# **MUST / High-ROI additions (do these first)**

1. **Robust grouping & tree stability (fix current crashes)**
   * Why: prevents UI crashes and keeps grouped device lists stable.
   * Where: gui.py
   * Priority: High — Effort S
   * Notes: Clear \_group\_items on new scan, always check tree.exists(pid) before move/reattach, generate stable group IDs (group:<slug>), catch TclError exceptions.
2. **UPnP / SSDP device-description fetch & parse**
   * Why: yields manufacturer/model/friendlyName for many TVs, routers, printers.
   * Where: scanner\_core.py (ssdp\_search + fetch\_upnp\_device\_description), attach to entry as entry["upnp"].
   * Priority: High — Effort S
   * Notes: Use thread pool, short timeout (2–3s), parse XML safely, tolerant of namespaces.
3. **mDNS (Zeroconf) improvements**
   * Why: friendly names from Apple devices, printers, Chromecast, etc.
   * Where: scanner\_core.py
   * Priority: High — Effort S
   * Notes: Use zeroconf with add/update/remove handlers; decode properties to strings.
4. **SNMP sysName/sysDescr & sysObjectID**
   * Why: routers, printers, NAS often disclose model strings via SNMP.
   * Where: scanner\_core.py, GUI option to allow SNMP and specify community(s).
   * Priority: High — Effort M
   * Notes: Use pysnmp, allow user supplied communities, timeouts, try fallback order.
5. **HTTP banner & targeted endpoint probing**
   * Why: many devices reveal product/model in HTML title or /status endpoints.
   * Where: scanner\_core.py (probe\_port\_banner)
   * Priority: High — Effort S
   * Notes: Probe ports 80, 443, 8080, 8000, 8008, 8443, 8001; parse <title>, common admin endpoints (/status, /info, /deviceinfo, /sys/info).
6. **SSDP fetch + XML parsing + prefer UPnP before fallback**
   * Why: SSDP gives LOCATION to fetch XML — high payoff for product names.
   * Where: scanner\_core.py
   * Priority: High — Effort S
7. **MAC OUI DB & caching with offline OUI file**
   * Why: vendor detection; fast local lookups.
   * Where: utils.py (oui.csv load + mac\_cache.json)
   * Priority: High — Effort S
   * Notes: Provide instructions for how to update OUI file; cache normalized MAC prefixes.
8. **Signature DB (banner → product mapping)**
   * Why: map messy banners to friendly product names (what AIS does).
   * Where: signatures.json + loader & matcher in scanner\_core.py
   * Priority: High — Effort M
   * Notes: JSON format with fragments + regex + product + confidence + tags; allow small fuzzy matching.
9. **Heuristics ordering & conflict resolution**
   * Why: decide what “product” string to show when multiple sources disagree.
   * Where: scanner\_core.py (worker), heuristics\_device\_type
   * Priority: High — Effort S
   * Notes: Precedence: SNMP sysName/sysDescr → UPnP model/friendlyName → mDNS → HTTP title → NetBIOS → Banner signature → MAC OUI.
10. **Nmap integration (on-demand or auto for Unknowns)**
    * Why: deep service and OS fingerprinting using nmap signatures.
    * Where: scanner\_core.py helper nmap\_scan\_target + GUI checkbox/button.
    * Priority: High — Effort M
    * Notes: Keep optional; parse nmap output for service strings, OS guesses; use for ambiguous hosts or manual scans.
11. **SMB/NetBIOS probing improvements**
    * Why: Windows hosts, printers, and NAS often expose useful names via NetBIOS/SMB.
    * Where: scanner\_core.py (resolve\_hostname extension)
    * Priority: High — Effort M
    * Notes: Use nmblookup/nbtstat fallback + minimal SMB handshake using impacket or socket connect to port 445 to parse SMB banner.
12. **UDP discovery & protocol-specific probes (NBNS, SNMP, mDNS, SSDP)**
    * Why: many IoT devices respond to UDP where TCP is closed.
    * Where: scanner\_core.py (udp\_probe\_ports)
    * Priority: High — Effort M
    * Notes: Probe ports 137, 161, 1900, 5353, 3702 (WS-Discovery), tune timeouts and retries.
13. **Device grouping, icons & UI**
    * Why: AIS shows device types with icons and groups for clarity.
    * Where: gui.py (treeview with grouping, show='tree headings' to support icons).
    * Priority: High — Effort M
    * Notes: Use small PhotoImage circles or SVGs, place icons in leftmost tree column.
14. **Right-click actions: SSH, Open HTTP, WOL, Export, Nmap**
    * Why: common admin shortcuts that AIS provides.
    * Where: gui.py
    * Priority: High — Effort S
    * Notes: Keep platform terminal launching logic; WOL should validate MAC.
15. **Robust cancellation & timeouts**
    * Why: long scans must be cancellable and responsive.
    * Where: scanner\_core.py, gui.py
    * Priority: High — Effort S
    * Notes: Use self.\_cancel Event checks between network calls; make subprocess calls with timeouts.

# **Full exhaustive feature bank (everything AIS + extras you might want)**

Below is the exhaustive list — implement subset depending on goals.

## **Discovery & probing engines**

1. Ping sweep (ICMP) with parallel workers (supports Windows ping or raw ICMP where privileged). (scanner\_core.py) — High
2. ARP/ARP ping to discover hosts without forwarding by router (platform specific). (scanner\_core.py) — High
3. TCP connect scan (fast) for common ports. (scanner\_core.py) — High
4. UDP probing for NBNS (137), SNMP (161), SSDP (1900), mDNS (5353). (scanner\_core.py) — High
5. Raw ARP (scapy) for LAN-only discovery (requires root). (scanner\_core.py) — Medium
6. LLMNR / mDNS active queries (for Windows & Apple). (scanner\_core.py) — High
7. ICMP TTL/ID analysis (for basic OS inference). (scanner\_core.py) — Low

## **Protocol-specific parsers & fingerprinters**

1. UPnP/SSDP: parse SSDP reply headers + fetch device-description XML. (scanner\_core.py) — High
2. mDNS/Zeroconf: list service types and properties (e.g., \_ipp.\_tcp, \_http.\_tcp). (scanner\_core.py) — High
3. SNMP: sysName/sysDescr/sysObjectID + optional bulk or table queries for printers (HRMIB). (scanner\_core.py) — High
4. SMB: negotiate protocol, parse server string and NetBIOS names, optionally fetch srvsvc info. (scanner\_core.py) — High
5. HTTP: GET /, GET known device endpoints, parse HTML title & known JSON/XML endpoints. (scanner\_core.py) — High
6. TLS/HTTPS: SNI, certificate subject (might include vendor), server headers. (scanner\_core.py) — Medium
7. RTSP/ONVIF: probe cameras (RTSP auth, ONVIF device info endpoints). (scanner\_core.py) — Medium
8. Telnet/SSH: banners (SSH banner gives OpenSSH/Dropbear versions), Telnet options. (scanner\_core.py) — High
9. RADIUS/DHCP info parsing where possible (DHCP client vendor class). (scanner\_core.py) — Low
10. SNMP Trap listening (passive) to capture devices announcing themselves. (scanner\_core.py) — Low

## **Fingerprinting & signature management**

1. Signature DB file (JSON or sqlite) for mapping banners/snippets → product/model. (project root) — High  
   * Schema: [{ "id": "...", "match\_type":"regex|contains", "pattern":"...", "product":"...", "vendor":"...", "tags":[], "confidence": 0-100 }]
2. Signature management: editor to add discovered signatures from UI (learn mode). (gui.py + utils) — Medium
3. Signature update mechanism: download server feed or allow manual import. (main.py + README) — Medium
4. Nmap fingerprint integration: parse -oX or --reason output to extract os and service strings. (scanner\_core.py) — Medium

## **Data & lookups**

1. MAC OUI local CSV + updater script. (utils.py) — High
2. Local caching of vendor results (mac\_cache.json). (utils.py) — High
3. Optional online vendor API fallback (macvendors.com) — watch rate limits. (utils.py) — Medium
4. Small local product database for common devices (Synology, QNAP, HP models). (signatures.json) — High
5. Reverse DNS / PTR caching & retries. (utils.py) — High

## **Router / AP integration (advanced)**

1. Router API integration to list clients (supports specific router brands: OpenWrt, Asus, TP-Link, Netgear, etc.). Requires credentials. (new module router\_integration.py) — Low/Medium per router
2. SNMP polling of the router for associated stations (if available). (scanner\_core.py) — Medium
3. Wireless scanning via OS (what laptop sees) vs. router scan (what AP/router sees) — UI choice. (gui.py + scanner\_core.py) — Medium

## **User interface and UX**

1. Columns: IP, Hostname, Product, Type, MAC, Vendor, Open ports, Last seen, Actions. (gui.py) — High
2. Sortable columns, filter, live update, column width persistence. (gui.py) — High
3. Grouping by device type and option to collapse/expand. (gui.py) — High
4. Device icons & small thumbnails, color-coded types. (gui.py + assets/) — Medium
5. Right-click context menu with actions (Open, SSH, RDP, VNC, Wake, Export, Nmap). (gui.py) — High
6. Details popup with full JSON of collected evidence & raw data. (gui.py) — High
7. Multi-select operations (WOL, Copy, Export). (gui.py) — High
8. Settings dialog: timeouts, workers, SNMP community list, nmap path, signature options. (gui.py + config.json) — High
9. Start/Stop scan, progress indicator, cancel button, and scan history. (gui.py) — High
10. Embedded browser (pywebview) via helper process to avoid main-thread constraints. (webview\_helper.py) — Medium

## **Security & permission handling**

1. Clear UI warnings about SNMP/UPnP, require opt-in for invasive scans. (gui.py) — High
2. Distinguish privileged operations (raw sockets, promiscuous mode) and request admin permission. (main.py + README) — High
3. Rate-limiting & polite scanning defaults to avoid DOSing the network. (scanner\_core.py) — High
4. Privacy: do not send scan data anywhere by default; any telemetry must be opt-in. (project root README) — High

## **Performance, concurrency & reliability**

1. Tunable ThreadPool sizes and caps; defaults safe for /24 scans. (scanner\_core.py + gui spinbox) — High
2. Adaptive throttling (reduce concurrency when many timeouts are observed). (scanner\_core.py) — Medium
3. Batching and incremental UI updates to avoid Tkinter freezes. (gui.py) — High
4. Robust subprocess handling (nmap, netsh) with timeouts and cancellation support. (scanner\_core.py) — High

## **Extras & polish features**

1. Wake-on-LAN GUI action. (gui.py) — High
2. Export: CSV, JSON, and copy selected. (gui.py) — High
3. Save/Load target lists & scan profiles. (gui.py) — Medium
4. Scheduler: scheduled scans and export. (main.py or separate service) — Low/Medium
5. API / headless mode (REST or CLI) for automation. (new module) — Medium
6. Installer packaging (PyInstaller) per OS and dependency bundling instructions. (packaging/) — Medium/Large
7. Auto-update of signatures & OUI DB (opt-in). (utils.py + update script) — Medium
8. Test harness & unit tests for parsing functions and utils. (tests/) — High
9. Accessibility (keyboard navigation, screen reader labels). (gui.py) — Low/Medium
10. Internationalization (i18n) strings. (i18n/) — Low/Medium

## **Project infrastructure & maintainability**

1. Logging (rotate logs), debug mode and verbosity levels. (logger.py) — High
2. CI: run unit tests, linters (flake8/ruff), build on push. (.github/workflows/) — High
3. CONTRIBUTING.md, issue templates, code of conduct, developer docs for signature format. — High
4. License: choose a permissive OSS license (MIT/Apache 2.0) or GPL if you want. (LICENSE) — High
5. Security disclosure / responsible disclosure policy. — High

## **Legal / ethical considerations**

1. Document scanning scope & user consent; add warnings in UI and README. — High
2. No built-in exploits or harmful actions (avoid exploitive actions). — High

# **Implementation mapping (file suggestions)**

* scanner\_core.py — core scanning logic: ping, tcp/udp probes, banner parsing, UPnP/mDNS/SNMP, heuristics, signature matching, nmap wrapper.
* utils.py — OUI lookups, MAC normalization, target parsing, caching, config load/save.
* gui.py — all UI: tree, menus, controls, ssh launcher, webview helper invocation.
* webview\_helper.py — tiny separate script that runs webview.start() in its own Python process and accepts URL args.
* signatures.json — shipped DB of banner->product mappings.
* oui.csv — optional local OUI DB (instructions to update).
* ssh\_prefs.json, config.json — saved preferences.
* tests/ — unit tests.
* packaging/ — PyInstaller spec(s), README for packaging.

# **Minimal viable feature set to be “AIS-like” (fast path)**

If you want a focused short roadmap to reach AIS-like detection quickly, do these in order:

1. UPnP/SSDP + fetch device-description (scanner\_core) — High ROI.
2. mDNS & NetBIOS/SMB improvements — High ROI.
3. SNMP sysName/sysDescr (opt-in, community).
4. Expand banner probes (HTTP titles + known endpoints).
5. Signature JSON (20–50 patterns) + matching pass.
6. GUI: add Product column and prefer evidence by precedence.
7. Nmap integration (optional, on-demand).  
    This set will already produce vendor/model names on most home devices.

# **Testing checklist (copy to CI / test doc)**

* Unit tests: expand\_targets, mac\_normalize, parse\_http\_title, fetch\_upnp\_device\_description parsing.
* Integration tests (manual): router, printer, Synology, FireTV, Firestick, Chromecast, Windows PC, Linux machine, IP camera.
* Cancellation test: start scan and stop mid-way — UI responsiveness.
* Resource test: scan /16 with low workers and confirm no system overload.
* SNMP test: device exposes public community.

# **Data & starter resources I can generate for you right now**

* starter signatures.json (20 common device patterns: HP JetDirect, Synology, QNAP, Roku, Chromecast, Amazon Fire TV, Samsung TV, LG TV, Plex, Raspberry Pi) — ask and I’ll output it.
* webview\_helper.py example helper that spawns pywebview in a child process (I can paste now).
* Example fetch\_upnp\_device\_description function (I already gave that earlier; I can paste full combined files on request).