IntCyt v2 Repository Guide

Version: **0.1.0** | Generated: **2026-01-16**

This document is a compact guide to the IntCyt v2 codebase. It is written to be usable both as repository documentation and as a supplementary appendix describing computational reproducibility (benchmark evaluation, ablations, and the thermodynamic ledger/metrics pipeline).

1. Overview

IntCyt v2 pairs a preserved legacy research implementation (for behavioral reproducibility) with modern helper modules that make it easier to run experiments programmatically, collect thermodynamic ledgers, compute derived metrics, and generate figures.

At a high level, learning proceeds via repeated cycles of numerical state adaptation (allostasis/homeostasis) interleaved with structural operations (fission, fusion, and operadic composition), followed by irreversible cleaning steps that enable a thermodynamic accounting of organization versus dissipation.

2. Repository layout

Path	Purpose
legacy/	Original IntCyt modules and scripts (historical behavior).
core/	Modern API wrappers around legacy scripts and streaming simulator.
examples/	Paper-facing evaluation scripts (test recall and ablations).
thermo/	Thermodynamic ledger instrumentation utilities.
metrics/	Entropy/LND/DOE/TUR computations from ledger traces.
viz/	Plotting helpers.
cli.py	Convenience CLI that forwards to legacy scripts and ledger demos.
README.md	Top-level usage guide.
README-paper.md	Paper summary and context.
README-math.md	Mathematical background notes.
new-documentation.md	Text version of the legacy documentation PDF.

3. Installation and quick sanity checks

Recommended: Python 3.10+.

Basic install (editable mode):

```
python -m pip install --editable .
```

Optional extras (plotting + dev tools):

```
python -m pip install --editable '.[viz,dev]'
```

Paper evaluation scripts in examples/ require additional dependencies (PyTorch, torchvision, scikit-learn). If you want to keep the base install lightweight, define an optional dependency group (for example 'eval') in pyproject.toml and install via '.[eval]'.

Sanity checks:

Import and CLI help:

```
python -c "import intcyt; print(intcyt.__name__)" python cli.py --help
```

4. Data

The paper-facing evaluation scripts support two dataset sources: torchvision (auto-download) and local IDX gzip files. If torchvision import fails in a given environment, use --source idx and provide the four gzip files below.

Dataset	IDX gzip files (inside --data-dir)
MNIST	train-images-idx3-ubyte.gz, train-labels-idx1-ubyte.gz, t10k-images-idx3-ubyte.gz, t10k-labels-idx1-ubyte.gz
Fashion-MNIST	train-images-idx3-ubyte.gz, train-labels-idx1-ubyte.gz, t10k-images-idx3-ubyte.gz, t10k-labels-idx1-ubyte.gz

5. Reproducing paper-facing computational experiments

The examples/ scripts implement the classification benchmark evaluation protocol used for the paper tables: IntCyt is trained without labels, then a supervised readout is constructed by mapping learned compartments/prototypes to labels via majority vote on the training set, and performance is evaluated on the held-out test set.

5.1 Test recall benchmark (K = 10)

In the paper-facing benchmarks, K denotes the number of initial terminal prototypes/compartments used by the evaluator. In the provided scripts this corresponds to setting --arity 10 with --levels 0 (default).

MNIST (5 runs):

```
python examples/evaluate_test_recall.py --dataset mnist --arity 10 --levels 0 --runs 5
```

Fashion-MNIST (5 runs):

```
python examples/evaluate_test_recall.py --dataset fashion-mnist --arity 10 --levels 0
--runs 5
```

5.2 Residual ablation (K = 10)

Residual ablation disables residual accumulation during training while keeping the rest of the learning dynamics unchanged. Conceptually, this removes the residual ledger signal from the learning process while leaving numerical updates and (when enabled) structural operations active.

```
python examples/ablation_residuals.py --dataset both --arity 10 --levels 0 --runs 5
```

5.3 Structural ablations (optional)

If you include structural ablations, implement them by disabling scheduled fission/fusion/composition events during training while keeping the numerical update loop intact. This isolates the contribution of structural reorganization to specialization. A minimal pattern is to set the fission/fusion/compose event lists in the events schedule to empty lists for the ablation condition.

6. Legacy scripts (historical implementation)

The legacy implementation lives under legacy/ and remains runnable as originally designed. It contains standalone scripts for challenge generation, training, and DREAM3 analysis. See legacy/README.md and new-documentation.md for full historical details.

6.1 Challenge generator

Examples:

```
python legacy/software_implementation/challenge.py MNIST 20 -right python
legacy/software_implementation/challenge.py fashion-MNIST 20 -left-noisy 2 5
```

This writes challenge snapshots to result-load/load_initial.gz.

6.2 Legacy training loop

```
python legacy/software_implementation/main.py MNIST --data-dir data --result-load
result-load/load_initial.gz -iterations 10000
```

Typical outputs: save_roo.gz, save_org.gz, save_tre.gz, and c_info.txt. These are legacy-format artifacts used by the original workflow.

6.3 DREAM3 analysis helpers

```
python legacy/software_implementation/data_processing.py training python
legacy/software_implementation/data_processing.py figure1 python
legacy/software_implementation/data_processing.py method
```

7. Thermodynamic ledger and metrics pipeline

The thermo/ module provides opt-in instrumentation that logs logically irreversible events (e.g., cytosol resets and organelle discards) and converts them to bit lower bounds. Derived metrics are provided in metrics/ (entropy, lnd, doe, tur).

Minimal programmatic pattern:

```
from thermo import Ledger, step_with_ledger from legacy.celloperad.cl_ope import Operad #
build operad + initial SuperCell ledger = Ledger() for t, payload in enumerate(stream):
step_with_ledger(operad, supercell, index=t, events=events, vector=payload['vector'],
gamma=payload['gamma'], filtering=[1.5,1.5,10], ledger=ledger)
```

CLI demo:

```
python cli.py ledger-info python cli.py ledger-run --config demo/config.json --stream
demo/stream.jsonl --output demo/ledger.json
```

8. API entry points

Programmatic wrappers live in core/api.py (run_challenge, run_training, run_data_processing, run_ledger_stream). A minimal streaming Simulator is provided in core/simulator.py.

Core modules:

Module	Key functionality
core/api.py	Call legacy scripts programmatically; JSON ledger stream runner.
core/simulator.py	Streaming step() wrapper with optional ledger logging.
thermo/runtime.py	Ledger, snapshot(), and step_with_ledger() instrumentation.
metrics/*	Entropy + dissipation + efficiency metrics on ledger traces.

9. Recommended repository improvements

- Add a .gitignore (exclude .data/, data/, result-load/, __pycache__/, *.gz outputs, and OS/editor caches).
- Remove accidental large artifacts such as .clang-cache from the repository history and ignore them going forward.
- Add an 'eval' optional dependency group (torch, torchvision, scikit-learn) so paper scripts can be installed reproducibly.
- Add smoke tests under tests/ (import tests; 1-2 iteration training step; script CLI help).
- Add LICENSE and CITATION.cff for reuse and proper attribution.
- Document paper reproduction commands in README.md (Table S2 benchmark and ablations).

For a deeper conceptual description of the computational framework, see Intcyt_v2_computational_framework_summary.pdf and the legacy documentation under legacy/documentation-compintcyt.pdf.