TNorms

**✅ Summary Table of Design Principles**

| **Category** | **Name** | **Usage & Where Applied** |
| --- | --- | --- |
| **Design Pattern** | **Factory Method** | TNorm.create(name, \*\*kwargs) dynamically creates TNorm objects from registered classes based on name/alias. |
| **Design Pattern** | **Registry Pattern** | TNorm.\_registry and register() decorator maintain a central registry of all available TNorm subclasses and aliases. |
| **Design Pattern** | **Template Method** | Abstract base class TNorm defines method signatures like \_\_call\_\_() and reduce(), which are implemented differently by subclasses. |
| **Design Pattern** | **Decorator (Class Registration)** | @TNorm.register('name', ...) dynamically registers subclasses with aliases using a class decorator. |
| **Design Pattern** | **Strategy Pattern** | Each TNorm subclass (e.g., MinTNorm, YagerTNorm) implements its own strategy for computing the T-norm. |
| **Design Pattern** | **Adapter (Serialization)** | to\_dict() / from\_dict() methods act as an adapter for converting between object and dictionary representation for serialization. |
| **Architecture** | **Pluggable Architecture** | New T-norms can be added without modifying core logic—just register via decorator, enabling extensibility. |
| **Clean Code** | **Single Responsibility Principle** | Each class and method does one thing (e.g., YagerTNorm handles only the Yager logic; validate\_params handles only validation). |
| **Clean Code** | **Open/Closed Principle** | Framework is open for extension (add new TNorms) but closed for modification (no need to change base class). |
| **Clean Code** | **DRY (Don't Repeat Yourself)** | Common logic (e.g. alias filtering, validation pattern) is centralized and reused via helper methods. |
| **Clean Code** | **Encapsulation & Abstraction** | Abstract base class hides implementation details from users and enforces a consistent API. |
| **Clean Code** | **Liskov Substitution Principle** | All TNorm subclasses can be used wherever a TNorm object is expected without breaking functionality. |
| **Clean Code** | **Docstring Documentation** | Doxygen-style docstrings enable IDE and tool-friendly introspection and maintainability. |
| **Design Support** | **Reflection/Introspection** | inspect.signature() in \_filter\_args() allows dynamic and safe instantiation of subclasses with correct parameters. |