to enter BIOS (the specific key for this manufacturer — DEL, F2, etc.)
  2. How to enable XMP/DOCP/EXPO (exact menu path for this manufacturer's BIOS)
  3. How to enable Resizable BAR / Above 4G Decoding (if applicable)
  4. Optimal fan curve settings for their cooler type
  5. Whether to enable or disable CSM, Fast Boot, Secure Boot
  6. Any manufacturer-specific performance settings (ASUS: AI Overclocking, MSI: Game Boost, etc.)
  7. Settings to NOT touch (things that could brick the system if changed incorrectly)

  Format each step with:
  - Exact menu path: Advanced > AMD CBS > DRAM Configuration > XMP Profile
  - What to change it to
  - Why
  - Screenshot description of what it looks like (since we can't show actual screenshots)"

Display as a step-by-step checklist the user can follow and check off.

4. Comparison Database:
Build a simple component lookup for putting results in context:

- Hardcode a JSON file with performance tiers for common CPUs and GPUs:
  {
    "cpus": {
      "AMD Ryzen 7 5800X": { "tier": "high", "gaming_score": 82, "release_year": 2020, "msrp": 449, "current_price_approx": 200 },
      "Intel Core i7-13700K": { "tier": "very_high", "gaming_score": 91, "release_year": 2022, "msrp": 409, "current_price_approx": 350 },
      ...
    },
    "gpus": {
      "NVIDIA RTX 4070 Ti": { "tier": "high", "gaming_score": 85, "vram_gb": 12, "release_year": 2023, "msrp": 799, "current_price_approx": 700 },
      ...
    }
  }

- Include the top ~50 most common CPUs and GPUs (last 5 years)
- Use this data for:
  - Showing where the user's parts rank: "Your CPU ranks in the top 25% of gaming CPUs"
  - Calculating the bottleneck balance between CPU and GPU
  - Recommending specific upgrade parts in the right tier

- If a user's part isn't in the database: note "Unknown model — AI analysis may be less precise" and rely more on the AI analysis

TESTING:
- [ ] AI analysis generates for your actual scan — is the advice accurate and specific?
- [ ] AI doesn't hallucinate parts you don't have
- [ ] AI correctly identifies your biggest bottleneck (manually verify — do you agree with its assessment?)
- [ ] Chat follow-up works: ask "is my CPU good enough for 1440p gaming?" — does it reference your specific CPU?
- [ ] BIOS guide generates for your specific motherboard — are the menu paths plausible for your manufacturer?
- [ ] Comparison database: your CPU and GPU are found and ranked correctly
- [ ] An unknown CPU/GPU (fake model) is handled gracefully
- [ ] AI responses are cached — clicking "Get AI Analysis" twice doesn't re-call the API
- [ ] Chat history persists during the session (doesn't reset on every message)
- [ ] Response time: AI analysis returns in under 20 seconds
- [ ] Mobile: AI analysis and chat work on phone

Show me the full AI analysis of your actual system. Show test results.

SESSION 4: Real-Time Monitoring Dashboard

Add a live monitoring mode that shows real-time system stats — like Task Manager but actually useful and pretty.

1. Live Data Collection (Python Agent Upgrade):
Update the Python scanner to support a "monitoring" mode:

python scanner.py --monitor

In monitoring mode:
- Collect data every 2 seconds
- Send to the web dashboard via WebSocket (use websockets library)
- Collect these metrics per tick:
  - CPU: usage per core (%), clock speed per core (GHz), temperature, power draw
  - GPU: usage (%), VRAM used (GB), clock speed (MHz), temperature, fan speed (%)
  - RAM: used (GB), available (GB), usage (%)
  - Network: upload/download speed (Mbps), latency (ms)
  - Per-process: top 10 processes by CPU usage, top 10 by RAM usage
  - Disk I/O: read/write speed per drive (MB/s)

2. WebSocket Server:
- The Python agent runs a local WebSocket server on localhost:8765
- The web dashboard connects to ws://localhost:8765
- Each message is a JSON blob with all current metrics + timestamp
- Handle disconnection gracefully (show "Scanner disconnected — restart the agent" in the UI)

3. Live Dashboard UI:

Layout: full-screen monitoring view at /monitor

Top row: Key metrics in large number cards
- CPU Usage: big % number with color (green <50%, yellow 50-80%, red >80%)
- CPU Temp: big °C number with color coding
- GPU Usage: same style
- GPU Temp: same style
- RAM Usage: X.X / XX.X GB with progress bar

Main area: Real-time charts (use Recharts with streaming data)

Chart 1: CPU Usage (last 60 seconds)
- Line chart, updates every 2 seconds
- Shows overall CPU % and per-core lines (toggleable)

- Y-axis: 0-100%

Chart 2: GPU Usage + VRAM (last 60 seconds)
- Dual-axis: GPU utilization % on left, VRAM usage GB on right
- Two lines on same chart

Chart 3: Temperatures (last 5 minutes)
- CPU temp and GPU temp lines
- Horizontal threshold lines at 80°C (warning) and 90°C (danger)

Chart 4: RAM & Disk I/O
- RAM usage over time
- Disk read/write speeds

Bottom section: Process List
- Table showing top 10 processes by resource usage
- Columns: Process Name, CPU%, RAM (MB), GPU% (if available), Disk I/O
- Sortable by any column
- Highlight processes using >10% CPU in yellow, >25% in red
- "Kill Process" button for each (sends command to the Python agent — requires admin)

4. Alert System:
Set thresholds that trigger visual alerts:
- CPU temp >85°C: red banner "CPU overheating — thermal throttling likely"
- GPU temp >90°C: red banner "GPU overheating — check fan curve"
- CPU usage 100% sustained for >30 seconds: "CPU maxed out — check top processes"
- RAM usage >90%: "Running low on memory — close some applications"
- Disk at 95%+ capacity: "Drive almost full — this causes performance degradation"

Show alerts as toast notifications + a persistent alert bar at the top.

5. Session Recording:
- "Record" button: starts recording all metrics to a file
- "Stop" button: saves the recording
- Can play back recordings with the same charts (like a DVR for your system stats)
- Use case: "Let me record while I play this game so I can see where the bottleneck was"
- Save recordings as JSON files with timestamps
- Show a list of past recordings with date, duration, and peak metrics

6. Game/App Performance Overlay (stretch goal):
If time permits:
- Detect when a game/fullscreen app launches
- Show a compact "mini-monitor" overlay: CPU%, GPU%, temps, FPS (if detectable)
- This is like MSI Afterburner's overlay but powered by our tool

- If this is too complex for one session, create the infrastructure and note it as a future feature

TESTING:
- [ ] Python agent starts in monitor mode and begins streaming data
- [ ] Web dashboard connects to WebSocket and shows live data
- [ ] Charts update in real-time (every 2 seconds) without lag
- [ ] Open a heavy application (game, video editor, or just many Chrome tabs) — do the charts spike correctly?
- [ ] Close the heavy application — do the charts drop back down?
- [ ] Process list shows the correct top processes (compare with Task Manager)
- [ ] Temperature warnings trigger at the right thresholds
- [ ] RAM warning triggers when running many apps
- [ ] Disconnect the Python agent — dashboard shows "disconnected" message
- [ ] Reconnect the agent — dashboard automatically reconnects and resumes
- [ ] Record a 30-second session → stop → play it back — does the playback match real-time?
- [ ] Charts scroll smoothly (no janky rendering or memory leaks after 10+ minutes)
- [ ] Dashboard works on a second monitor while gaming on the primary monitor
- [ ] Mobile: monitoring dashboard is usable on phone (simplified layout)

Show me the live dashboard with your real system data. Show test results.

SESSION 5: Benchmark & Comparison Engine

Build a benchmarking tool that tests the user's actual performance and compares it to what their hardware SHOULD achieve. This is how we prove whether there's a bottleneck or not.

1. Built-In Benchmarks:
Create lightweight benchmarks that run in the Python agent:

a. **CPU Benchmark (Multi-threaded):**
   - Run a standardized computation: matrix multiplication, prime number calculation, or compression task
   - Use all cores (multiprocessing)
   - Time the execution
   - Score = operations per second, normalized to a reference system
   - Takes ~15 seconds to run
   - Compare to database of known scores for the same CPU:
     "Your Ryzen 7 5800X scored 12,450. Expected range: 12,000-13,500. You're at 95% of expected performance. ✓"

b. **CPU Benchmark (Single-threaded):**
   - Same computation but single-core
   - Important because gaming is often single-thread dependent
   - Compare to expected single-thread scores

c. **RAM Bandwidth Test:**
   - Measure memory read/write speed (MB/s)
   - Compare to expected speed for their RAM configuration
   - DDR4-3600 dual channel should get ~50-55 GB/s
   - If significantly below: confirms XMP isn't enabled or RAM is misconfigured

d. **Storage Benchmark:**
   - Sequential read/write test (1GB file)
   - Random 4K read/write test (IOPS)
   - Compare to expected performance for drive type:
     - NVMe Gen4: expect 5000+ MB/s sequential read
     - NVMe Gen3: expect 3000+ MB/s
     - SATA SSD: expect 500+ MB/s
     - HDD: expect 100-200 MB/s

e. **GPU Benchmark (if possible):**
   - This is harder without DirectX/OpenGL rendering
   - Option 1: Run a simple compute shader benchmark if possible
   - Option 2: Use a pre-built tool like FurMark CLI or 3DMark demo score parsing
   - Option 3: Skip built-in GPU bench and just compare using known benchmark databases
   - If skipping: note "GPU benchmark requires a game or stress test. Run 3DMark TimeSpy and enter your score here for comparison."
   - Add a manual score input field for GPU benchmarks

2. Benchmark Results Page:
Show results with clear pass/fail indicators:

For each benchmark:
- Score achieved
- Expected score range for this hardware
- Percentage of expected performance (colored: green >90%, yellow 75-90%, red <75%)
- If below expected: specific diagnosis
  - CPU below expected + high temps = thermal throttling
  - CPU below expected + normal temps = background processes or power plan issue
  - RAM below expected = XMP not enabled or misconfigured timings
  - Storage below expected = wrong interface or driver issue

Visual: horizontal bar showing your score vs expected range:
```
[---|====YOUR SCORE====|---]
60%                 95%    100%
↑ minimum expected    ↑ you   ↑ max for this hardware
```

3. Historical Comparisons:

- Save benchmark results with timestamps
- Track performance over time:
  "Your CPU score has dropped 8% since last month. This could indicate thermal paste degradation, dust buildup, or new background processes."
- Show a line chart of scores over time
- Alert if performance drops >5% between runs

4. Community Comparison (future feature placeholder):
Structure the system so that in the future, users could optionally upload their benchmark scores to a central database and compare against others with similar hardware:
- Same CPU, different scores = configuration issue
- "Your Ryzen 7 5800X scores in the 45th percentile among all 5800X users. Users scoring higher tend to have: XMP enabled (89%), High Performance power plan (76%), updated BIOS (65%)"
- Don't implement the server for this yet — just design the data model and the comparison UI with mock data

5. One-Click Optimization:
Based on benchmark results + bottleneck analysis, offer automated fixes for Windows settings:

Safe, reversible optimizations the Python agent can apply:
- Set power plan to High Performance: powercfg /setactive 8c5e7fda-e8bf-4a96-9a85-a6e23a8c635c
- Disable startup programs that aren't needed (show the list, let user pick which to disable)
- Clear temp files: del /q/f/s %TEMP%*
- Set Windows visual effects to "Adjust for best performance"
- Enable Hardware-accelerated GPU scheduling (registry key)
- Enable Game Mode
- Disable unnecessary Windows services hogging resources
- Update GPU driver (detect current vs latest, provide download link)

For EACH optimization:
- Explain what it does in plain English
- Show expected impact (small / medium / large)
- "Apply" button that runs the command via the Python agent
- "Undo" button that reverts the change
- Log all changes made so the user can undo everything at once

⚠ SAFETY: Never auto-apply. Always show the user what will change and get confirmation. Create a restore point before any changes.

TESTING:
- [ ] CPU multi-thread benchmark runs and produces a score
- [ ] CPU single-thread benchmark runs and produces a score
- [ ] RAM bandwidth test returns a speed in GB/s
- [ ] Storage benchmark returns read/write speeds for each drive
- [ ] Scores are compared to expected ranges — are the expected ranges accurate for your hardware?

- [ ] If you artificially limit CPU (set power plan to Power Saver), does the benchmark score drop and the tool flag it?
- [ ] Historical comparison: run benchmark twice — both results saved and displayed
- [ ] Performance bar chart shows your score vs expected range correctly
- [ ] One-click optimizations: apply "High Performance power plan" — verify it changed
- [ ] One-click optimizations: undo the change — verify it reverted
- [ ] All optimizations show confirmation dialog before applying
- [ ] Benchmark runs don't freeze the UI (they run in the Python agent, not the browser)
- [ ] Total benchmark suite completes in under 2 minutes

Show benchmark results and test report.

SESSION 6: Testing, Polish & Packaging

Final session. Testing everything end-to-end, polishing the UI, and making the tool easy to distribute.

1. End-to-End Testing:

Test Scenario A: Full New User Flow
- Download the Python scanner
- Run: python scanner.py (basic scan)
- Upload system_scan.json to the web dashboard
- View System Overview — all cards populated correctly?
- View Bottleneck Analysis — reasonable findings?
- Click "Get AI Analysis" — helpful response?
- Ask a follow-up question in the chat — relevant answer?
- View BIOS Optimization Guide — specific to your motherboard?
- View Recommendations — actionable and prioritized?

REPORT: pass/fail for each step.

Test Scenario B: Live Monitoring Flow
- Run: python scanner.py --monitor
- Open /monitor in the browser
- Charts showing live data?
- Open Chrome with 50 tabs — CPU/RAM spike visible?
- Close Chrome — metrics drop back?
- Hit record → wait 30 seconds → stop record → play back — works?

REPORT: pass/fail for each step.

Test Scenario C: Benchmark Flow
- Run benchmarks from the dashboard
- All benchmarks complete without errors?

- Scores seem reasonable for your hardware?
- Expected ranges are accurate?
- Run benchmarks twice — historical comparison shows both results?
- Apply an optimization → re-run benchmark → score changes?
REPORT: pass/fail for each step.

Test Scenario D: Edge Cases
- Scan a system with no discrete GPU (integrated graphics only) — handles gracefully?
- Scan a system with no SSD (HDD only) — correct recommendations?
- Upload corrupted JSON — friendly error?
- Disconnect WebSocket during monitoring — reconnects or shows clear error?
- Run benchmark while another benchmark is running — handled?
REPORT: pass/fail for each step.

2. Packaging the Python Agent:
Make the scanner easy to install and run for non-technical users:

Option A: PyInstaller executable
- Use PyInstaller to create a single .exe file: pyinstaller --onefile scanner.py
- User just downloads and double-clicks — no Python installation needed
- Test: does the .exe work on a clean Windows install (no Python)?

Option B: Simple installer script
- Create install.bat that:
  1. Checks if Python is installed
  2. If not: prints instructions to install Python
  3. If yes: pip install requirements, run scanner.py
- Less elegant but simpler to build

Go with whichever is more reliable. The .exe is ideal for distribution.

3. Landing Page:
Create a simple landing page for the tool:
- Headline: "Is Your PC Running at Full Speed? Find Out in 60 Seconds."
- What it does: scan, analyze, recommend (3-step visual)
- Sample report screenshot from your actual PC
- Download button for the scanner (.exe or .py)
- "View Demo" link to /demo with sample data
- Don't gate behind auth — this is a free tool for now
- Include a "Star on GitHub" button if you want to open-source it

4. README & Documentation:
- Clear README.md:
  - What the tool does (2 sentences)

  - Screenshot of the dashboard
  - How to install and run (3 steps max)
  - How to use monitoring mode
  - How to run benchmarks
  - System requirements (Windows 10/11, Python 3.8+ if not using .exe)
  - Privacy note: "All data stays on your machine. Nothing is uploaded to any server. The AI analysis sends hardware specs to Claude API but no personal data."

5. Future Monetization Hooks (don't build yet, just design):
Document in a MONETIZATION.md file — ideas for when/if demand appears:

Free Tier (current):
- Basic scan + bottleneck detection
- Rule-based recommendations
- Live monitoring
- Benchmarks

Pro Tier ($5/month or $29 one-time):
- AI deep analysis (Claude API costs money per call)
- AI chat follow-up
- Custom BIOS guide for your specific motherboard
- Benchmark history tracking
- Performance alerts (notify when PC performance drops)
- Priority support

Revenue Alternatives:
- Affiliate links for recommended upgrades (Amazon Associates — 2-4% commission)
  "We recommend upgrading to a Corsair Vengeance DDR4-3600 16GB kit [affiliate link]"
  This alone could generate meaningful revenue if traffic is high
- Sponsored recommendations from hardware manufacturers (longer-term)
- API access for PC builders / system integrators

Distribution Channels:
- r/buildapc (7M members) — "I built a free tool that finds hidden bottlenecks"
- r/pcmasterrace (7M members)
- r/overclocking (500K members)
- YouTube: make a short video showing the tool finding a real bottleneck
- PC building forums: LinusTechTips, Tom's Hardware, Hardware Unboxed communities
- Product Hunt launch

TESTING:
- [ ] .exe file runs on Windows without Python installed
- [ ] Full flow from scanner.exe → upload → dashboard → AI analysis works
- [ ] Landing page loads and looks professional

- [ ] Download button works
- [ ] Demo mode shows realistic sample data
- [ ] README is clear enough that a friend could follow it
- [ ] No console errors on any page
- [ ] No hardcoded API keys in the distributed scanner
- [ ] Privacy: verify no personal data is sent anywhere (network tab check)

Show me the final state. Show all test results.

TOTAL SESSION MAP

Session	Focus	What Gets Built
1	System Scanner	Python agent that collects all hardware data
2	Web Dashboard	Display + rule-based bottleneck analysis + scoring
3	AI Analysis	Claude-powered deep analysis + chat + BIOS guide
4	Live Monitoring	Real-time stats dashboard with WebSocket streaming
5	Benchmarks	Performance tests + comparisons + one-click optimization
6	Testing & Packaging	Full testing, .exe packaging, landing page, docs

6 sessions. One per day = usable tool in under a week.

This one's shorter than the other two projects because there's no auth, no payments, no email system (yet). It's a utility — scan, analyze, fix. The value is immediate and obvious, which makes it great for Reddit/forum distribution.

When this gets traction (and PC tools DO get traction on Reddit — people love this stuff), the monetization is straightforward: the AI features cost you money (Claude API), so gate those behind a Pro tier. The affiliate links on hardware recommendations are passive revenue that scales with traffic.