

**Universal Stochastic Predictor**  
**Phase 4: IO Layer**  
**v2.1.0 (Level 4 Autonomy)**

Implementation Team

February 19, 2026

# Contents

<b>1 Phase 4: IO Layer Overview</b>	<b>3</b>
1.1 Tag Information . . . . .	3
1.2 Scope . . . . .	3
1.3 Design Principles . . . . .	3
<b>2 Ingestion and Validation</b>	<b>4</b>
2.1 Implementation Modules . . . . .	4
2.2 Catastrophic Outlier Filter . . . . .	4
2.2.1 Implementation Notes . . . . .	4
2.3 Frozen Signal Alarm . . . . .	4
2.3.1 Recovery Criteria (V-MAJ-6: Frozen Signal Recovery Ratio) . . . . .	5
2.4 Staleness Policy (TTL) . . . . .	6
2.4.1 Implementation Notes . . . . .	6
<b>3 Telemetry Abstraction</b>	<b>7</b>
3.1 TelemetryBuffer Emission . . . . .	7
3.1.1 Implementation Notes . . . . .	7
3.2 No Compute Stalls . . . . .	7
<b>4 Deterministic Logging</b>	<b>8</b>
4.1 Hash-Based Parity Checks . . . . .	8
4.2 Audit Hashes . . . . .	8
<b>5 Snapshot Strategy</b>	<b>9</b>
5.1 Atomic Persistence . . . . .	9
5.2 Binary Serialization . . . . .	9
5.2.1 Implementation Notes . . . . .	9
5.3 Integrity Verification . . . . .	9
5.4 Atomic Write Protocol . . . . .	9
5.4.1 Implementation Notes . . . . .	10
<b>6 Security Policies</b>	<b>11</b>
6.1 Credential Injection . . . . .	11
6.1.1 Implementation Notes . . . . .	11
6.2 Version Control Exclusion . . . . .	11
<b>7 Orchestrator Integration</b>	<b>12</b>
7.1 Ingestion Gate in <code>orchestrate_step()</code> . . . . .	12
7.1.1 Execution Flow . . . . .	12
7.1.2 Flag Semantics . . . . .	13
7.1.3 Early Return on Rejection . . . . .	13
7.2 PRNG Constant . . . . .	13

7.3	64-bit Precision Enforcement . . . . .	13
<b>8</b>	<b>Telemetry Buffer Integration (P2.3)</b>	<b>14</b>
8.1	Motivation . . . . .	14
8.2	Data Model . . . . .	14
8.3	Integration into <code>orchestrate_step()</code> . . . . .	15
8.4	Configuration Parameters . . . . .	15
8.4.1	Configuration Injection Example . . . . .	15
8.5	Thread-Safety Model . . . . .	16
8.6	Backward Compatibility . . . . .	16
8.7	Usage Example . . . . .	16
8.8	Benefits . . . . .	17
8.9	Visualization Dashboard (GAP-6) . . . . .	17
8.9.1	Usage . . . . .	17
8.9.2	Notes . . . . .	17
<b>9</b>	<b>Level 4 Autonomy: Configuration Mutation Safety</b>	<b>18</b>
9.1	Overview . . . . .	18
9.2	V-CRIT-2: Atomic Configuration Mutation Protocol . . . . .	18
9.2.1	Problem Statement . . . . .	18
9.2.2	Requirements . . . . .	19
9.2.3	Implementation . . . . .	19
9.2.4	Usage Example . . . . .	22
9.2.5	Files Modified . . . . .	22
9.2.6	Compliance Impact . . . . .	22
9.3	V-MAJ-4: Configuration Mutation Rate Limiting . . . . .	23
9.3.1	Problem Statement . . . . .	23
9.3.2	Safety Requirements . . . . .	23
9.3.3	Implementation . . . . .	23
9.3.4	Usage Pattern . . . . .	24
9.4	V-MAJ-5: Degradation Detection Auto-Rollback . . . . .	25
9.4.1	Problem Statement . . . . .	25
9.4.2	Safety Requirements . . . . .	25
9.4.3	Implementation . . . . .	25
9.4.4	Usage Pattern . . . . .	27
9.4.5	Audit Log Format . . . . .	28
9.5	V-MAJ-7: Adaptive Telemetry Monitoring . . . . .	28
9.5.1	Implementation Status . . . . .	28
9.5.2	InternalState Extensions . . . . .	28
9.5.3	Adaptive Telemetry Collection . . . . .	28
9.5.4	Integration Pattern . . . . .	29
9.5.5	Metrics Tracked . . . . .	30
9.5.6	Future Extensions . . . . .	30
9.6	Utility Functions . . . . .	31
9.7	Implementation Status . . . . .	31
<b>10</b>	<b>Compliance Checklist</b>	<b>32</b>
<b>11</b>	<b>Phase 4 Summary</b>	<b>33</b>

# Chapter 1

## Phase 4: IO Layer Overview

### 1.1 Tag Information

- **Tag:** impl/v2.1.0
- **Commit:** 49bd30e
- **Status:** Level 4 Autonomy compliance (V-MAJ-4, V-MAJ-5, V-MAJ-7 implemented; GAP-6 complete)

Phase 4 introduces the asynchronous I/O layer for snapshots, streaming, and telemetry export. The primary design goal is to preserve JAX/XLA throughput by decoupling compute from disk or network latency.

### 1.2 Scope

Phase 4 covers:

- **Telemetry Buffering:** Non-blocking emission of telemetry snapshots
- **Deterministic Logging:** Hash-based parity checks for CPU/GPU validation
- **Snapshot Strategy:** Atomic persistence of predictor state
- **Ingestion and Validation:** Input filtering, staleness policy, frozen signal detection
- **Security Enforcement:** Credential injection and secret exclusion
- **IO Modules:** validators, loaders, telemetry, snapshots, credentials

### 1.3 Design Principles

- **No Compute Stalls:** JAX compute threads never block on I/O
- **Determinism:** Logs capture reproducible hashes instead of raw state dumps
- **Security:** No raw signals or secrets in logs
- **Configurability:** Logging intervals and destinations injected via config
- **Integrity:** Snapshots and parity logs are hash-verified

# Chapter 2

# Ingestion and Validation

## 2.1 Implementation Modules

Phase 4 IO introduces the following modules:

- `io/validators.py`: Outlier, frozen signal, and staleness checks
- `io/loaders.py`: Ingestion gate and decision flags
- `io/telemetry.py`: Non-blocking telemetry buffer and parity hashes
- `io/snapshots.py`: Binary snapshots, hash verification, atomic writes
- `io/credentials.py`: Environment-based credential injection helpers

## 2.2 Catastrophic Outlier Filter

Input validation must reject catastrophic outliers when  $|y_t| > 20\sigma$  relative to historical normalization. In this case, the system must preserve inertial state and emit a critical alert without advancing the transport update.

- Reject observation and keep current state unchanged.
- Emit a critical alert for audit visibility.
- Do not update JKO/Sinkhorn weights for the rejected step.

### 2.2.1 Implementation Notes

Outlier detection is implemented as a pure function with configuration-driven thresholds. The ingestion gate returns a decision object that preserves inertial state when an outlier is detected.

## 2.3 Frozen Signal Alarm

If the exact same value is observed for  $N_{freeze} \geq 5$  consecutive steps, emit a `FrozenSignalAlarmEvent`. This invalidates the multifractal spectrum and requires:

- Freeze the topological branch (Kernel D).
- Switch to degraded inference mode.
- Continue monitoring until signal variation resumes.

### 2.3.1 Recovery Criteria (V-MAJ-6: Frozen Signal Recovery Ratio)

The frozen signal lock is released when variance recovers above a configurable ratio of historical variance for a configurable number of consecutive steps.

#### Algorithm

$$\text{recovered} = \text{detect\_frozen\_recovery}(\text{variance\_history}, \text{historical\_var}, \rho, n_c) \quad (2.1)$$

where:

- `variance_history`: Recent residual variances
- `historical_var`: Baseline variance reference
- $\rho = \text{config.frozen\_signal\_recovery\_ratio}$  (default: 0.1): Recovery threshold multiplier
- $n_c = \text{config.frozen\_signal\_recovery\_steps}$  (default: 2): Confirmation window

Recovery is confirmed when:

$$\text{variance}_t > \rho \cdot \text{historical\_var} \quad \text{for } n_c \text{ consecutive steps} \quad (2.2)$$

#### Implementation

```

1 # In evaluate_ingestion():
2 if frozen:
3     residual_window = np.asarray(state.residual_window, dtype=np.float64)
4     historical_variance = float(np.var(residual_window)) if residual_window.size > 1 else
5         0.0
6     recent_window = residual_window[-config.frozen_signal_recovery_steps:]
7     recent_variance = float(np.var(recent_window)) if recent_window.size > 1 else 0.0
8     variance_history = [recent_variance] * config.frozen_signal_recovery_steps
9     in_recovery = detect_frozen_recovery(
10         variance_history=variance_history,
11         historical_variance=historical_variance,
12         ratio_threshold=config.frozen_signal_recovery_ratio, # V-MAJ-6: Use parameter
13         consecutive_steps=config.frozen_signal_recovery_steps
14     )
15     if in_recovery:
16         frozen = False # Lift the frozen signal flag

```

#### Configuration Parameters

From PredictorConfig:

Parameter	Default	Purpose
<code>frozen_signal_min_steps</code>	5	Consecutive equal values to trigger alarm
<code>frozen_signal_recovery_ratio</code>	0.1	Variance ratio threshold for recovery (10% of baseline)
<code>frozen_signal_recovery_steps</code>	2	Confirmation window for recovery

Table 2.1: V-MAJ-6 Frozen Signal Recovery Configuration

## Benefits

- **Automatic Recovery:** No manual intervention needed when signal variance improves
- **Hysteresis:** Recovery threshold ( $\rho = 0.1$ ) is more lenient than typical alarm threshold, preventing oscillation
- **Configuration-Driven:** All parameters injected from config.toml (zero-heuristics policy)
- **State Preservation:** Maintains frozen flag during low-variance periods, automatically lifts when variance returns
- **Signal Quality Supervision:** Enables secondary observability on signal quality degradation patterns

## 2.4 Staleness Policy (TTL)

Every observation must carry a timestamp for TTL evaluation. If the target delay exceeds  $\Delta_{max}$ , the JKO update must be suspended immediately.

- Compute staleness as  $\Delta_t = t_{now} - t_{obs}$ .
- If  $\Delta_t > \Delta_{max}$ , skip the transport update.
- Preserve state and record a staleness warning event.

### 2.4.1 Implementation Notes

Staleness is computed as the difference between current time and observation timestamp. The ingestion decision flags a suspended JKO update when the TTL is exceeded. The orchestrator also applies a secondary staleness check using  $\Delta t = t_{obs} - t_{last\_update}$  (`timestamp_ns` minus `state.last_update_ns`) to trigger degraded mode on large inter-sample gaps.

# Chapter 3

## Telemetry Abstraction

### 3.1 TelemetryBuffer Emission

The JKO orchestrator should emit a `TelemetryBuffer` at the end of each step. This buffer is consumed by a dedicated process outside the JAX execution thread.

- The buffer contains summary metrics (prediction, weights, free energy, diagnostics, regime flags).
- The compute path only enqueues the buffer and continues.
- The consumer is responsible for serialization and persistence.

#### 3.1.1 Implementation Notes

The telemetry buffer is a bounded, thread-safe queue. Buffer capacity is explicitly injected from `PredictorConfig.telemetry_buffer_capacity` to eliminate implicit defaults (zero-heuristics policy). Parity hashes are emitted on a configurable interval and derived from canonical float64 serialization.

```
1 # Instantiation pattern (capacity injected from config)
2 buffer = TelemetryBuffer(capacity=config.telemetry_buffer_capacity)
```

### 3.2 No Compute Stalls

JAX compute threads must never block on I/O. Telemetry buffers must be non-blocking and consumed by a separate process or thread outside the JAX execution path.

# Chapter 4

## Deterministic Logging

### 4.1 Hash-Based Parity Checks

For hardware parity audits, the logger records SHA-256 hashes of the weight vector  $\rho$  and the OT cost at configurable intervals. This permits CPU/GPU parity validation without dumping VRAM data.

- Hash interval configured per deployment.
- Hashes derived from canonical float64 serialization.
- Logs are append-only and immutable.

### 4.2 Audit Hashes

Parity audits must log SHA-256 hashes of  $\rho$  and OT cost at configured intervals. Hash input must be derived from canonical float64 serialization to ensure reproducibility across CPU and GPU.

# Chapter 5

## Snapshot Strategy

### 5.1 Atomic Persistence

Snapshots must be persisted atomically to prevent partial writes. The IO layer is responsible for:

- Writing to temporary files and renaming atomically.
- Optional compression configured by policy.
- Coordinating snapshot cadence with telemetry output.

### 5.2 Binary Serialization

Text formats (JSON, XML) are prohibited for critical snapshots due to latency and ambiguity. The configuration allows `snapshot_format` values `msgpack` or `protobuf`, but the current IO implementation supports MessagePack only.

- Encode all fields deterministically.
- Preserve float64 for numerical fidelity.

#### 5.2.1 Implementation Notes

The snapshot serializer currently supports MessagePack only (`snapshot_format = "msgpack"`); protobuf is reserved for a future implementation. Hash verification is performed before state injection.

### 5.3 Integrity Verification

Each snapshot  $\Sigma_t$  must include a hash footer (SHA-256 or CRC32c). The load routine must verify the hash before injecting state into memory.

- Fail closed if hash verification fails.
- Log integrity failures at critical severity.

### 5.4 Atomic Write Protocol

To avoid partial writes, persist snapshots to a temporary file and then atomically rename to the target path. The rename step must be the only visible operation to consumers.

- Use a unique temporary filename per snapshot.
- Ensure the target file is replaced atomically.

#### **5.4.1 Implementation Notes**

Snapshots are written to a unique temporary file and moved into place using atomic rename. Optional fsync ensures persistence across power loss.

# Chapter 6

# Security Policies

## 6.1 Credential Injection

Tokens and API keys must not appear in source code. Credentials must be injected at runtime via environment variables or .env files.

### 6.1.1 Implementation Notes

Credential helpers read from environment variables or .env files and raise explicit errors on missing values.

## 6.2 Version Control Exclusion

The repository must exclude .env files and credential directories via .gitignore. Secrets must never be committed.

# Chapter 7

## Orchestrator Integration

### 7.1 Ingestion Gate in `orchestrate_step()`

The core orchestration pipeline now integrates ingestion validation as a pre-kernel gate. The `orchestrate_step()` function signature is extended to accept observation metadata:

```
1 def orchestrate_step(
2     signal: Float[Array, "n"],
3     timestamp_ns: int,
4     state: InternalState,
5     config: PredictorConfig,
6     observation: ProcessState,
7     now_ns: int,
8     telemetry_buffer: Optional[TelemetryBuffer] = None,
9     step_counter: int = 0,
10    mutation_rate_limiter: Optional[MutationRateLimiter] = None,
11    degradation_monitor: Optional[DegradationMonitor] = None,
12 ) -> OrchestrationResult:
13     """Run a single orchestration step with IO ingestion validation."""
```

#### 7.1.1 Execution Flow

The ingestion gate operates as follows:

1. **Input Validation:** Standard signal length and dtype checks.
2. **Ingestion Decision:** Call `evaluate_ingestion()` with current state, observation, and configuration.
3. **Rejection Logic:** If `accept_observation == False`, reject the entire observation without state update (emergency mode).
4. **Degradation Flags:** Apply `suspend_jko_update` and `freeze_kernel_d` flags to control fusion behavior.
5. **Kernel Execution:** Run kernels A-D; if `freeze_kernel_d == True`, mark kernel D output as frozen.
6. **Fusion Selection:** Skip JKO/Sinkhorn if degraded mode or `suspend_jko_update` is set.
7. **State Update:** Only update `InternalState` if observation is accepted.

### 7.1.2 Flag Semantics

The `IngestionDecision` object carries the following flags:

- `accept_observation`: If False, reject and preserve inertial state.
- `suspend_jko_update`: If True, freeze weights and skip Sinkhorn.
- `degraded_mode`: If True, emit degraded inference mode prediction.
- `freeze_kernel_d`: If True, mark kernel D output as frozen (no weight update).
- `staleness_ns`: Staleness in nanoseconds for audit logging.
- `events`: Emitted validation events (outliers, frozen signals, staleness alarms).

### 7.1.3 Early Return on Rejection

If an observation is rejected (catastrophic outlier), the orchestrator returns a degraded result without advancing the state:

```
1 # If observation is rejected, skip state update entirely
2 if reject_observation:
3     updated_state = state
4 else:
5     updated_state = atomic_state_update(...)
```

## 7.2 PRNG Constant

To eliminate magic numbers in PRNG splitting, we introduce a module-level constant in `api/prng.py`:

```
1 # api/prng.py: GLOBAL PRNG CONFIGURATION
2 RNG_SPLIT_COUNT = 2 # For kernel execution subkeys
```

This constant is now imported by `core/orchestrator.py` to maintain layer isolation and clarity. All PRNG-related constants reside in the API layer.

## 7.3 64-bit Precision Enforcement

To ensure Malliavin calculus and Signature computation stability, 64-bit precision must be activated at module import time, before any XLA tracing:

```
1 # api/config.py: JAX CONFIGURATION (at module level)
2 import jax
3 jax.config.update("jax_enable_x64", True)
```

This enforces bit-exact reproducibility across CPU/GPU/FPGA backends and must execute before `ConfigManager` initialization.

# Chapter 8

## Telemetry Buffer Integration (P2.3)

### 8.1 Motivation

Phase 2 implementations (P2.1 WTMM, P2.2 SDE stiffness, V-MAJ violations) generate rich diagnostic data during orchestration. To enable post-mortem analysis, compliance audits, and debugging without stalling inference, P2.3 integrates a **non-blocking telemetry buffer** into the orchestration pipeline.

Key requirements:

- **Non-Blocking:** Logging never blocks compute threads (async enqueue only)
- **Thread-Safe:** Multiple consumers can safely drain buffer
- **Audit Trail:** Records capture complete prediction state snapshot
- **Integrity:** Parity hashes verify weights and free energy
- **Config-Driven:** Emission interval and buffer capacity injected from config

### 8.2 Data Model

Each telemetry record captures:

```
1 @dataclass(frozen=True)
2 class TelemetryRecord:
3     step: int                      # Monotonic counter
4     payload: dict                  # Rich diagnostic data
5
6
7 # Payload structure (P2.3):
8 payload = {
9     "step": 42,
10    "timestamp_ns": 170830800000000000000000,   # Nanosecond precision
11    "prediction_ref": fused_prediction,          # DeviceArray reference
12    "weights_ref": final_rho,                   # DeviceArray reference
13    "free_energy_ref": free_energy,             # DeviceArray reference
14    "kurtosis_ref": updated_state.kurtosis,
15    "holder_exponent_ref": updated_state.holder_exponent,
16    "dgm_entropy_ref": updated_state.dgm_entropy,
17    "modeCollapse_warning": False,              # V-MAJ-5 flag
18    "degraded_mode": False,                     # V-MAJ-7 hysteresis
19    "emergency_mode": False                    # Circuit breaker
20 }
```

Parity hashes are computed during `materialize_telemetry_batch()` after `jax.device_get()` to avoid host-device synchronization in the orchestrator path.

## 8.3 Integration into orchestrate\_step()

The orchestrator now accepts optional `telemetry_buffer` and `step_counter` parameters:

```
1 def orchestrate_step(
2     signal: Float[Array, "n"],
3     timestamp_ns: int,
4     state: InternalState,
5     config: PredictorConfig,
6     observation: ProcessState,
7     now_ns: int,
8     telemetry_buffer: Optional[TelemetryBuffer] = None, # P2.3
9     step_counter: int = 0, # P2.3
10    mutation_rate_limiter: Optional[MutationRateLimiter] = None,
11    degradation_monitor: Optional[DegradationMonitor] = None,
12 ) -> OrchestrationResult:
13     """
14         Run a single orchestration step with telemetry buffering.
15
16         If telemetry_buffer is None, skips telemetry (backward compatible).
17         If provided, enqueues record when hash_interval triggers.
18     """
19
20     # ... existing orchestration logic ...
21
22     # P2.3: Telemetry Buffer Integration (before return)
23     if telemetry_buffer is not None:
24         telemetry_payload = {
25             "step": step_counter,
26             "timestamp_ns": timestamp_ns,
27             "prediction_ref": fused_prediction,
28             "weights_ref": final_rho,
29             "free_energy_ref": free_energy if fusion is not None else jnp.array(0.0),
30             "kurtosis_ref": updated_state.kurtosis,
31             "holder_exponent_ref": updated_state.holder_exponent,
32             "dgm_entropy_ref": updated_state.dgm_entropy,
33             "mode_collapse_warning": mode_collapse_warning,
34             "degraded_mode": degraded_mode,
35             "emergency_mode": emergency_mode,
36         }
37
38         # Emit only when hash_interval triggers (config-driven)
39         if should_emit_hash(step_counter, config.telemetry_hash_interval_steps):
40             telemetry_record = TelemetryRecord(step=step_counter, payload=
41                 telemetry_payload)
42             telemetry_buffer.enqueue(telemetry_record)
43
44     return OrchestrationResult(...)
```

## 8.4 Configuration Parameters

Parameter	Default	Purpose
<code>telemetry_buffer_capacity</code>	1024	Maximum records in ring buffer
<code>telemetry_hash_interval_steps</code>	1	Emit telemetry every N steps

Table 8.1: P2.3 Telemetry Configuration

### 8.4.1 Configuration Injection Example

```

1 # config.toml
2 [io]
3 # Telemetry
4 telemetry_hash_interval_steps = 1          # Emit every step (or 10 for sparser logging)
5 telemetry_buffer_capacity = 1024           # Ring buffer size (zero-heuristics injection)

```

## 8.5 Thread-Safety Model

TelemetryBuffer uses `threading.Lock` for atomic operations:

- `enqueue()`: Acquires lock, appends record, releases ( $O(1)$  amortized)
- `drain()`: Acquires lock, extracts all records, clears buffer, releases
- `size()`: Acquires lock, returns current count, releases

This prevents race conditions when orchestrator (compute thread) enqueues while consumer thread drains.

## 8.6 Backward Compatibility

P2.3 is fully backward compatible:

- If `telemetry_buffer=None`, no telemetry is emitted (default)
- Existing calls to `orchestrate_step()` without telemetry params continue working
- No changes to compute path (telemetry entirely outside `@jax.jit` scope)

## 8.7 Usage Example

```

1 from stochastic_predictor.io.telemetry import TelemetryBuffer
2 from stochastic_predictor.api.config import PredictorConfigInjector
3
4 # Initialize
5 config = PredictorConfigInjector().create_config()
6 telemetry_buffer = TelemetryBuffer(capacity=config.telemetry_buffer_capacity)
7
8 # In prediction loop:
9 for step in range(num_steps):
10     result = orchestrate_step(
11         signal=current_signal,
12         timestamp_ns=now_ns(),
13         state=state,
14         config=config,
15         observation=obs,
16         now_ns=now_ns(),
17         telemetry_buffer=telemetry_buffer,      # P2.3: Pass buffer
18         step_counter=step,                      # P2.3: Pass step
19     )
20
21     # Optional: Drain telemetry in background thread
22     if step \% 100 == 0:
23         records = telemetry_buffer.drain()
24         # Write to file/database (non-blocking, doesn't stall orchestrator)
25         write_telemetry_async(records)

```

## 8.8 Benefits

- **Non-Blocking:** Telemetry enqueue is O(1), never stalls inference
- **Audit Trail:** Complete state snapshots for compliance and debugging
- **Integrity:** Parity hashes enable CPU/GPU parity validation
- **Configurable:** Emission interval and buffer size injected from config
- **Thread-Safe:** Lock-based synchronization for multi-threaded consumers
- **Backward Compatible:** Fully optional, no impact on existing code paths

## 8.9 Visualization Dashboard (GAP-6)

The IO layer includes a static HTML dashboard generator for telemetry snapshots. It requires no external dependencies and produces a standalone report suitable for offline reviews and audits.

### 8.9.1 Usage

```
1 from stochastic_predictor.io.telemetry import TelemetryBuffer
2 from stochastic_predictor.io.dashboard import export_dashboard_snapshot
3
4 telemetry_buffer = TelemetryBuffer(capacity=1024)
5 # ... enqueue telemetry records during inference ...
6
7 exported = export_dashboard_snapshot(
8     telemetry_buffer,
9     "io/reports/telemetry_dashboard.html",
10)
11 print(f"Exported {exported} records")
```

### 8.9.2 Notes

- The dashboard drains the buffer, materializes device arrays, and writes HTML.
- Charts are rendered as inline SVG with no external assets.
- Output is deterministic and suitable for audit trails.

# Chapter 9

## Level 4 Autonomy: Configuration Mutation Safety

### 9.1 Overview

Phase 2.1.0 introduces **Level 4 Autonomy** compliance for autonomous configuration mutations during meta-optimization. This chapter documents the implementation of V-CRIT-2, V-MAJ-4, V-MAJ-5, and V-MAJ-7 violations identified during the specification compliance audit (AUDIT\_SPEC\_COMPLIANCE\_2026-02-19.md).

#### Specification References:

- `I0.tex` §3.3 - Configuration Mutation Protocol (Atomic Write)
- `I0.tex` §3.3.6 - Rate Limiting and Safety Guardrails
- `Implementation.tex` §5.4.3 - Degradation Detection Protocol
- `Theory.tex` §2.3.6 - Monitoring and Telemetry for adaptive SDE schemes

#### Implementation Scope:

- **V-CRIT-2:** Atomic TOML mutation protocol with locked subsection protection
- V-MAJ-4: Configuration mutation rate limiting
- V-MAJ-5: Degradation detection with automatic rollback
- V-MAJ-7: Adaptive telemetry monitoring (partial - safety guardrails)

### 9.2 V-CRIT-2: Atomic Configuration Mutation Protocol

#### 9.2.1 Problem Statement

**Violation:** No atomic write mechanism for config.toml mutations during autonomous meta-optimization. Race conditions, partial writes, and locked parameter violations could corrupt system configuration without rollback capability.

#### Impact:

- Partial file corruption during concurrent writes
- No protection for immutable subsections (e.g., `float_precision`, `snapshot_format`)
- No audit trail for forensic analysis
- Cannot rollback to previous working configuration
- Loss of checkpoint resumability if meta-optimization paths corrupted

## 9.2.2 Requirements

### IO.tex §3.3.2 - Atomic TOML Update Algorithm:

1. **Phase 1 - Validation:** Check parameter ranges, types, locked subsections
2. **Phase 2 - Backup:** Create timestamped backup + latest .bak
3. **Phase 3 - Atomic Write:** Write to .tmp, fsync(), os.replace()
4. **Phase 4 - Audit Log:** Record mutation to io/mutations.log
5. **Phase 5 - Success:** Return or raise on failure

### IO.tex §3.3.4 - Locked Subsections (Asimov's Zeroth Law):

- [meta]: schema\_version
- [core]: jax\_platforms, jax\_default\_dtype, float\_precision, staleness\_ttl\_ns
- [io]: snapshot\_format, snapshot\_hash\_algorithm, snapshot\_compression, telemetry\_hash\_interval\_steps
- [meta\_optimization]: n\_trials, n\_startup\_trials, multivariate, train\_ratio, n\_folds

## 9.2.3 Implementation

Module: stochastic\_predictor/io/config\_mutation.py (EXTENDED)

### Functions Added:

- atomic\_write\_config(): 5-phase POSIX-compliant atomic mutation
- validate\_config\_mutation(): Schema validation + locked subsection checks
- append\_audit\_log(): JSON Lines audit trail logging
- create\_config\_backup(): Timestamped backup creation

### Constants Added:

- LOCKED\_SUBSECTIONS: Immutable parameter dictionary
- VALIDATION\_SCHEMA: Type/range constraints for mutable parameters

## Atomic Write Protocol Implementation

```
1 def atomic_write_config(
2     config_path: Path,
3     new_params: Dict[str, Any],
4     trigger: str = "ManualMutation",
5     best_objective: Optional[float] = None,
6     audit_log_path: Optional[Path] = None,
7 ) -> None:
8     """
9         Atomically mutate configuration with POSIX-compliant write protocol.
10
11     COMPLIANCE: IO.tex §3.3.2 - Atomic TOML Update Algorithm
12     """
13     audit_log_path = audit_log_path or Path("io/mutations.log")
14
15     # Phase 1: Validation
16     current_config = toml.load(config_path)
```

```

17     merged_config = validate_config_mutation(current_config, new_params)
18
19     # Compute delta for audit log
20     delta = {
21         param_key: (_get_nested_param(current_config, param_key), new_value)
22         for param_key, new_value in new_params.items()
23     }
24
25     # Phase 2: Immutable Backup
26     timestamp = datetime.now(timezone.utc).strftime("%Y-%m-%dT%H:%M:%SZ")
27     backup_path = config_path.with_suffix(f".bak.{timestamp}")
28     latest_backup_path = config_path.with_suffix(".bak")
29
30     shutil.copy2(config_path, backup_path)
31     shutil.copy2(config_path, latest_backup_path)
32
33     # Phase 3: Atomic Write via Temporary File
34     tmp_path = config_path.with_suffix(".tmp")
35
36     # O_EXCL flag detects concurrent mutation
37     fd = os.open(tmp_path, os.O_WRONLY | os.O_CREAT | os.O_EXCL, 0o644)
38
39     try:
40         toml_bytes = toml.dumps(merged_config).encode("utf-8")
41         os.write(fd, toml_bytes)
42
43         # CRITICAL: fsync() ensures kernel buffer flush to disk
44         os.fsync(fd)
45     finally:
46         os.close(fd)
47
48     # Phase 4: Atomic Replacement (POSIX os.replace)
49     os.replace(tmp_path, config_path)
50
51     # Phase 5: Audit Logging
52     append_audit_log(audit_log_path, {
53         "timestamp": datetime.now(timezone.utc).isoformat(),
54         "event": "MUTATION_SUCCESS",
55         "trigger": trigger,
56         "delta": delta,
57         "best_objective": best_objective,
58         "backup": str(backup_path),
59         "status": "SUCCESS",
60     })

```

## Validation Schema Implementation

```

1 # Locked subsections (immutable to prevent self-corruption)
2 LOCKED_SUBSECTIONS = {
3     "meta": ["schema_version"],
4     "core": ["jax_platforms", "jax_default_dtype", "float_precision", "staleness_ttl_ns"]
5     ],
6     "io": [
7         "snapshot_format",
8         "snapshot_hash_algorithm",
9         "snapshot_compression",
10        "telemetry_hash_interval_steps",
11    ],
12    "meta_optimization": ["n_trials", "n_startup_trials", "multivariate", "train_ratio",
13    "n_folds"],
14}

```

```

14 # Validation schema for mutable parameters
15 VALIDATION_SCHEMA = {
16     "orchestration.cusum_k": {"type": float, "range": (0.1, 1.0)},
17     "orchestration.cusum_h": {"type": float, "range": (2.0, 10.0)},
18     "orchestration.grace_period_steps": {"type": int, "range": (5, 100)},
19     "orchestration.volatility_alpha": {"type": float, "range": (0.05, 0.3)},
20     "orchestration.learning_rate": {"type": float, "range": (1e-5, 1e-1)},
21     "orchestration.entropy_window": {"type": int, "range": (10, 500)},
22     "orchestration.entropy_threshold": {"type": float, "range": (0.5, 0.95)},
23     "orchestration.holder_threshold": {"type": float, "range": (0.2, 0.65)},
24     "orchestration.sinkhorn_alpha": {"type": float, "range": (0.1, 1.0)},
25     "orchestration.sinkhorn_epsilon_min": {"type": float, "range": (0.001, 0.1)},
26     "orchestration.sinkhorn_epsilon_0": {"type": float, "range": (0.05, 0.5)},
27     "orchestration.sinkhorn_max_iter": {"type": int, "range": (50, 500)},
28     "kernels.log_sig_depth": {"type": int, "range": (2, 5)},
29     "kernels.wtmm_buffer_size": {"type": int, "range": (64, 512),
30                                 "constraint": "power_of_2"},
31     "kernels.besov_cone_c": {"type": float, "range": (1.0, 3.0)},
32     "kernels.dgm_width_size": {"type": int, "range": (32, 256),
33                                "constraint": "power_of_2"},
34     "kernels.stiffness_low": {"type": float, "range": (50.0, 500.0)},
35     "kernels.stiffness_high": {"type": float, "range": (500.0, 5000.0)},
36     "kernels.sde_dt": {"type": float, "range": (0.001, 0.1)},
37     "kernels.sde_numel_integrations": {"type": int, "range": (50, 200)},
38     "kernels.sde_diffusion_sigma": {"type": float, "range": (0.05, 0.5)},
39     "kernels.kernel_ridge_lambda": {"type": float, "range": (1e-8, 1e-3)},
40     # ... additional mutable parameters
41 }
42
43 def validate_config_mutation(
44     current_config: Dict[str, Any],
45     new_params: Dict[str, Any],
46 ) -> Dict[str, Any]:
47     """
48         Validate configuration mutation against schema and locked subsections.
49
50     Raises:
51         ConfigMutationError: If locked parameter mutation attempted
52         ConfigMutationError: If parameter out of safe range
53     """
54     # Check locked subsection violations
55     for param_key in new_params.keys():
56         subsection = param_key.split(".") [0]
57         param_name = ".".join(param_key.split(".")[1:])
58
59         if subsection in LOCKED_SUBSECTIONS:
60             if param_name in LOCKED_SUBSECTIONS [subsection]:
61                 raise ConfigMutationError(
62                     f"Parameter '{param_key}' is LOCKED (immutable)"
63                 )
64
65     # Validate against schema (ranges, types, constraints)
66     merged_config = dict(current_config)
67     for param_key, new_value in new_params.items():
68         rules = VALIDATION_SCHEMA [param_key]
69
70         # Type check
71         if not isinstance(new_value, rules ["type"]):
72             raise ConfigMutationError("Type mismatch")
73
74         # Range check
75         min_val, max_val = rules ["range"]
76         if not (min_val <= new_value <= max_val):

```

```

77     raise ConfigMutationError("Out of safe range")
78
79     # Apply constraints (e.g., power_of_2)
80     if "constraint" in rules:
81         # Check constraint...
82
83     _set_nested_param(merged_config, param_key, new_value)
84
85     # Cross-parameter constraints (e.g., stiffness_low < stiffness_high)
86     # ...
87
88 return merged_config

```

#### 9.2.4 Usage Example

```

1 from stochastic_predictor.io import atomic_write_config
2
3 # After Deep Tuning completes
4 best_params = {
5     "orchestration.cusum_k": 0.72,
6     "kernels.dgm_width_size": 256,
7     "kernels.stiffness_low": 143.0,
8 }
9
10 # Atomic mutation with audit trail
11 atomic_write_config(
12     Path("config.toml"),
13     best_params,
14     trigger="DeepTuning_Iteration_500",
15     best_objective=0.0234  # MAPE
16 )
17
18 # Creates:
19 #   - config.toml.bak.2026-02-19T14:32:05Z (timestamped backup)
20 #   - config.toml.bak (latest backup)
21 #   - io/mutations.log (audit entry)

```

#### 9.2.5 Files Modified

- stochastic\_predictor/io/config\_mutation.py: +280 LOC
- stochastic\_predictor/io/\_\_init\_\_.py: +7 exports

#### 9.2.6 Compliance Impact

**V-CRIT-2 Resolution:** Atomic TOML mutation protocol fully implemented with:

- POSIX-compliant atomic write (fsync + os.replace)
- Locked subsection protection (Asimov's Zeroth Law)
- Validation schema enforcement (20+ mutable parameters)
- Timestamped backups for manual rollback
- JSON Lines audit trail for forensic analysis

**Level 4 Autonomy Enablement:** System can now autonomously mutate config.toml during Deep Tuning campaigns without human intervention, while maintaining safety guarantees.

## 9.3 V-MAJ-4: Configuration Mutation Rate Limiting

### 9.3.1 Problem Statement

**Violation:** No safety guardrails enforced for autonomous configuration mutations during meta-optimization. Optimizer could thrash between configurations, mutate too frequently, or make excessively large parameter jumps.

**Impact:** System instability, pathological optimizer behavior, configuration thrashing without convergence.

### 9.3.2 Safety Requirements

#### IO.tex §3.3.6 Requirements:

- Maximum mutation rate: 10 mutations/hour
- Minimum stability period: 1,000 prediction steps between mutations
- Delta magnitude limit: 50% relative change per parameter
- Audit trail: JSON Lines log of all mutation events

### 9.3.3 Implementation

**Module:** stochastic\_predictor/io/config\_mutation.py (NEW)

**Class:** MutationRateLimiter

```
1 @dataclass
2 class MutationRateLimiter:
3     """Enforce safety guardrails for autonomous configuration mutations.
4
5     Prevents optimizer pathologies:
6         - Thrashing between configurations
7         - Excessive mutation frequency
8         - Large parameter jumps
9         - Pathological degradation without rollback
10
11     Args:
12         max_mutations_per_hour: Maximum allowed mutations/hour (default 10)
13         stability_steps_required: Minimum steps before next mutation (default 1000)
14         max_relative_change: Maximum parameter change per mutation (default 0.5)
15     """
16
17     max_mutations_per_hour: int = 10
18     stability_steps_required: int = 1000
19     max_relative_change: float = 0.5
20
21     _mutation_history: List[Tuple[float, Dict]] = field(default_factory=list)
22     _last_mutation_timestamp: Optional[float] = None
23     _current_steps_since_mutation: int = 0
24
25     def can_mutate(self) -> Tuple[bool, str]:
26         """Check if mutation is allowed under safety guardrails.
27
28         Returns:
29             (allowed: bool, reason: str)
30         """
31
32         # Check maximum mutation rate (sliding 1-hour window)
33         one_hour_ago = time.time() - 3600
34         recent_mutations = [
35             ts for ts, _ in self._mutation_history if ts > one_hour_ago
```

```

35     ]
36     if len(recent_mutations) >= self.max_mutations_per_hour:
37         return False, f"Rate limit: {len(recent_mutations)}/10 mutations"
38
39     # Check minimum stability period
40     if self._current_steps_since_mutation < self.stability_steps_required:
41         return False, f"Stability: {self._current_steps_since_mutation}/1000"
42
43     return True, "OK"
44
45 def validate_delta(
46     self,
47     delta: Dict[str, Tuple[float, float]],
48     max_relative_change: Optional[float] = None
49 ) -> Tuple[bool, str]:
50     """Validate parameter delta magnitude.
51
52     Args:
53         delta: {param: (old_value, new_value)}
54         max_relative_change: Override default max change
55
56     Returns:
57         (valid: bool, reason: str)
58     """
59     max_change = max_relative_change or self.max_relative_change
60
61     for param, (old_val, new_val) in delta.items():
62         if abs(old_val) < 1e-12:
63             continue # Skip zero division
64
65         relative_change = abs((new_val - old_val) / old_val)
66
67         if relative_change > max_change:
68             return False, (
69                 f"{param} change too large: {relative_change:.1%} > 50%"
70             )
71
72     return True, "OK"
73
74 def record_mutation(self, delta: Dict) -> None:
75     """Record successful mutation."""
76     now = time.time()
77     self._mutation_history.append((now, delta))
78     self._current_steps_since_mutation = 0
79
80 def increment_stability_counter(self) -> None:
81     """Call after each prediction step."""
82     self._current_steps_since_mutation += 1

```

### 9.3.4 Usage Pattern

```

1 # In meta-optimizer loop
2 limiter = MutationRateLimiter(max_mutations_per_hour=10)
3
4 # Export best params using atomic mutation protocol
5 optimizer.export_best_params_to_config(
6     "config.toml",
7     trigger="DeepTuning_Iteration_500",
8     rate_limiter=limiter,
9 )
10
11 # In main prediction loop

```

```

12 for step in range(1000):
13     prediction = orchestrate_step(..., mutation_rate_limiter=limiter)
14     # If not passing mutation_rate_limiter, call limiter.increment_stability_counter()
15     # manually

```

## 9.4 V-MAJ-5: Degradation Detection Auto-Rollback

### 9.4.1 Problem Statement

**Violation:** No automatic rollback mechanism when post-mutation performance degrades beyond acceptable threshold. Pathological mutations persist indefinitely, requiring manual operator intervention.

**Impact:** System can enter degraded state without automatic recovery, violating Level 4 autonomy requirements.

### 9.4.2 Safety Requirements

#### IO.tex §3.3.6 Requirements:

- Monitor RMSE over N=100 predictions post-mutation
- Compare to pre-mutation baseline RMSE
- If relative increase > 30%, trigger automatic rollback
- Restore config.toml from config.toml.bak
- Log rollback event to audit trail

### 9.4.3 Implementation

**Module:** stochastic\_predictor/io/config\_mutation.py

**Class:** DegradationMonitor

```

1 @dataclass
2 class DegradationMonitor:
3     """Monitor post-mutation performance and trigger rollback on degradation.
4
5     Implements closed-loop safety mechanism:
6         1. Record pre-mutation baseline RMSE
7         2. Monitor post-mutation predictions (N=100 sample window)
8         3. Compute post-mutation RMSE
9         4. If relative increase > threshold (default 30%), auto-rollback
10
11     Args:
12         degradation_threshold: Max allowed RMSE increase (default 0.3 = 30%)
13         monitoring_window: Predictions to sample post-mutation (default 100)
14         config_path: Path to config.toml (default "config.toml")
15         backup_path: Path to backup config (default "config.toml.bak")
16         audit_log_path: Path to mutation audit log
17         ...
18
19     degradation_threshold: float = 0.3
20     monitoring_window: int = 100
21     config_path: Path = Path("config.toml")
22     backup_path: Path = Path("config.toml.bak")
23     audit_log_path: Path = Path("io/mutations.log")
24
25     _pre_mutation_rmse: Optional[float] = None

```

```

26     _post_mutation_errors: List[float] = field(default_factory=list)
27     _monitoring_active: bool = False
28
29     def start_monitoring(self, baseline_rmse: float) -> None:
30         """Record pre-mutation baseline and start monitoring."""
31         self._pre_mutation_rmse = baseline_rmse
32         self._post_mutation_errors = []
33         self._monitoring_active = True
34
35     def record_prediction_error(self, error: float) -> None:
36         """Accumulate post-mutation prediction error."""
37         if not self._monitoring_active:
38             return
39         self._post_mutation_errors.append(error)
40
41     def check_degradation(self) -> Tuple[bool, float]:
42         """Check if post-mutation performance degraded beyond threshold.
43
44         Returns:
45             (degraded: bool, relative_increase: float)
46         """
47
48         if not self._monitoring_active:
49             return False, 0.0
50
51         # Require full monitoring window
52         if len(self._post_mutation_errors) < self.monitoring_window:
53             return False, 0.0
54
55         # Compute post-mutation RMSE
56         post_mutation_rmse = np.sqrt(
57             np.mean(np.square(self._post_mutation_errors)))
58
59         # Compute relative increase
60         if self._pre_mutation_rmse is None or self._pre_mutation_rmse == 0:
61             relative_increase = 0.0
62         else:
63             relative_increase = (
64                 (post_mutation_rmse - self._pre_mutation_rmse) /
65                 self._pre_mutation_rmse
66             )
67
68         degraded = relative_increase > self.degradation_threshold
69
70         # Stop monitoring after check
71         if degraded or len(self._post_mutation_errors) >= self.monitoring_window:
72             self._monitoring_active = False
73
74         return degraded, relative_increase
75
76     def trigger_rollback(self) -> None:
77         """Execute automatic rollback to pre-mutation configuration.
78
79         CRITICAL: Overwrites config.toml with backup.
80         """
81
82         if not self.backup_path.exists():
83             raise FileNotFoundError(f"Backup config not found: {self.backup_path}")
84
85         # Restore config from backup
86         shutil.copy2(self.backup_path, self.config_path)
87
88         # Compute post-mutation RMSE for logging
89         post_mutation_rmse = float(np.sqrt(

```

```

89         np.mean(np.square(self._post_mutation_errors)))
90     )) if self._post_mutation_errors else 0.0
91
92     # Append rollback event to audit log (JSON Lines format)
93     audit_entry = {
94         'timestamp': datetime.now(timezone.utc).isoformat(),
95         'event': 'AUTO_ROLLBACK',
96         'reason': 'Performance degradation detected',
97         'pre_mutation_rmse': self._pre_mutation_rmse,
98         'post_mutation_rmse': post_mutation_rmse,
99         'relative_increase': (
100             (post_mutation_rmse - self._pre_mutation_rmse) /
101             self._pre_mutation_rmse
102         ),
103         'degradation_threshold': self.degradation_threshold,
104         'monitoring_window': self.monitoring_window,
105         'status': 'ROLLBACK_SUCCESS'
106     }
107
108     # Append to audit log
109     self.audit_log_path.parent.mkdir(parents=True, exist_ok=True)
110     with open(self.audit_log_path, 'a') as f:
111         f.write(json.dumps(audit_entry) + '\n')
112
113     # Reset monitoring state
114     self._monitoring_active = False
115     self._post_mutation_errors = []

```

#### 9.4.4 Usage Pattern

```

1 # Before mutation
2 monitor = DegradationMonitor(degradation_threshold=0.3)
3
4 # Compute baseline RMSE over recent predictions
5 recent_errors = [0.04, 0.05, 0.06, 0.05, 0.04]
6 baseline_rmse = np.sqrt(np.mean(np.square(recent_errors)))
7
8 # Apply mutation
9 delta = {"orchestration.learning_rate": 0.015}
10 atomic_write_config(Path("config.toml"), delta, trigger="AutoTuning")
11
12 # Start monitoring
13 monitor.start_monitoring(baseline_rmse)
14
15 # In prediction loop (next 100 predictions)
16 for i in range(100):
17     prediction = orchestrate_step(..., degradation_monitor=monitor)
18     actual_next = load_actual_observation()
19     error = abs(prediction.predicted_next - actual_next)
20
21     monitor.record_prediction_error(error)
22
23     # Check for degradation
24     degraded, increase = monitor.check_degradation()
25     if degraded:
26         logger.critical(f"Degradation detected: RMSE +{increase:.1%}")
27         monitor.trigger_rollback()
28         logger.info("Config rolled back to pre-mutation state")
29         break

```

### 9.4.5 Audit Log Format

Mutation events are logged in JSON Lines format to `io/mutations.log`:

```
1 {"timestamp": "2026-02-19T14:30:00Z", "event": "MUTATION_SUCCESS",
2 "delta": {"orchestration.learning_rate": [0.01, 0.015]}, "status": "SUCCESS"}
3 {"timestamp": "2026-02-19T14:35:00Z", "event": "AUTO_ROLLBACK",
4 "reason": "Performance degradation detected",
5 "pre_mutation_rmse": 0.05, "post_mutation_rmse": 0.07,
6 "relative_increase": 0.40, "degradation_threshold": 0.30,
7 "status": "ROLLBACK_SUCCESS"}
```

## 9.5 V-MAJ-7: Adaptive Telemetry Monitoring

### 9.5.1 Implementation Status

V-MAJ-7 requires comprehensive telemetry for adaptive architecture and solver selection metrics. This implementation extends `InternalState` with counters for Level 4 autonomy monitoring and provides full adaptive telemetry collection.

### 9.5.2 InternalState Extensions

**Module:** `stochastic_predictor/api/types.py`

Added fields to `InternalState` (lines 443-447):

```
1 # V-MAJ-7: Level 4 Autonomy Adaptive Telemetry
2 baseline_entropy: Float[Array, "1"]      # H_baseline: Reference entropy for
3 solver_explicit_count: int              # N_explicit: Explicit SDE solver steps
4 solver_implicit_count: int              # N_implicit: Implicit SDE solver steps
5 architecture_scaling_events: int        # N_scale: DGM architecture scaling events
```

**Purpose:**

- `baseline_entropy`: Reference entropy  $H_{\text{baseline}}$  for computing entropy ratio  $\kappa = H_{\text{current}}/H_{\text{baseline}}$
- `solver_explicit_count`, `solver_implicit_count`: Track SDE solver scheme frequencies within monitoring window (default 100 steps)
- `architecture_scaling_events`: Count DGM architecture scaling events (cumulative)

### 9.5.3 Adaptive Telemetry Collection

**Module:** `stochastic_predictor/io/telemetry.py`

**Function:** `collect_adaptive_telemetry()`

```
1 def collect_adaptive_telemetry(
2     state: Any,    # InternalState
3     config: Any,   # PredictorConfig
4     window_size: int = 100
5 ) -> Optional[AdaptiveTelemetry]:
6     """
7         Collect telemetry for adaptive architecture/solver diagnostics.
8
9     COMPLIANCE: V-MAJ-7 - Adaptive Telemetry Monitoring
10
11    Args:
12        state: Current InternalState (contains counters, entropy, etc.)
13        config: Current PredictorConfig (may have been mutated)
14        window_size: Monitoring window for frequency calculations
15
```

```

16     Returns:
17         AdaptiveTelemetry instance or None if insufficient data
18     """
19     import jax.numpy as jnp
20
21     # Compute solver frequencies (clipped to [0,1])
22     total_solver_steps = state.solver_explicit_count + state.solver_implicit_count
23     if total_solver_steps == 0:
24         return None # Insufficient data
25
26     freq_explicit = float(state.solver_explicit_count) / total_solver_steps
27     freq_implicit = float(state.solver_implicit_count) / total_solver_steps
28
29     # Compute entropy ratio = H_current / H_baseline
30     baseline_entropy_val = float(state.baseline_entropy)
31     if baseline_entropy_val <= 0.0:
32         entropy_ratio = 1.0 # Avoid division by zero
33     else:
34         entropy_ratio = float(state.dgm_entropy) / baseline_entropy_val
35
36     # Extract DGM architecture from config
37     dgm_width = config.dgm_width_size
38     dgm_depth = config.dgm_depth
39
40     # Extract JKO flow parameters from config
41     entropy_window = config.entropy_window
42     learning_rate = config.learning_rate
43     volatility_sigma_squared = float(state.ema_variance)
44
45     # Extract adaptive stiffness thresholds from config
46     stiffness_low = config.stiffness_low
47     stiffness_high = config.stiffness_high
48     holder_exponent_val = float(state.holder_exponent)
49
50     return AdaptiveTelemetry(
51         # SDE Solver Frequency
52         scheme_frequency_explicit=freq_explicit,
53         scheme_frequency_implicit=freq_implicit,
54         max_stiffness_metric=0.0, # Placeholder - requires Kernel C integration
55         num_internal_iterations_mean=0.0, # Placeholder
56         implicit_residual_norm_max=0.0, # Placeholder
57
58         # DGM Architecture
59         entropy_ratio_current=entropy_ratio,
60         dgm_width_current=dgm_width,
61         dgm_depth_current=dgm_depth,
62         architecture_scaling_events=state.architecture_scaling_events,
63
64         # JKO Flow
65         entropy_window_current=entropy_window,
66         learning_rate_current=learning_rate,
67         volatility_sigma_squared(volatility_sigma_squared),
68
69         # Stiffness Thresholds
70         stiffness_low_adaptive=stiffness_low,
71         stiffness_high_adaptive=stiffness_high,
72         holder_exponent_wtmm=holder_exponent_val
73     )

```

#### 9.5.4 Integration Pattern

Adaptive telemetry collection is called periodically (e.g., every 100 steps) in the orchestration loop:

```

1 # In orchestrate_step() or meta-loop
2 if step % telemetry_interval == 0:
3     adaptive_tel = collect_adaptive_telemetry(state, config)
4     if adaptive_tel:
5         emit_adaptive_telemetry(adaptive_tel)

```

**Output:** JSON Lines format appended to `io/adaptive_telemetry.jsonl`

### 9.5.5 Metrics Tracked

#### SDE Solver Frequency:

- `scheme_frequency_explicit`: Fraction of explicit Euler steps in window  $\in [0, 1]$
- `scheme_frequency_implicit`: Fraction of implicit solver steps in window  $\in [0, 1]$

#### DGM Architecture:

- `entropy_ratio_current`:  $\kappa = H_{\text{current}}/H_{\text{baseline}}$  (regime transition detector)
- `dgm_width_current`, `dgm_depth_current`: Current DGM network dimensions (may be scaled dynamically)
- `architecture_scaling_events`: Cumulative count of architecture mutations

#### JKO Flow:

- `entropy_window_current`: Current entropy window  $W \propto L^2/\sigma^2$
- `learning_rate_current`: Current JKO learning rate  $\eta < 2\varepsilon\sigma^2$
- `volatility_sigma_squared`: Current EWMA variance  $\sigma^2$

#### Stiffness Thresholds:

- `stiffness_low_adaptive`, `stiffness_high_adaptive`: Hölder-informed thresholds  $\theta_L, \theta_H \propto 1/(1 - \alpha)^2$
- `holder_exponent_wtmm`: Current Hölder exponent  $\alpha \in [0.2, 0.9]$

### 9.5.6 Future Extensions

- `max_stiffness_metric`: Maximum stiffness metric  $\mathcal{S}$  in window (requires Kernel C integration)
- `num_internal_iterations_mean`: Average Newton iterations for implicit solver
- `implicit_residual_norm_max`: Maximum residual norm for implicit solver convergence

These metrics require deeper integration with Kernel C's SDE solver internals and are currently placeholders (set to 0.0).

## 9.6 Utility Functions

Module: stochastic\_predictor/io/config\_mutation.py

```
1 def create_config_backup(
2     config_path: Path = Path("config.toml"),
3     backup_path: Path = Path("config.toml.bak")
4 ) -> None:
5     """Create config backup before mutation."""
6     if not config_path.exists():
7         raise FileNotFoundError(f"Config file not found: {config_path}")
8     shutil.copy2(config_path, backup_path)
9
10 def append_audit_log(log_path: Path, entry: Dict) -> None:
11     """Append mutation event to audit log (JSON Lines format)."""
12     log_path.parent.mkdir(parents=True, exist_ok=True)
13     with open(log_path, 'a') as f:
14         f.write(json.dumps(entry) + '\n')
```

## 9.7 Implementation Status

V-MAJ Violation	Status	Module
V-MAJ-4 (Rate Limiting)	Implemented	config_mutation.py
V-MAJ-5 (Auto-Rollback)	Implemented	config_mutation.py
V-MAJ-7 (Telemetry)	Implemented	telemetry.py, types.py

Table 9.1: Level 4 Autonomy - Configuration Mutation Safety Implementation

**Note:** V-MAJ-7 is fully implemented with `InternalState` extensions (solver counters, baseline entropy, architecture scaling events) and `collect_adaptive_telemetry()` function. Three Kernel C solver metrics remain as placeholders (`max_stiffness_metric`, `num_internal_iterations_mean`, `implicit_residual_norm_max`) requiring deeper SDE solver integration in a future phase.

# Chapter 10

## Compliance Checklist

- **No Compute Stalls:** All logging is asynchronous
- **Binary Format:** MessagePack for snapshots
- **Atomic Snapshots:** Write-then-rename protocol
- **Deterministic Hashing:** SHA-256 on  $\rho$  and OT cost
- **Security:** No raw signals, VRAM dumps, or secrets
- **Integrity:** Snapshot hashes verified before load
- **Config-Driven:** Intervals and destinations are injected
- **Module Coverage:** IO helpers implemented for validation, telemetry, snapshots, and credentials
- **Orchestrator Integration:** IO ingestion gate integrated into `orchestrate_step()`
- **PRNG Constants:** Named constants (`RNG_SPLIT_COUNT`) reside in `api/prng.py`
- **Buffer Capacity Injection:** TelemetryBuffer capacity injected from config (zero-heuristics policy)
- **64-bit Precision:** Enforced at module load time (`api/config.py`) before XLA tracing
- **Layer Isolation:** PRNG constants in API layer, not Core layer

## **Chapter 11**

# **Phase 4 Summary**

Phase 4 introduces a non-blocking I/O architecture that preserves deterministic compute while enabling telemetry, logging, and atomic snapshot persistence.