NeSy-Core — Production File Structure

```
nesy-core/
│
├── README.md
├── LICENSE
├── pyproject.toml                  # Build system config (PEP 517)
├── setup.cfg                       # Package metadata
├── Makefile                        # Dev shortcuts: make test, make lint, make b
├── .env.example                    # Environment variables template
├── .gitignore
├── .pre-commit-config.yaml         # Code quality hooks
│
├── nesy/                           # ← Main Python package
│   ├── __init__.py                 # Public API exports
│   ├── version.py                  # Single source of truth for version
│   │
│   ├── core/                       # Core framework — no ML dependencies
│   │   ├── __init__.py
│   │   ├── types.py                # All dataclasses: PresentSet, NullSet, NSIOu
│   │   ├── exceptions.py           # Custom exceptions: NeSyError, SymbolicConfl
│   │   ├── config.py               # Global config with Pydantic validation
│   │   └── registry.py             # Component registry for plugins
│   │
│   ├── symbolic/                   # Symbolic reasoning engine
│   │   ├── __init__.py
│   │   ├── engine.py               # Main SymbolicEngine class
│   │   ├── logic.py                # First-order logic primitives, predicates
│   │   ├── rules.py                # Rule definition, loading, validation
│   │   ├── solver.py               # Constraint solver (Z3 integration)
│   │   ├── betti.py                # Topological consistency checker
│   │   └── ontology/
│   │       ├── __init__.py
│   │       ├── loader.py           # Load OWL, RDF, custom ontologies
│   │       └── adapters/
│   │           ├── owl.py
│   │           └── rdf.py
│   │
│   ├── neural/                     # Neural backbone
│   │   ├── __init__.py
│   │   ├── base.py                 # Abstract NeSyModel base class
│   │   ├── grounding.py            # Symbol grounding: embeddings → predicates
```

```
│   │   ├── bridge.py              # Neural ↔ Symbolic bidirectional bridge
│   │   ├── backbones/
│   │   │   ├── __init__.py
│   │   │   ├── transformer.py     # Transformer backbone wrapper
│   │   │   └── gnn.py             # Graph Neural Network backbone
│   │   └── loss.py                # Symbolic-guided loss functions
│   │
│   ├── metacognition/             # ← PRIMARY NOVELTY LAYER
│   │   ├── __init__.py
│   │   ├── monitor.py             # MetaCognitionMonitor — main class
│   │   ├── confidence.py          # Three-score confidence: factual/reasoning/b
│   │   ├── trace.py               # ReasoningTrace builder
│   │   ├── doubt.py               # Self-doubt layer: when to halt output
│   │   └── calibration.py         # Calibration tuning: confidence vs actual ac
│   │
│   ├── continual/                 # Continual learning — no catastrophic forget
│   │   ├── __init__.py
│   │   ├── learner.py             # ContinualLearner main class
│   │   ├── ewc.py                 # Elastic Weight Consolidation
│   │   ├── symbolic_anchor.py     # Immutable symbolic fact store
│   │   ├── memory_buffer.py       # Episodic memory replay
│   │   └── scheduler.py           # Decides when/what to consolidate
│   │
│   ├── api/                       # Developer-facing API (the "PyTorch feel")
│   │   ├── __init__.py
│   │   ├── nesy_model.py          # nesy.reason(), nesy.learn(), nesy.explain()
│   │   ├── pipeline.py            # NeSyPipeline: end-to-end composable
│   │   ├── decorators.py          # @nesy.symbolic_rule, @nesy.requires_proof
│   │   └── context.py             # Context manager: with nesy.strict_mode():
│   │
│   ├── deployment/                # Edge + production deployment
│   │   ├── __init__.py
│   │   ├── optimizer.py           # Prune/quantize guided by symbolic importanc
│   │   ├── npu.py                 # NPU-specific inference path
│   │   ├── lite.py                # NeSy-Lite: reduced graph for mobile
│   │   ├── exporter.py            # Export to ONNX, TFLite, CoreML
│   │   └── server/
│   │       ├── app.py             # FastAPI inference server
│   │       ├── routes.py          # /reason, /learn, /explain endpoints
│   │       └── middleware.py      # Auth, rate limiting, logging
│   │
│   └── integrations/              # Plug into existing ecosystems
│       ├── __init__.py
│       ├── huggingface.py         # Wrap any HuggingFace model with NeSy
│       ├── langchain.py           # LangChain tool + chain integration
│       ├── pytorch_lightning.py   # Lightning training loop
│       └── openai.py              # OpenAI API response post-processing
```

```
│
├── configs/                          # Domain-specific configs
│   ├── default.yaml
│   ├── medical.yaml
│   ├── legal.yaml
│   └── edge.yaml
│
├── tests/                            # Full test suite
│   ├── conftest.py                   # Shared fixtures
│   ├── unit/
│   │   ├── test_symbolic_engine.py
│   │   ├── test_logic.py
│   │   ├── test_grounding.py
│   │   ├── test_metacognition.py
│   │   ├── test_confidence.py
│   │   └── test_continual.py
│   ├── integration/
│   │   ├── test_pipeline.py          # End-to-end pipeline tests
│   │   ├── test_huggingface.py
│   │   └── test_server.py
│   └── benchmarks/
│       ├── test_inference_speed.py
│       ├── test_memory_usage.py
│       └── test_npu_latency.py
│
├── examples/                         # Working code examples
│   ├── basic_reasoning.py
│   ├── medical_diagnosis.py
│   ├── continual_learning.py
│   ├── edge_deployment.py
│   └── notebooks/
│       ├── 01_quickstart.ipynb
│       ├── 02_symbolic_rules.ipynb
│       └── 03_metacognition.ipynb
│
├── docs/                             # Documentation source
│   ├── index.md
│   ├── quickstart.md
│   ├── architecture.md
│   ├── api_reference/
│   │   ├── nesy_model.md
│   │   ├── symbolic_engine.md
│   │   └── metacognition.md
│   └── guides/
│       ├── adding_rules.md
│       ├── edge_deployment.md
│       └── custom_backbone.md
```

```
│
├── scripts/                          # Dev and ops scripts
│   ├── build_concept_graph.py        # Build domain concept graph from corpus
│   ├── evaluate_benchmarks.py        # Run full benchmark suite
│   ├── export_model.py               # Export trained model
│   └── generate_docs.py
│
├── docker/
│   ├── Dockerfile                    # Production image
│   ├── Dockerfile.dev                # Development image
│   └── docker-compose.yml            # Full stack: server + dependencies
│
└── .github/
    ├── workflows/
    │   ├── ci.yml                    # Run tests on every PR
    │   ├── release.yml               # Publish to PyPI on tag
    │   └── benchmark.yml             # Performance regression checks
    └── ISSUE_TEMPLATE/
        ├── bug_report.md
        └── feature_request.md
```

Key Design Decisions

Why `nesy/core/types.py` exists separately: Every module imports from `core/types.py`. No circular imports. Clean dependency graph.

Why `symbolic/` and `neural/` are siblings, not parent/child: Either can work without the other. A pure symbolic pipeline is valid. A pure neural pipeline is valid. The `api/bridge.py` connects them — but does not own either.

Why `metacognition/` is its own top-level module: It is the primary novelty. It should be importable independently: `from nesy.metacognition import MetaCognitionMonitor`. Teams can adopt it without the full framework.

Why `continual/symbolic_anchor.py` is separate from the neural weights: Symbolic facts are immutable. Neural weights drift. They must never be in the same storage layer.

Why `deployment/` has its own `server/`: The inference server is production code, not a script. It needs its own routes, middleware, and auth — treated as a microservice.

Install Targets

```
pip install nesy-core                    # Core only
pip install nesy-core[torch]             # + PyTorch backbone
pip install nesy-core[server]            # + FastAPI server
pip install nesy-core[edge]              # + ONNX + TFLite export
pip install nesy-core[all]              # Everything
```

First File to Write

Start here: `nesy/core/types.py`

```
from dataclasses import dataclass, field
from typing import List, Dict, Optional
from enum import Enum

class ConfidenceType(Enum):
    FACTUAL = "factual"
    REASONING = "reasoning"
    KNOWLEDGE_BOUNDARY = "knowledge_boundary"

@dataclass
class SymbolicRule:
    id: str
    predicate: str
    constraints: List[str]
    weight: float = 1.0

@dataclass
class ReasoningTrace:
    steps: List[str]
    rules_activated: List[SymbolicRule]
    neural_confidence: float
    symbolic_confidence: float

@dataclass
class NSIOutput:
    answer: str
    confidence: Dict[ConfidenceType, float]
    reasoning_trace: ReasoningTrace
    flags: List[str] = field(default_factory=list)
    status: str = "ok"   # ok | flagged | uncertain
```

Everything else builds on top of these types.