ET Module Integration Standard — v2.0

Overview

Every E-T Systems module is independently useful. Put two or more together and they automatically share intelligence. Add NeuroGraph and every module in the system gets full SNN capabilities — no config required.

This is the Apple ecosystem principle applied to AI infrastructure:

Modules present	What happens
Any single module	Tier 1: local Hebbian learning via ng_lite.py
2+ modules on same host	Tier 2: free cross-module pattern sharing via NGPeerBridge
Any module + NeuroGraph	Tier 3: all modules upgrade to full SNN (STDP, hyperedges, CES)

All tier transitions are **automatic, transparent, and non-breaking**. A module running at Tier 1 today upgrades to Tier 3 the moment NeuroGraph is installed — no code changes, no restarts required.

What Every Module Must Include

1. `ng_lite.py` (Tier 1) — always required

Vendored copy of the lightweight Hebbian learning substrate. Canonical source: NeuroGraph repo root. No external dependencies beyond numpy.

2. `ng_peer_bridge.py` (Tier 2) — always required

Vendored copy of the peer-to-peer learning bridge. Canonical source: NeuroGraph repo root. Reads/writes `~/.et_modules/shared_learning/<module_id>.jsonl`.

3. `ng_ecosystem.py` (Tier 1→2→3 orchestration) — always required

New in v2. The single vendorable file that manages the entire tier lifecycle automatically.

Replaces the manual “copy the TrollGuard init pattern” approach.

Canonical source: NeuroGraph repo root.

```
import ng_ecosystem

eco = ng_ecosystem.init(
    module_id="your_module_id",
    state_path="~/.your_module/ng_lite_state.json",
)

# That's it.  The ecosystem handles Tier 1→2→3 automatically.
# Your module uses eco.record_outcome(), eco.get_recommendations(), etc.
```

4. `et_module.json` (module manifest) — always required

Declares module identity to the ET Module Manager. Use the v2 schema: see `et_module_template.json` in the NeuroGraph repo.

Key v2 additions in the `ecosystem` block:

- `tier3_upgrade` : true if module uses ng_ecosystem.py auto-upgrade
- `ng_ecosystem_version` : version of ng_ecosystem.py vendored
- `openclaw_adapter` : filename of OpenClaw hook if present
- `capabilities` : free-form tags for feature discovery
- `provides` : what this module can serve as a backend for

5. `openclaw_adapter.py` (optional, OpenClaw only)

New in v2. If your module will be an OpenClaw skill, vendor this file alongside ng_ecosystem.py. Provides the standard `on_message(text)` / `recall(text)` / `stats()` interface.

Canonical source: NeuroGraph repo root.

The Standard Integration Pattern

Step 1 — `ng_ecosystem.py` in your module init

```
# At module startup (e.g. __init__.py, app.py, main.py):
```

```
import ng_ecosystem

eco = ng_ecosystem.init(
    module_id="your_module_id",      # must match et_module.json
    state_path="~/.et_modules/your_module/ng_lite_state.json",
    config={
        "peer_bridge": {"sync_interval": 100},     # optional overrides
        "tier3_upgrade": {"poll_interval": 300.0},
    },
)

# Use the ecosystem in your hot path:
embedding = your_embedder(text)
eco.record_outcome(embedding, target_id="action:selected", success=True)
recs = eco.get_recommendations(embedding)
ctx = eco.get_context(embedding)  # includes ng_context at Tier 3

# Check tier at any time (1, 2, or 3):
print(eco.tier, eco.tier_name)

# Graceful shutdown:
eco.save()
```

Step 2 — `et_module.json` (v2 schema)

Copy `et_module_template.json` from the NeuroGraph repo and fill it in. The `ecosystem` block is the key new addition:

```
{
  "_schema": "et_module/2.0",
  "module_id": "your_module_id",
  "version": "0.1.0",
  "ecosystem": {
    "ng_lite": true,
    "ng_lite_version": "1.0.0",
    "peer_bridge": true,
    "tier3_upgrade": true,
    "ng_ecosystem_version": "1.0.0",
    "openclaw_adapter": "your_module_hook.py",
    "openclaw_skill_name": "Your Module Name",
    "shared_learning_writes": true,
    "capabilities": ["your-capability-tag"]
  }
}
```

Step 3 — Register at install time

In your `install.sh` or `deploy.sh`:

```
ET_MODULES_DIR="${ET_MODULES_DIR:-$HOME/.et_modules}"
mkdir -p "$ET_MODULES_DIR/shared_learning"

python3 -c "
import json, sys, time
registry = '$ET_MODULES_DIR/registry.json'
try:
    with open(registry) as f:
        reg = json.load(f)
except:
    reg = {'modules': {}}

# Read et_module.json from the install dir
with open('$INSTALL_DIR/et_module.json') as f:
    manifest = json.load(f)

manifest['install_path'] = '$INSTALL_DIR'
reg['modules']['your_module_id'] = manifest
reg['last_updated'] = time.time()

with open(registry, 'w') as f:
    json.dump(reg, f, indent=2)
print('Registered with ET Module Manager')
"
```

Step 4 (OpenClaw only) — OpenClaw hook via OpenClawAdapter

```
# your_module_hook.py
from openclaw_adapter import OpenClawAdapter
import numpy as np

class YourModuleHook(OpenClawAdapter):
    MODULE_ID = "your_module_id"
    SKILL_NAME = "Your Module Name"
    WORKSPACE_ENV = "YOUR_MODULE_WORKSPACE_DIR"
    DEFAULT_WORKSPACE = "~/.openclaw/your_module"

    def _embed(self, text: str) -> np.ndarray:
        # Return your module's embedding, or:
        return self._hash_embed(text)  # zero-dep fallback
```

```
    def _module_on_message(self, text: str, embedding: np.ndarray) -> dict:
        # Your module's core processing.  Return a dict of results.
        return {}

    def _module_stats(self) -> dict:
        return {}

_INSTANCE = None

def get_instance() -> YourModuleHook:
    global _INSTANCE
    if _INSTANCE is None:
        _INSTANCE = YourModuleHook()
    return _INSTANCE
```

SKILL.md entry:

```
name: your_module_id
version: 0.1.0
autoload: true
```

Tier Behavior Reference

Tier 1 — Standalone (NGLite only)

- Pattern recognition via embedding similarity
- Hebbian learning: success boosts weight, failure reduces
- Novelty detection
- JSON persistence to disk
- Works completely offline, no network, no peers

Tier 2 — Peer-pooled (+ NGPeerBridge)

Adds on top of Tier 1:

- Shared JSONL event log at `~/.et_modules/shared_learning/`
- Cross-module recommendations: TrollGuard's threat patterns inform TID's routing
- Cross-module novelty detection: novel to you but known to a peer = lower score

- Async sync (default every 100 outcomes) — <5ms overhead per op
- Auto-connects on startup if shared dir exists and any peer JSONL is present

Tier 3 — Full SNN (+ NGSaaSBridge → NeuroGraph)

Adds on top of Tier 2:

- Full STDP learning (causal, temporally precise)
- Hyperedge associations (N-ary relationships, not just pairwise)
- Predictive coding (surprise-driven exploration)
- CES streaming (real-time activation warmth, surfacing monitor)
- Cross-session activation persistence
- `get_context()` returns `ng_context` string ready for prompt injection

Tier 3 upgrade is **automatic**: NGEcosystem polls for NeuroGraph every 5 minutes (configurable). No restart required. The bridge swaps silently.

Files to Vendor (copy from NeuroGraph repo)

File	Required	Notes
`ng_lite.py`	Always	Standalone learning substrate
`ng_peer_bridge.py`	Always	Tier 2 peer bridge
`ng_ecosystem.py`	Always	Tier orchestration (v2 new)
`openclaw_adapter.py`	OpenClaw only	Hook base class (v2 new)
`ng_bridge.py`	Optional	NGSaaSBridge — auto-used by ng_ecosystem.py when NeuroGraph is found; only need to vendor if you want to connect manually

---

Module-Specific Notes

NeuroGraph

NeuroGraph IS Tier 3. It does not upgrade itself. It provides the upgrade.

- `ng_ecosystem.py` should be initialized with `tier3_upgrade.enabled: False`
- `openclaw_hook.py` continues to serve as the OpenClaw integration (it was written before OpenClawAdapter existed; migrating it to the adapter pattern is a future nice-to-have, not required)
- `et_module.json` sets `"provides": ["*"]` — signals to ETModuleManager that it can upgrade any co-located module
- The peer bridge (`NGPeerBridge`) is still active so NeuroGraph participates in Tier 2 cross-module learning in addition to providing Tier 3

The-Inference-Difference

- Wire `ng_ecosystem.init("inference_difference")` at app startup
- Call `eco.record_outcome(embedding, target_id="model:{name}", success=...)` after each routing decision
- Call `eco.get_recommendations(embedding)` to get cross-module routing hints
- If using OpenClaw: subclass `OpenClawAdapter` as shown above

TrollGuard

- See `trollguard_hook_example.py` in the NeuroGraph repo for a complete reference implementation using OpenClawAdapter
- `eco.record_outcome(..., target_id="threat:{label}", success=not is_threat)`
- Novelty score from `eco.detect_novelty(embedding)` can inform scan depth

Cricket

- Same pattern as TrollGuard / TID
- `target_id` convention: `"action:{action_name}"` or `"intent:{intent}"` as appropriate for Cricket's domain

---

Directory Structure Reference

```
~/.et_modules/
├── registry.json                    # ETModuleManager module index
```

```
└── shared_learning/                    # NGPeerBridge event exchange
    ├── neurograph.jsonl
    ├── trollguard.jsonl
    ├── inference_difference.jsonl
    ├── cricket.jsonl
    └── _peer_registry.json

~/.et_modules/{module_id}/              # Per-module state (ng_ecosystem default)
    └── ng_lite_state.json
```

Design Principles (unchanged from v1)

1. **Graceful degradation is mandatory.** Every tier upgrade attempt is wrapped in try/except. A failed Tier 3 probe never affects Tier 1 or 2.
2. **The module doesn't know which tier it's on.** NGBridge is the universal interface. NGEcosystem swaps the implementation.
3. **Shared directory is the coordination point.** `~/.et_modules/` is the agreed root. All modules read/write there.
4. **Manifests are the discovery mechanism.** `et_module.json` is the contract. The ETModuleManager finds modules by manifest.
5. **Living changelogs in every file.** Format:

```
# ---- Changelog ----
# [YYYY-MM-DD] Author — Short title.
#   What: What was changed.
#   Why:  Why it was changed.
#   Settings: Defaults chosen and reasoning.
#   How:  Implementation approach.
# --------------------
```

Changelog

v2.0 (2026-02-22)

- **Added `ng_ecosystem.py`** — canonical Tier 1→2→3 orchestration. Replaces copy-paste initialization pattern from the v1 primer.

- **Added `openclaw_adapter.py` —** standardized OpenClaw hook base class. All modules expose identical `on_message()/recall()/stats()` vocabulary.
- **Updated `et_module.json` to v2 schema —** added `ecosystem` block with `tier3_upgrade`, `ng_ecosystem_version`, `openclaw_adapter`, `capabilities`, and `provides` fields.
- Added `et_module_template.json` as the canonical starting point for new modules.
- Updated module-specific notes for all four modules.

v1.0 (2026-02-17)

- Initial ET Module Manager integration primer.
- TrollGuard established as reference implementation.
- Three-tier architecture defined.