```
"""
NG Ecosystem — E-T Systems Module Integration Standard

Single vendorable file that gives any E-T Systems module the full
three-tier learning architecture:

    Tier 1 (Standalone):  NGLite alone.  Local Hebbian learning.
                          Zero deps beyond ng_lite.py.  Always works.

    Tier 2 (Peer-pooled): NGPeerBridge.  Co-located modules share
                          learning events via ~/.et_modules/shared_learning/.
                          Auto-connects when the shared dir exists.

    Tier 3 (Full SNN):    NGSaaSBridge to full NeuroGraph Foundation.
                          Auto-upgrades when NeuroGraph is detected on
                          the same host via ETModuleManager.

The ecosystem is "Apple-like" by design:

  - Every module is independently useful at Tier 1.
  - Any two co-located modules get a free Tier 2 boost — no config needed.
  - When NeuroGraph is present, all co-located modules transparently
    upgrade to Tier 3: full STDP, hyperedges, predictive coding, and
    CES streaming.  The module code doesn't change.  The bridge swaps.

Usage (inside any E-T Systems module):

    from ng_ecosystem import NGEcosystem

    # In your module's __init__ or startup:
    eco = NGEcosystem.get_instance(
        module_id="trollguard",
        state_path="~/.trollguard/ng_lite_state.json",
    )

    # Use the ecosystem in your module's hot path:
    embedding = my_embedder(text)
    eco.record_outcome(embedding, target_id="threat:prompt_injection", success=Tr
    recs = eco.get_recommendations(embedding)
    novelty = eco.detect_novelty(embedding)
    ctx = eco.get_context(embedding)   # dict of cross-module insights

    # Inspect tier at any time:
    print(eco.tier)    # 1, 2, or 3
    print(eco.stats()) # full telemetry

    # Periodic save (call on graceful shutdown or on a timer):
    eco.save()

Framework adapters (optional, load separately):
    - openclaw_adapter.py  — on_message(text)/recall(text)/stats() for OpenClaw s

Canonical source: https://github.com/greatnorthernfishguy-hub/NeuroGraph
License: AGPL-3.0

# ---- Changelog ----
# [2026-02-22] Claude (Sonnet 4.6) — Initial creation.
#   What: NGEcosystem class — singleton wrapper implementing the
#         standardized E-T Systems optional integration protocol.
#         Handles Tier 1→2→3 progression, auto-upgrade on NeuroGraph
#         detection, graceful degradation, and unified telemetry.
#         Also defines NGEcosystemAdapter ABC for framework adapters.
#   Why:  Each module was wiring ng_lite + peer bridge + SaaS bridge
#         independently with no shared contract.  This file is the
#         single vendorable standard so all modules behave identically
#         from an integration perspective.
#   Settings: tier3_upgrade defaults to True (auto-upgrade when NeuroGraph
#         is found).  peer_sync_interval=100 (balances freshness vs I/O).
#         upgrade_poll_interval=300s (re-check for NeuroGraph every 5
#         minutes — handles cases where NeuroGraph is installed after
#         the module starts).
#   How:  get_instance() creates NGLite, then tries peer bridge, then
#         queries ETModuleManager for NeuroGraph.  All in try/except so
#         each tier attempt is fully independent.  A background thread
#         polls for tier upgrades at upgrade_poll_interval.
# -------------------
"""

from __future__ import annotations

import json
import logging
import os
import threading
import time
from abc import ABC, abstractmethod
from pathlib import Path
from typing import Any, Dict, List, Optional, Tuple

import numpy as np
```

```
logger = logging.getLogger("ng_ecosystem")

__version__ = "1.0.0"

# ---------------------------------------------------------------------------
# Constants
# ---------------------------------------------------------------------------

ET_MODULES_ROOT = Path.home() / ".et_modules"
SHARED_LEARNING_DIR = ET_MODULES_ROOT / "shared_learning"
REGISTRY_PATH = ET_MODULES_ROOT / "registry.json"

TIER_STANDALONE = 1   # NGLite only
TIER_PEER = 2         # + NGPeerBridge
TIER_FULL_SNN = 3     # + NGSaaSBridge (full NeuroGraph)

TIER_NAMES = {
    TIER_STANDALONE: "Standalone (Tier 1)",
    TIER_PEER: "Peer-pooled (Tier 2)",
    TIER_FULL_SNN: "Full SNN (Tier 3)",
}


# ---------------------------------------------------------------------------
# Framework Adapter Interface
# ---------------------------------------------------------------------------

class NGEcosystemAdapter(ABC):
    """Abstract base for framework-specific adapters over NGEcosystem.

    Implement this to expose the ecosystem to a specific framework.
    Each adapter is a singleton that wraps the shared NGEcosystem
    instance — the same ecosystem, different vocabulary.

    Provided implementations:
        - OpenClawAdapter  (openclaw_adapter.py, vendored separately)

    Custom adapters:
        Subclass NGEcosystemAdapter and implement all abstract methods.
        Call NGEcosystem.get_instance() inside __init__ to bind the
        shared ecosystem.  Maintain your own singleton if needed.

    Design contract:
        - Adapters MUST NOT bypass NGEcosystem internals.
        - Adapters handle embedding generation; NGEcosystem handles learning.
        - Adapters are optional and framework-specific.  The core
          NGEcosystem is framework-agnostic and always the source of truth.
```

```python
    """

    @abstractmethod
    def on_message(self, text: str) -> Dict[str, Any]:
        """Process one unit of framework input (message, request, event).

        Args:
            text: Raw text to process.

        Returns:
            Dict with at minimum: {"status": "ingested"|"skipped", "tier": int}
        """
        ...

    @abstractmethod
    def get_context(self, text: str) -> Dict[str, Any]:
        """Retrieve cross-module context for the given text.

        Args:
            text: Query text.

        Returns:
            Dict with recommendations, novelty score, and tier info.
        """
        ...

    @abstractmethod
    def stats(self) -> Dict[str, Any]:
        """Return framework-specific stats including ecosystem tier."""
        ...


# ---------------------------------------------------------------------------
# NGEcosystem Core
# ---------------------------------------------------------------------------

class NGEcosystem:
    """Singleton E-T Systems learning ecosystem for a module.

    Manages the full Tier 1→2→3 lifecycle automatically.  Modules
    call record_outcome(), get_recommendations(), detect_novelty(),
    and get_context() without knowing or caring which tier is active.

    Thread-safety: All public methods are safe to call from multiple
    threads.  The tier upgrade loop runs in a daemon thread.
    """
    _instances: Dict[str, "NGEcosystem"] = {}
    _lock = threading.Lock()

    def __init__(
        self,
        module_id: str,
        state_path: Optional[str] = None,
        config: Optional[Dict[str, Any]] = None,
    ):
        """
        Args:
            module_id: Unique module identifier (e.g., "trollguard").
                       Must match the module_id in et_module.json.
            state_path: Path to persist NGLite state JSON.
                        Defaults to ~/.et_modules/{module_id}/ng_lite_state.json
            config: Optional config overrides.  Keys:
                      peer_bridge.enabled (bool, default True)
                      peer_bridge.sync_interval (int, default 100)
                      tier3_upgrade.enabled (bool, default True)
                      tier3_upgrade.poll_interval (float, default 300.0)
                      ng_lite.*  (passed through to NGLite)
        """
        self.module_id = module_id
        self._config = {
            "peer_bridge": {
                "enabled": True,
                "sync_interval": 100,
            },
            "tier3_upgrade": {
                "enabled": True,
                "poll_interval": 300.0,
            },
        }
        if config:
            _deep_merge(self._config, config)

        # State persistence path
        if state_path:
            self._state_path = Path(state_path).expanduser()
        else:
            self._state_path = (
                ET_MODULES_ROOT / module_id / "ng_lite_state.json"
            )
        self._state_path.parent.mkdir(parents=True, exist_ok=True)

        # Internal state
        self._tier = TIER_STANDALONE
        self._ng: Any = None              # NGLite instance
        self._peer_bridge: Any = None     # NGPeerBridge instance
        self._saas_bridge: Any = None     # NGSaaSBridge instance
        self._ng_memory: Any = None       # NeuroGraphMemory (Tier 3)
        self._upgrade_thread: Optional[threading.Thread] = None
        self._shutdown_event = threading.Event()
        self._ops_lock = threading.Lock()

        # Boot sequence
        self._init_ng_lite()
        self._init_peer_bridge()
        self._init_tier3_upgrade()

        logger.info(
            "[%s] NGEcosystem ready at %s",
            module_id,
            TIER_NAMES[self._tier],
        )

    # -------------------------------------------------------------------
    # Singleton factory
    # -------------------------------------------------------------------

    @classmethod
    def get_instance(
        cls,
        module_id: str,
        state_path: Optional[str] = None,
        config: Optional[Dict[str, Any]] = None,
    ) -> "NGEcosystem":
        """Return the singleton NGEcosystem for this module_id.

        Thread-safe.  Subsequent calls with the same module_id return
        the existing instance regardless of state_path/config args.
        """
        with cls._lock:
            if module_id not in cls._instances:
                cls._instances[module_id] = cls(module_id, state_path, config)
            return cls._instances[module_id]

    @classmethod
    def reset_instance(cls, module_id: str) -> None:
        """Destroy the singleton for module_id (testing only)."""
        with cls._lock:
            inst = cls._instances.pop(module_id, None)
            if inst:
                inst._shutdown_event.set()
```

```
# -------------------------------------------------------------------
# Tier 1: NGLite init
# -------------------------------------------------------------------

def _init_ng_lite(self) -> None:
    """Initialize local NGLite substrate (always Tier 1)."""
    try:
        from ng_lite import NGLite  # vendored alongside this file
        ng_config = self._config.get("ng_lite", {})
        self._ng = NGLite(module_id=self.module_id, config=ng_config)
        if self._state_path.exists():
            self._ng.load(str(self._state_path))
            logger.debug("[%s] NGLite state loaded from %s", self.module_id,
    except Exception as exc:
        logger.error("[%s] NGLite init failed: %s", self.module_id, exc)
        self._ng = None

# -------------------------------------------------------------------
# Tier 2: NGPeerBridge init
# -------------------------------------------------------------------

def _init_peer_bridge(self) -> None:
    """Try to connect NGPeerBridge (Tier 2). Non-blocking."""
    if not self._config["peer_bridge"]["enabled"]:
        return
    if self._ng is None:
        return
    try:
        from ng_peer_bridge import NGPeerBridge  # vendored alongside
        bridge = NGPeerBridge(
            module_id=self.module_id,
            shared_dir=str(SHARED_LEARNING_DIR),
            sync_interval=self._config["peer_bridge"]["sync_interval"],
        )
        self._ng.connect_bridge(bridge)
        self._peer_bridge = bridge
        self._tier = TIER_PEER
        logger.info("[%s] NGPeerBridge connected (Tier 2)", self.module_id)
    except Exception as exc:
        logger.debug("[%s] NGPeerBridge unavailable: %s", self.module_id, exc

# -------------------------------------------------------------------
# Tier 3: NeuroGraph auto-upgrade
# -------------------------------------------------------------------

def _init_tier3_upgrade(self) -> None:
```

```python
        """Start background thread that polls for NeuroGraph availability."""
        if not self._config["tier3_upgrade"]["enabled"]:
            return
        # Try once immediately (NeuroGraph may already be running)
        self._try_tier3_upgrade()
        if self._tier == TIER_FULL_SNN:
            return  # Already upgraded; no need for polling thread

        poll_interval = self._config["tier3_upgrade"]["poll_interval"]
        t = threading.Thread(
            target=self._upgrade_loop,
            args=(poll_interval,),
            daemon=True,
            name=f"ng_eco_upgrade_{self.module_id}",
        )
        t.start()
        self._upgrade_thread = t

    def _upgrade_loop(self, interval: float) -> None:
        """Background polling loop for Tier 3 upgrade."""
        while not self._shutdown_event.wait(timeout=interval):
            if self._tier == TIER_FULL_SNN:
                break
            self._try_tier3_upgrade()

    def _try_tier3_upgrade(self) -> bool:
        """
        Detect NeuroGraph on this host and upgrade to Tier 3 if found.

        Detection strategy (in order):
          1. Check if NeuroGraphMemory is already imported (same process).
          2. Query ETModuleManager for NeuroGraph's install path.
          3. Check known install paths directly.

        Returns True if upgrade succeeded.
        """
        if self._ng is None:
            return False

        ng_memory = self._find_neurograph_memory()
        if ng_memory is None:
            return False

        try:
            from ng_bridge import NGSaaSBridge  # vendored from NeuroGraph
            bridge = NGSaaSBridge(ng_memory)
            with self._ops_lock:
                self._ng.connect_bridge(bridge)
                self._saas_bridge = bridge
                self._ng_memory = ng_memory
                self._tier = TIER_FULL_SNN
            logger.info(
                "[%s] Upgraded to NGSaaSBridge → full NeuroGraph SNN (Tier 3)",
                self.module_id,
            )
            return True
        except Exception as exc:
            logger.debug("[%s] Tier 3 upgrade attempt failed: %s", self.module_id
            return False

    def _find_neurograph_memory(self) -> Optional[Any]:
        """
        Return a live NeuroGraphMemory instance if NeuroGraph is available,
        else None.

        Three-probe strategy, each fully guarded:
          1. Already-imported singleton (same Python process).
          2. ETModuleManager registry lookup + dynamic import.
          3. Direct filesystem probe of known install paths.
        """
        # --- Probe 1: same process ---
        try:
            from openclaw_hook import NeuroGraphMemory
            mem = NeuroGraphMemory.get_instance()
            if mem is not None:
                logger.debug("[%s] NeuroGraph found in-process", self.module_id)
                return mem
        except Exception:
            pass

        # --- Probe 2: ETModuleManager registry ---
        try:
            ng_path = self._neurograph_path_from_registry()
            if ng_path:
                return self._import_neurograph_memory(ng_path)
        except Exception:
            pass

        # --- Probe 3: Known filesystem paths ---
        for candidate in _NEUROGRAPH_KNOWN_PATHS:
            path = Path(candidate).expanduser()
            if path.exists():
                mem = self._import_neurograph_memory(str(path))
                if mem is not None:
```

```
                return mem

        return None

    def _neurograph_path_from_registry(self) -> Optional[str]:
        """Read ETModuleManager registry and return NeuroGraph install path."""
        if not REGISTRY_PATH.exists():
            return None
        with open(REGISTRY_PATH) as f:
            registry = json.load(f)
        modules = registry.get("modules", {})
        ng_entry = modules.get("neurograph", {})
        install_path = ng_entry.get("install_path", "")
        return install_path if install_path else None

    def _import_neurograph_memory(self, ng_path: str) -> Optional[Any]:
        """Dynamically import NeuroGraphMemory from ng_path."""
        import sys
        orig_path = sys.path[:]
        try:
            if ng_path not in sys.path:
                sys.path.insert(0, ng_path)
            from openclaw_hook import NeuroGraphMemory
            return NeuroGraphMemory.get_instance()
        except Exception as exc:
            logger.debug("[%s] Dynamic NeuroGraph import from %s failed: %s",
                         self.module_id, ng_path, exc)
            return None
        finally:
            sys.path[:] = orig_path

    # ------------------------------------------------------------------
    # Public API (framework-agnostic)
    # ------------------------------------------------------------------

    @property
    def tier(self) -> int:
        """Current learning tier (1, 2, or 3)."""
        return self._tier

    @property
    def tier_name(self) -> str:
        """Human-readable tier name."""
        return TIER_NAMES.get(self._tier, "Unknown")

    def record_outcome(
        self,
        embedding: np.ndarray,
        target_id: str,
        success: bool,
        metadata: Optional[Dict[str, Any]] = None,
    ) -> Optional[Dict[str, Any]]:
        """Record a learning outcome.

        The embedding is the semantic representation of the input.
        The target_id is an opaque string representing what was decided
        (e.g., "model:llama3", "threat:prompt_injection", "action:search").

        Returns enriched response from the active bridge (Tier 2/3) or
        None if only Tier 1 is active.
        """
        if self._ng is None:
            return None
        with self._ops_lock:
            return self._ng.record_outcome(
                embedding, target_id, success, metadata=metadata
            )

    def get_recommendations(
        self,
        embedding: np.ndarray,
        top_k: int = 3,
    ) -> Optional[List[Tuple[str, float, str]]]:
        """Get recommendations from the active learning substrate.

        Returns list of (target_id, confidence, reasoning) or None.
        At Tier 1, returns local recommendations only.
        At Tier 2, includes cross-module peer patterns.
        At Tier 3, includes full SNN recommendations + hyperedge context.
        """
        if self._ng is None:
            return None
        with self._ops_lock:
            return self._ng.get_recommendations(embedding, top_k=top_k)

    def detect_novelty(self, embedding: np.ndarray) -> float:
        """Return novelty score [0.0=routine, 1.0=completely novel].

        At Tier 2+, novelty is cross-module: something novel to this
        module but known to a peer scores lower than it would at Tier 1.
        """
        if self._ng is None:
            return 1.0  # Conservative: unknown = novel
        with self._ops_lock:
            result = self._ng.detect_novelty(embedding)
        return result if result is not None else 1.0

    def get_context(
        self,
        embedding: np.ndarray,
        top_k: int = 3,
    ) -> Dict[str, Any]:
        """Unified context retrieval for prompt enrichment or decision support.

        Returns a dict suitable for injecting into a prompt or logging:
            tier:            int — current tier
            tier_name:       str — human-readable tier
            recommendations: list of (target_id, confidence, reasoning)
            novelty:         float — novelty score [0.0, 1.0]
            ng_context:      str|None — Tier 3 SNN surfaced context if available
        """
        recs = self.get_recommendations(embedding, top_k=top_k) or []
        novelty = self.detect_novelty(embedding)
        ng_context = None

        # Tier 3: ask NeuroGraph for surfaced cognitive context
        if self._tier == TIER_FULL_SNN and self._ng_memory is not None:
            try:
                ng_context = self._ng_memory.surface_context(embedding)
            except Exception:
                pass

        return {
            "tier": self._tier,
            "tier_name": self.tier_name,
            "recommendations": recs,
            "novelty": novelty,
            "ng_context": ng_context,
        }

    def save(self) -> None:
        """Persist NGLite state to disk."""
        if self._ng is None:
            return
        try:
            with self._ops_lock:
                self._ng.save(str(self._state_path))
            logger.debug("[%s] NGLite state saved to %s", self.module_id, self._s
        except Exception as exc:
            logger.warning("[%s] Save failed: %s", self.module_id, exc)
    def stats(self) -> Dict[str, Any]:
        """Return unified telemetry for logging, dashboards, or skill SKILL.md ou
        ng_stats: Dict[str, Any] = {}
        if self._ng is not None:
            try:
                ng_stats = self._ng.get_stats()
            except Exception:
                pass

        peer_stats: Dict[str, Any] = {}
        if self._peer_bridge is not None:
            try:
                peer_stats = self._peer_bridge.get_stats()
            except Exception:
                pass

        ng_memory_stats: Dict[str, Any] = {}
        if self._ng_memory is not None:
            try:
                ng_memory_stats = self._ng_memory.stats()
            except Exception:
                pass

        return {
            "ecosystem_version": __version__,
            "module_id": self.module_id,
            "tier": self._tier,
            "tier_name": self.tier_name,
            "ng_lite": ng_stats,
            "peer_bridge": peer_stats if peer_stats else None,
            "ng_memory": (
                {
                    "connected": True,
                    "version": ng_memory_stats.get("version", "unknown"),
                    "nodes": ng_memory_stats.get("graph", {}).get("node_count", "
                }
                if ng_memory_stats else None
            ),
            "state_path": str(self._state_path),
        }

    def shutdown(self) -> None:
        """Graceful shutdown: save state and stop the upgrade thread."""
        self._shutdown_event.set()
        self.save()
        logger.info("[%s] NGEcosystem shutdown complete", self.module_id)
```

```python
# ---------------------------------------------------------------------------
# Known NeuroGraph install paths (Tier 3 auto-detection)
# ---------------------------------------------------------------------------

_NEUROGRAPH_KNOWN_PATHS: List[str] = [
    "~/NeuroGraph",
    "~/.openclaw/skills/neurograph",
    "/opt/neurograph",
    "~/.et_modules/modules/neurograph",
]


# ---------------------------------------------------------------------------
# Internal helpers
# ---------------------------------------------------------------------------

def _deep_merge(base: dict, override: dict) -> None:
    """Recursively merge override into base in-place."""
    for key, val in override.items():
        if key in base and isinstance(base[key], dict) and isinstance(val, dict):
            _deep_merge(base[key], val)
        else:
            base[key] = val


# ---------------------------------------------------------------------------
# Convenience: module-level quick-start
# ---------------------------------------------------------------------------

def init(
    module_id: str,
    state_path: Optional[str] = None,
    config: Optional[Dict[str, Any]] = None,
) -> NGEcosystem:
    """One-call initialization for simple module integrations.

    Example:
        import ng_ecosystem
        eco = ng_ecosystem.init("trollguard")
    """
    return NGEcosystem.get_instance(module_id, state_path, config)
```