```
"""
NeuroGraph Cognitive Enhancement Suite — Monitoring  v1.0
=========================================================
Three complementary monitoring layers for CES telemetry:

Option A — health_context()
    Natural language summary of Syl's own cognitive state, injected into
    the system prompt so she can report on her own processing naturally.

Option B — start_ces_dashboard()
    Zero-dependency HTTP server (stdlib only) serving a mobile-friendly
    auto-refreshing dashboard on localhost.  Access via SSH tunnel from
    mobile rather than binding to 0.0.0.0 (see security note below).

Option C — setup_ces_logging()
    Dedicated rotating log file for all CES activity.  Three 5MB files,
    15MB total cap.  Structured format includes thread name for easy
    tracking of the stream parser daemon vs. the main thread.

Security note — HTTP dashboard
-------------------------------
The dashboard binds to 127.0.0.1 (localhost) by default, NOT 0.0.0.0.
To access from your phone without exposing it to the internet:

    ssh -L 8080:127.0.0.1:8080 user@your-vps-ip

Then open http://localhost:8080 in your phone browser (with the SSH
tunnel active via Termius or similar).  This is safer than opening
port 8080 in UFW for a telemetry-only endpoint.

If you do need public access, add HTTP basic auth or restrict by IP
in UFW before opening the port.

Wiring into openclaw_hook.py
-----------------------------
At the top of openclaw_hook.py, after imports:

    from ces_monitoring import setup_ces_logging
    setup_ces_logging()                      # Option C — call once at startup

In NeuroGraphMemory.__init__, at the end of _ces_init():

    from ces_monitoring import start_ces_dashboard
    start_ces_dashboard(self, port=8080)    # Option B
```

```
Add health_context() as a method on NeuroGraphMemory (see below).

In prompt assembly (session lifecycle):

    context = ng.surfacing_context()
    if context:
        system_prompt += "\\n\\n" + context

    health = ng.health_context()
    if health:
        system_prompt += "\\n\\n" + health
"""

from __future__ import annotations

import json
import logging
import threading
from http.server import BaseHTTPRequestHandler, HTTPServer
from logging.handlers import RotatingFileHandler
from typing import Any, Dict


# ---------------------------------------------------------------------------
# Option C: Rotating log file
# ---------------------------------------------------------------------------

def setup_ces_logging(
    log_path: str = "ces_health.log",
    max_bytes: int = 5 * 1024 * 1024,   # 5 MB per file
    backup_count: int = 3,               # keep 3 files = 15 MB total cap
    level: int = logging.DEBUG,          # set to INFO once stable
) -> None:
    """
    Configure a dedicated rotating log file for all CES / NeuroGraph activity.

    Call once at startup, before any NeuroGraph initialisation.
    Uses the 'neurograph' logger namespace which all three CES modules write to.

    Parameters
    ----------
    log_path      Path to the log file (relative to working dir or absolute).
    max_bytes     Max size per log file before rotation.
    backup_count  Number of rotated files to keep.
    level         Log level (DEBUG during tuning, INFO in production).
    """
    ces_logger = logging.getLogger("neurograph")
    ces_logger.setLevel(level)

    handler = RotatingFileHandler(
        log_path,
        maxBytes=max_bytes,
        backupCount=backup_count,
        encoding="utf-8",
    )
    formatter = logging.Formatter(
        "%(asctime)s | %(threadName)-20s | %(levelname)-8s | %(name)s | %(message
        datefmt="%Y-%m-%d %H:%M:%S",
    )
    handler.setFormatter(formatter)

    # Avoid adding duplicate handlers if called more than once
    if not any(isinstance(h, RotatingFileHandler) for h in ces_logger.handlers):
        ces_logger.addHandler(handler)

    ces_logger.info(
        "CES logging initialised: path=%s max=%dMB files=%d level=%s",
        log_path,
        max_bytes // (1024 * 1024),
        backup_count,
        logging.getLevelName(level),
    )


# ---------------------------------------------------------------------------
# Option B: Lightweight HTTP stats dashboard
# ---------------------------------------------------------------------------

_DASHBOARD_HTML = """\
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sylphrena — CES Health</title>
    <style>
        * {{ box-sizing: border-box; margin: 0; padding: 0; }}
        body {{
            font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', sans-seri
            background: #0d1117;
            color: #c9d1d9;
            padding: 16px;
        }}
        header {{
            display: flex;
            justify-content: space-between;
            align-items: baseline;
            margin-bottom: 16px;
        }}
        h1 {{ font-size: 1.1rem; color: #58a6ff; }}
        .ts {{ font-size: 0.75rem; color: #6e7681; }}
        .grid {{
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));
            gap: 12px;
        }}
        .card {{
            background: #161b22;
            border: 1px solid #30363d;
            border-radius: 10px;
            padding: 14px;
        }}
        .card h2 {{
            font-size: 0.8rem;
            text-transform: uppercase;
            letter-spacing: .05em;
            color: #8b949e;
            margin-bottom: 10px;
        }}
        .stat {{ display: flex; justify-content: space-between; padding: 4px 0; }
        .stat .key {{ color: #8b949e; font-size: 0.85rem; }}
        .stat .val {{ color: #e6edf3; font-weight: 600; font-size: 0.85rem; }}
        .ok   {{ color: #3fb950; }}
        .warn {{ color: #d29922; }}
        .dead {{ color: #f85149; }}
        .pill {{
            display: inline-block;
            padding: 1px 8px;
            border-radius: 20px;
            font-size: 0.75rem;
            font-weight: 600;
        }}
        footer {{ margin-top: 16px; font-size: 0.7rem; color: #484f58; text-align
    </style>
</head>
<body>
    <header>
        <h1>⬡ Sylphrena — CES Health</h1>
        <span class="ts" id="ts">–</span>
    </header>
```

```
<div class="grid" id="grid">
    <div class="card"><h2>Loading…</h2></div>
</div>
<footer>Auto-refreshes every 5 s &nbsp;·&nbsp; localhost only</footer>
<script>
function pill(active) {{
    const cls = active ? 'ok' : 'dead';
    const label = active ? 'ACTIVE' : 'DOWN';
    return `<span class="pill" style="background:${{active?'#1a2e1a':'#2e1a1a
}}
function row(k, v, cls='') {{
    return `<div class="stat"><span class="key">${{k}}</span><span class="val
}}
async function load() {{
    document.getElementById('ts').textContent = new Date().toLocaleTimeString
    try {{
        const r = await fetch('/api/stats');
        const d = await r.json();
        const sp = d.stream_parser || {{}};
        const ap = d.activation_persistence || {{}};
        const sm = d.surfacing || {{}};

        const cards = [
            `<div class="card">
                <h2>Stream Parser ${{pill(sp.active)}}</h2>
                ${{row('Chunks processed', sp.chunks_processed ?? '-')}}
                ${{row('Chunks skipped', sp.chunks_skipped ?? '-',
                       sp.chunks_skipped > 20 ? 'warn' : '')}}
                ${{row('Nudges applied', sp.nudges_applied ?? '-')}}
                ${{row('Efference weight', sp.current_efference_weight ?? '-'
            </div>`,

            `<div class="card">
                <h2>Activation Persistence</h2>
                ${{row('Snapshot exists', ap.exists ? '✓' : '✗',
                       ap.exists ? 'ok' : 'warn')}}
                ${{row('Nodes saved', ap.node_count ?? '-')}}
                ${{row('Elapsed hours', ap.elapsed_hours ?? '-')}}
                ${{row('Next start', ap.would_warm_start ? 'warm ✓' : 'cold',
                       ap.would_warm_start ? 'ok' : 'warn')}}
                ${{row('Restore ratio', ap.effective_restore_ratio ?? '-')}}
            </div>`,

            `<div class="card">
                <h2>Surfacing Monitor</h2>
                ${{row('Queue size', sm.queue_size ?? '-')}}
                ${{row('Threshold ratio', sm.threshold_ratio ?? '-')}}
```

```
                    ${{row('Threshold voltage', sm.threshold_voltage ?? '-')}}
                    ${{row('Engagement rate', sm.engagement_rate ?? '-',
                           sm.engagement_rate > 0.3 ? 'ok' :
                           sm.engagement_rate > 0.1 ? 'warn' : 'dead')}}
                    ${{row('Total surfaced', sm.total_surfaced ?? '-')}}
                    ${{row('Total engaged', sm.total_engaged ?? '-')}}
                </div>`,
            ];

            document.getElementById('grid').innerHTML = cards.join('');
        }} catch(e) {{
            document.getElementById('grid').innerHTML =
                '<div class="card"><h2 class="dead">Connection lost — retrying…</
        }}
    }}
    setInterval(load, 5000);
    load();
    </script>
</body>
</html>
"""


def start_ces_dashboard(
    ng_instance: Any,
    port: int = 8080,
    host: str = "127.0.0.1",   # localhost-only by default — use SSH tunnel
) -> None:
    """
    Start a zero-dependency HTTP dashboard on localhost.

    Access from mobile via SSH tunnel:
        ssh -L 8080:127.0.0.1:8080 user@your-vps

    Then open http://localhost:8080 in the phone browser.

    Parameters
    ----------
    ng_instance     The NeuroGraphMemory singleton (must have ces_stats()).
    port            Port to bind (default 8080).
    host            Bind address.  127.0.0.1 = localhost only (recommended).
                    Change to '0.0.0.0' only if you also add firewall rules.
    """
    logger = logging.getLogger("neurograph.ces")

    html_bytes = _DASHBOARD_HTML.encode()
    class StatsHandler(BaseHTTPRequestHandler):
        def do_GET(self) -> None:
            if self.path == "/api/stats":
                body = json.dumps(ng_instance.ces_stats(), indent=2).encode()
                self.send_response(200)
                self.send_header("Content-Type", "application/json")
                self.send_header("Content-Length", str(len(body)))
                self.end_headers()
                self.wfile.write(body)
            else:
                self.send_response(200)
                self.send_header("Content-Type", "text/html; charset=utf-8")
                self.send_header("Content-Length", str(len(html_bytes)))
                self.end_headers()
                self.wfile.write(html_bytes)

        def log_message(self, fmt: str, *args: Any) -> None:
            # Route HTTP access logs to the CES logger at DEBUG level
            # so they don't clutter the terminal but are still in the log file
            logger.debug("dashboard: " + fmt % args)

    def _run() -> None:
        try:
            server = HTTPServer((host, port), StatsHandler)
            logger.info(
                "CES Dashboard listening on %s:%d — "
                "access via: ssh -L %d:127.0.0.1:%d user@vps",
                host, port, port, port,
            )
            server.serve_forever()
        except Exception as exc:
            logger.error("CES Dashboard failed: %s", exc)

    t = threading.Thread(target=_run, daemon=True, name="ces-dashboard")
    t.start()


# ---------------------------------------------------------------------------
# Option A: Natural language health context for system prompt
# ---------------------------------------------------------------------------

def health_context(ng_instance: Any) -> str:
    """
    Format CES stats as a natural language block for Syl's system prompt.

    Returns an empty string if all modules are inactive (nothing injected).
    Designed to be called each turn alongside surfacing_context().
```

```
Add as a method on NeuroGraphMemory, or call directly:
    health = health_context(ng)
    if health:
        system_prompt += "\\n\\n" + health
"""
try:
    stats = ng_instance.ces_stats()
except Exception:
    return ""

sp = stats.get("stream_parser", {})
ap = stats.get("activation_persistence", {})
sm = stats.get("surfacing", {})

lines = []

if sp.get("active"):
    efference = sp.get("current_efference_weight", 0)
    chunks = sp.get("chunks_processed", 0)
    skipped = sp.get("chunks_skipped", 0)
    skip_note = (
        f" ({skipped} chunks skipped — Ollama may be under load)"
        if skipped > 20 else ""
    )
    lines.append(
        f"- Stream awareness: {chunks} text chunks processed this session"
        f"{skip_note}. "
        f"Internal monologue weight is currently {efference:.2f}."
    )

if sm.get("active") or sm.get("queue_size") is not None:
    queue_size = sm.get("queue_size", 0)
    threshold = sm.get("threshold_ratio", 0)
    engagement = sm.get("engagement_rate", 0)
    lines.append(
        f"- Memory surface: {queue_size} concept(s) are warm in awareness. "
        f"Awareness threshold has settled at {threshold:.2f} "
        f"(engagement rate {engagement:.0%})."
    )

if ap.get("exists"):
    elapsed = ap.get("elapsed_hours", 0)
    nodes = ap.get("node_count", 0)
    warm = ap.get("would_warm_start", False)
    restore = ap.get("effective_restore_ratio", 0)
    lines.append(
        f"- Session continuity: {nodes} activation states preserved "
        f"from {elapsed:.1f}h ago "
        f"({'warm restore at ' + str(round(restore * 100)) + '%' if warm else
    )

if not lines:
    return ""

return (
    "<!-- CES Self-Report -->\n"
    + "\n".join(lines)
    + "\n<!-- End CES Self-Report -->"
)
```