

# CLAUDE.md - ET Library Architecture (AI-Optimized)

## Complete Context, Minimum Tokens

**Purpose:** Full understanding of Exception Theory Python Library v3.0 architecture for adding code in batches of 10.

## CORE AXIOM

```
ET = P°D°T where:
P = Point (infinite substrate, cardinality Ω)
D = Descriptor (finite constraints, cardinality n)
T = Traverser (indeterminate agency, |T| = [0/0])
° = Binding operator (interaction)
```

### Base Constants:

- MANIFOLD\_SYMMETRY = 12 (3 primitives × 4 logic states)
- BASE\_VARIANCE = 1/12
- All math derives from these

## FILE STRUCTURE (6,402 lines total)

```
exception_theory/
├── core/
│   ├── constants.py      (210L) - ALL constants
│   ├── mathematics.py    (908L) - ETMathV2 class (52 static methods)
│   └── primitives.py      (289L) - P, D, T, E base classes
├── classes/
│   ├── batch1.py         (848L) - 8 classes (Computational ET)
│   ├── batch2.py         (859L) - 8 classes (Manifold Architectures)
│   └── batch3.py         (931L) - 10 classes (Distributed Consciousness)
├── engine/
│   └── sovereign.py       (1879L) - ETSovereign (101 methods)
└── utils/
    ├── calibration.py     (172L) - ETBeaconField, ETContainerTraverser
    └── logging.py         (94L) - Logger config
```

## DECISION TREE: WHERE CODE GOES

INPUT: 10 items (equations/classes/code)

- Is it a MATHEMATICAL OPERATION?
  - └ YES → Add to ETMathV2 in core/mathematics.py  
Pattern: @staticmethod, derives from P°D°T
- Is it a NEW CONSTANT?
  - └ YES → Add to core/constants.py  
Group by: BASE/BATCH1/BATCH2/BATCH3/PLATFORM
- Is it a FEATURE CLASS?
  - └ Computational/Algorithmic? → classes/batch1.py
  - └ Data Structure/Architecture? → classes/batch2.py
  - └ Distributed/Network? → classes/batch3.py
- Is it an INTEGRATION METHOD?
  - └ YES → Add to ETSovereign in engine/sovereign.py  
Pattern: create\_X(), get\_X(), direct\_X\_operation()
- Is it a UTILITY/HELPER?
  - └ YES → Add to utils/calibration.py or utils/logging.py

## MODULE RULES (STRICT)

### core/constants.py

```
# Pattern: SCREAMING_SNAKE_CASE
# Groups: BASE → BATCH1 → BATCH2 → BATCH3 → PLATFORM
# NO imports from other ET modules
# NO computation (constants only)
```

### core/mathematics.py

```
class ETMathV2:
    @staticmethod
    def operation_name(params) -> result:
        """Batch X, Eq Y: Description

        ET Math: [formula in P°D°T notation]
        """
        # MUST derive from P, D, T, E primitives
        # NO external algorithms (pure ET derivation)
        # Use constants from core.constants
```

### core/primitives.py

```
# Point, Descriptor, Traverser, Exception classes
# Foundation classes - RARELY modified
# Only edit when adding fundamental ET concepts
```

## classes/batchN.py

```
# Pattern per class:
from ..core.constants import (relevant_constants)
from ..core.mathematics import ETMathV2

class FeatureName:
    """Batch N, Eq X: Description

    ET Math: [formula]
    """
    def __init__(self, params):
        # Use constants, call ETMathV2 methods

    def operations(self):
        # Implement using ET math
```

## engine/sovereign.py

```
class ETSovereign:
    def __init__(self):
        # Subsystem registries:
        self._feature_registry = {} # Add new ones here

    # Integration pattern (3 methods per feature):
    def create_feature(self, name, params):
        """Batch N, Eq X: Create feature."""
        obj = FeatureClass(params)
        self._feature_registry[name] = obj
        return obj

    def get_feature(self, name):
        """Get registered feature."""
        return self._feature_registry.get(name)

    def direct_operation(self, params): # Optional
        """Batch N, Eq X: Direct ETMathV2 access."""
        return ETMathV2.operation(params)

    def close(self):
        # Add cleanup:
        self._feature_registry.clear()
```

---

# BATCH INTEGRATION PATTERN

---

## Adding 10 New Items to Batch 4:

### Step 1: Identify item types

```
Item 1-3: Math operations → ETMathV2
Item 4-7: Feature classes → classes/batch4.py (NEW)
```

```
Item 8-9: Constants → constants.py  
Item 10: Integration → sovereign.py
```

## Step 2: Add constants (if needed)

```
# core/constants.py (append to BATCH 4 section)  
BATCH4_PARAM_X = value  
BATCH4_THRESHOLD_Y = value
```

## Step 3: Add math operations

```
# core/mathematics.py (inside ETMathV2)  
@staticmethod  
def batch4_operation(x, y):  
    """Batch 4, Eq 31: Operation description  
  
    ET Math: result = f(P,D,T)  
    """  
    # Pure ET derivation  
    return result
```

## Step 4: Create batch4.py

```
# classes/batch4.py (NEW FILE)  
"""  
Exception Theory Batch 4 Classes  
[Theme description]  
"""  
  
import deps  
from ..core.constants import BATCH4_*  
from ..core.mathematics import ETMathV2  
  
class Feature1:  
    """Batch 4, Eq 31: Description"""  
    # Implementation using ETMathV2  
  
# ... 4 total classes for items 4-7
```

## Step 5: Update classes/init.py

```
# Add imports:  
from .batch4 import Feature1, Feature2, Feature3, Feature4
```

## Step 6: Add to sovereign.py init

```
# In ETSovereign.__init__:  
self._batch4_features = {} # Add registry
```

## Step 7: Add integration methods to sovereign.py

```
# Add before # CLEANUP section:
# =====
# v2.4: BATCH 4 INTEGRATIONS
# =====

def create_feature1(self, name, params):
    """Batch 4, Eq 31: Create feature1."""
    obj = Feature1(params)
    self._batch4_features[name] = obj
    return obj

def get_feature1(self, name):
    return self._batch4_features.get(name)

# ... repeat for all features
```

## Step 8: Update sovereign.py close()

```
# In close() method:
self._batch4_features.clear() # Add cleanup
```

## Step 9: Update sovereign.py docstring

```
class ETSovereign:
    """
    ET Sovereign v2.4 - [updated description]

    NEW IN v2.4:
    - Feature1: Description
    - Feature2: Description
    """
```

## Step 10: Update main init.py

```
# exception_theory/__init__.py
# Imports automatically cascade from classes/__init__.py
# Usually no change needed
```

---

## SIZE LIMITS

```
constants.py:    Keep < 300 lines (currently 210)
mathematics.py: Keep < 1200 lines (currently 908)
primitives.py:  Keep < 350 lines (currently 289)
batch1.py:       Keep < 1000 lines (currently 848)
batch2.py:       Keep < 1000 lines (currently 859)
batch3.py:       Keep < 1000 lines (currently 931)
```

```
batch4.py:      Target < 900 lines (NEW)
sovereign.py:    Keep < 2500 lines (currently 1879)
```

**If approaching limits:** Create batch5.py instead of expanding batch4.py

## INTEGRATION CHECKLIST (10 items)

- ☐ 1. Identify types (math/class/constant/integration)
- ☐ 2. Add constants to constants.py (if needed)
- ☐ 3. Add math to ETMathV2 (if needed)
- ☐ 4. Create/update batchN.py with classes
- ☐ 5. Update classes/\_\_init\_\_.py imports
- ☐ 6. Add registries to ETSovereign.\_\_init\_\_
- ☐ 7. Add create\_X/get\_X methods to sovereign.py
- ☐ 8. Add direct operation methods (if applicable)
- ☐ 9. Add cleanup to sovereign.close()
- ☐ 10. Update version/docstrings

## CODE PATTERNS (COMPRESSED)

### ETMathV2 Method Template

```
@staticmethod
def operation(p_input, d_constraint):
    """Batch N, Eq X: Brief
    ET Math: result = P°D formula"""
    # Derive from primitives
    return result
```

### Feature Class Template

```
class Feature:
    """Batch N, Eq X: Brief
    ET Math: formula"""

    def __init__(self, params):
        self.data = params
        self._hash = ETMathV2.hash_op(params)

    def operation(self):
        return ETMathV2.operation(self.data)
```

### Integration Template

```
# In sovereign.py:
def create_X(self, name, param):
```

```

obj = XClass(param)
self._x_registry[name] = obj
return obj

def get_X(self, name):
    return self._x_registry.get(name)

```

## DEPENDENCY FLOW

```

constants.py (NO deps)
  ↓
mathematics.py (imports constants)
  ↓
primitives.py (imports constants, mathematics)
  ↓
batch1/2/3/N.py (imports constants, mathematics)
  ↓
sovereign.py (imports ALL)
  ↓
__init__.py (exports ALL)

```

**RULE:** Never import from sibling or child modules, only parents.

## CURRENT STATE SNAPSHOT

### Batches Implemented:

- Batch 1: Computational ET (8 classes, Eq 1-10)
- Batch 2: Manifold Architectures (8 classes, Eq 11-20)
- Batch 3: Distributed Consciousness (10 classes, Eq 21-30)

### Next Available:

- Batch 4: Eq 31-40
- Batch 5: Eq 41-50
- etc.

**ETMathV2 Methods:** 52

**ETsouverign Methods:** 101

**Total Classes:** 26

## ADDING CODE WORKFLOW

**Input:** 10 items (equations/text/code)

1. **Parse:** Identify equation numbers, concepts, types

2. **Classify:** Math (ETMathV2) vs Feature (Class) vs Integration (Sovereign)
3. **Constants:** Extract any new constants needed
4. **Math First:** Add to ETMathV2 if mathematical operations
5. **Classes:** Create/update batchN.py with feature classes
6. **Integrate:** Add to ETSovereign (create/get/direct methods)
7. **Cleanup:** Update close(), docstrings, version
8. **Verify:** Check imports, registries, dependencies

**Output:** Staged files ready for integration

---

## FORBIDDEN PATTERNS

---

- ✗ Circular imports (batch1 importing batch2)
  - ✗ External algorithms (must derive from ET)
  - ✗ Hardcoded values (use constants.py)
  - ✗ Missing ET Math docstrings
  - ✗ Skipping integration methods
  - ✗ Forgetting cleanup in close()
  - ✗ Breaking dependency flow
  - ✗ Exceeding size limits
  - ✗ Placeholders/TODOs
  - ✗ Missing equation numbers
- 

## RESPONSE PROTOCOL

---

When given 10 items:

1. Acknowledge: "Received 10 items for Batch N"
2. Analyze: List each item with type classification
3. Report Redundancy: "Item X already exists in Y"
4. Plan: "Will add to: constants(2), ETMathV2(3), batchN(4), sovereign(1)"
5. Implement: Provide COMPLETE code (no truncation)
6. Stage: Split into manageable parts if needed
7. Verify: "Added X constants, Y methods, Z classes"
8. Summary: Version change, line counts, method counts

**Never:** Assume, truncate, skip, or placeholder.

**Always:** Complete, derive, integrate, verify.

---

## VERSION TRACKING

---

constants.py:	v3.0 (210L, ~80 constants)
mathematics.py:	v3.0 (908L, 52 methods)
primitives.py:	v3.0 (289L, 4 classes)

```
batch1.py:      v2.1 (848L, 8 classes, Eq 1-10)
batch2.py:      v2.2 (859L, 8 classes, Eq 11-20)
batch3.py:      v2.3 (931L, 10 classes, Eq 21-30)
sovereign.py:   v2.3 (1879L, 101 methods, Batches 1-3)
```

**On Update:** Increment minor version (v2.3 → v2.4)

---

## COMPACT REFERENCE

---

**Module Purpose** (1 line each):

- constants: All static values, no computation
- mathematics: Pure functions, ET-derived operations
- primitives: P/D/T/E base classes, foundation
- batchN: Feature implementations using ETMathV2
- sovereign: Unified API, creates/manages instances
- calibration: Memory field tracking, displacement
- logging: Logger configuration

**Size Strategy:**

- Keep modules < 1000L each
- Create new batch instead of expanding
- sovereign can grow to 2500L

**Integration:** 3-method pattern (create/get/direct)

**Testing:** Add to tests/test\_basic.py for each batch

---

## END

---

**Total:** 6,402 lines, 26 classes, 101 sovereign methods, 52 math methods

**Ready:** Batch 4 (Eq 31-40)

**Pattern:** Established and consistent

**Gaps:** ZERO

This document contains complete architecture context in minimum tokens.