

Universal Stochastic Predictor Bootstrap Infrastructure

Implementation Team

February 19, 2026

Índice

1 Executive Summary	2
1.1 Tag Information	2
2 Architecture: Clean 5-Layer Design	3
2.1 Architectural Constraints	3
2.2 Clean Architecture Compliance	4
3 Development Environment Setup	5
3.1 Python Ecosystem	5
3.2 Project Structure Initialization	5
4 Language Policy Enforcement	6
4.1 100% English in Code	6
5 Golden Master: Dependency Pinning	7
5.1 Frozen Requirements	7
5.2 Rationale	7
6 Configuration Management	8
7 Git Workflow and Versioning	9
7.1 Branch Strategy	9
7.2 Tag Naming Convention	9
8 Pre-Commit Quality Assurance	10
8.1 Error Types to Monitor	10
9 Documentation Structure	11
10 Initialization Checklist	12
10.1 Directory Structure	12
10.2 Configuration Files	12
10.3 Documentation	12
10.4 Version Control	12
11 Next Phase (Phase 1: Foundations)	13

Capítulo 1

Executive Summary

This document records the Bootstrap phase (Tag: `impl/v2.0.0-Bootstrap`) of the Universal Stochastic Predictor implementation. Bootstrap establishes the foundational 5-layer Clean Architecture structure and development environment.

1.1 Tag Information

Tag	<code>impl/v2.0.0-Bootstrap</code>
Commit	<code>85abb8c</code>
Branch	<code>implementation/base-jax</code>
Date	February 18, 2026

Capítulo 2

Architecture: Clean 5-Layer Design

2.1 Architectural Constraints

Per Predictor_Estocastico_Python.tex §2.1, the system enforces a strict 5-layer Clean Architecture:

```
stochastic_predictor/
    api/                                Layer 1: External Contracts
        types.py
        prng.py
        validation.py
        schemas.py
        config.py
        __init__.py

    core/                                 Layer 2: Orchestration Logic
        jko.py
        wasserstein.py
        __init__.py

    kernels/                               Layer 3: Stateless Kernels (A, B, C, D)
        base.py
        kernel_a.py
        kernel_b.py
        kernel_c.py
        kernel_d.py
        __init__.py

    io/                                    Layer 4: Snapshots & Streaming
        snapshots.py
        __init__.py

tests/                                  Layer 5: Test Infrastructure (scaffold)
    __init__.py
    [test files reserved for v3.x.x]
```

2.2 Clean Architecture Compliance

Each layer has strict responsibilities:

Layer	Responsibility	Prohibited
api/	External contracts, validation, configuration	Business logic, stateful operations
core/	Orchestration, decision logic, fusion algorithms	Direct device operations, I/O
kernels/	Pure, stateless JAX functions (JIT-compilable)	Configuration, file I/O, randomness generation
io/	Atomic snapshots, stream sanitization	Prediction logic, kernel computations
tests/	Test infrastructure scaffold (reserved for v3.x.x)	Implementation logic

Cuadro 2.1: Clean Architecture Layer Boundaries

Capítulo 3

Development Environment Setup

3.1 Python Ecosystem

Bootstrap establishes the Golden Master dependency pinning:

- Python 3.10.12
- JAX 0.4.20 (with XLA backend)
- Equinox 0.11.2 (neural networks)
- Diffraex 0.4.1 (differential equations)
- OTT-JAX 0.4.5 (optimal transport)
- Signax 0.1.4 (signatures/rough paths)
- PyWavelets 1.4.1 (wavelet analysis)

Critical Rule: All versions use strict equality operator (==). No >=, no pip install -U.

3.2 Project Structure Initialization

Bootstrap creates the 5-layer directory structure with minimal `__init__.py` files for module discovery.

```
1 # Create layer directories
2 mkdir -p stochastic_predictor/{api,core,kernels,io}
3 touch stochastic_predictor/{__init__.py,api/__init__.py,core/__init__.py,kernels/__init__
   .py,io/__init__.py}
4
5 # Create tests structure (scaffold only, actual tests in v3.x.x)
6 mkdir -p tests
7 touch tests/__init__.py
```

Capítulo 4

Language Policy Enforcement

4.1 100% English in Code

Bootstrap establishes the foundational language policy:

All code files MUST be 100% English:

- File names, class names, variable names, method names
- Docstrings (triple quotes)
- Inline comments (#)
- Log messages and error messages
- Configuration files (TOML, YAML, JSON)
- Requirements files and dependencies metadata
- README files and inline documentation

Spanish reserved for:

- LaTeX specification documents (.tex, .pdf)
- Chat responses and external communication

Rationale: Bit-exact reproducibility across global development environments requires linguistic homogeneity in all executable and configuration artifacts.

Capítulo 5

Golden Master: Dependency Pinning

5.1 Frozen Requirements

`requirements.txt` established with strict `==` operators:

```
jax==0.4.20
jaxlib==0.4.20
equinox==0.11.2
diffraex==0.4.1
jaxtyping==0.2.25
ott-jax==0.4.5
signax==0.1.4
PyWavelets==1.4.1
numpy==1.24.0
scipy==1.10.0
pandas==2.0.0
```

5.2 Rationale

Per `Predictor_Estocastico_Python.tex §1`:

- **Bit-exactness:** Numerical results must be reproducible
- **XLA caching:** JIT compilation depends on exact library versions
- **JAX API stability:** Breaking changes in minor versions
- **Research integrity:** Published results must be reproducible

Capítulo 6

Configuration Management

Bootstrap establishes config.toml for centralized parameter management:

```
1 [core]
2 jax_platforms = "cpu"
3 jax_default_dtype = "float32"
4
5 [orchestration]
6 cusum_grace_period = 20
7 cusum_threshold = 5.0
8 entropy_window = 100
9 sinkhorn_epsilon_0 = 0.1
10 sinkhorn_alpha = 0.5
11
12 [kernels]
13 stiffness_low = 100
14 stiffness_high = 1000
15 sde_dt = 0.01
16
17 [io]
18 market_feed_timeout = 30
19 market_feed_max_retries = 3
```

Capítulo 7

Git Workflow and Versioning

7.1 Branch Strategy

- `main`: Specification branch (locked at `spec/v1.0.0`)
- `implementation/base-jax`: Active development branch (incremental versioning)

7.2 Tag Naming Convention

Pattern	Usage
<code>spec/v1.x.x</code>	Specification versions (immutable)
<code>impl/v2.x.x-<PhaseName></code>	Implementation phases (incremental)

Bootstrap tag: `impl/v2.0.0-Bootstrap`

Capítulo 8

Pre-Commit Quality Assurance

Bootstrap establishes mandatory quality gates:

1. **Make changes** in working directory
2. **ALWAYS run `get_errors()`** to check for syntax/type errors
3. **If errors found:** Fix all errors BEFORE staging
4. **Only after** errors cleared:
 - `git add <files>`
 - `git commit -m "<meaningful message>"`
 - `git push origin <branch>`

8.1 Error Types to Monitor

- Markdown: MD060 (table formatting), MD036 (heading punctuation)
- LaTeX: Unicode incompatibility in verbatim blocks
- Python: Type hints, import statements, syntax errors
- YAML/TOML: Indentation, key format, string escaping

Capítulo 9

Documentation Structure

Bootstrap establishes doc/ hierarchy:

```
doc/
    README.md                         Documentation index
    compile.sh                         LaTeX compilation automation

    latex/
        specification/                Technical specifications (.tex)
            Predictor_Estocastico_Teoria.tex
            Predictor_Estocastico_Python.tex
            ...
        implementation/               Implementation milestone docs
            Implementacion_v2.0.0_Bootstrap.tex
            Implementacion_v2.0.1_Phase1_Foundations.tex
            Implementacion_v2.0.2_Phase1_Complete.tex
            [future phases]

    pdf/
        specification/                Compiled PDFs
        implementation/
```

Capítulo 10

Initialization Checklist

10.1 Directory Structure

- `stochastic_predictor/` created with 5-layer structure
- `tests/` directory scaffold (actual tests reserved for v3.x.x)
- All `__init__.py` files created for module discovery
- `doc/` structure established (specification + implementation)

10.2 Configuration Files

- `requirements.txt` with Golden Master versions
- `config.toml` with default parameters
- `pyproject.toml` if needed (project metadata)
- `.gitignore` with standard Python patterns

10.3 Documentation

- `README.md` (root) with project overview
- `doc/README.md` documentation index
- `CONTRIBUTING.md` guidelines
- `LICENSE` (MIT)

10.4 Version Control

- Git repository initialized on both `main` and `implementation/base-jax`
- Bootstrap commit tagged as `impl/v2.0.0-Bootstrap`
- Specification frozen at `spec/v1.0.0`
- Clean git history with meaningful commits

Capítulo 11

Next Phase (Phase 1: Foundations)

Bootstrap establishes the foundation. Phase 1 will implement:

- `api/types.py`: Immutable dataclasses (PredictorConfig, MarketObservation, PredictionResult)
- `api/prng.py`: JAX PRNG management for determinism
- `api/validation.py`: Input/output domain validation
- `api/schemas.py`: Pydantic models for serialization
- `api/config.py`: ConfigManager singleton with config.toml injection

Note: Test infrastructure (including fixtures) is reserved for v3.x.x.

All Phase 1 code will be 100% English, follow Clean Architecture constraints, and pass pre-commit quality gates.