Shunyaya Symbolic Mathematical Browse (SSM-Browse)

A Deterministic, Ethical, Open Standard for Symbolic Browsing

Status: Public Research Release (v2.0)
Date: November 30, 2025
Caution: Research/observation only. Not for critical decision-making.

---

0. Executive Overview

Modern browsing has become a paradox: it is the primary interface to the digital world, yet the rules governing **what appears**, **how it appears**, and **why it appears** remain hidden. Behind every page load, suggestion, ranking, and redirect, there are:

• undisclosed algorithms
• hidden prioritization weights
• unpredictable caching and redirect behavior
• non-transparent ranking logic
• non-verifiable interpretations of user activity

This creates a browsing environment where:

• the same query yields different outcomes across devices
• ranking shifts silently without explanation
• suggestions evolve in ways users cannot inspect
• organizations cannot reliably reconstruct navigation trails
• institutions cannot guarantee structural neutrality or fairness

SSM-Browse introduces a different idea:
browsing can be **structurally truthful**, **mathematically reproducible**, and **auditable** — without interpreting content or profiling individuals.

It does **not** change what users search or visit.
It changes **how browsers describe structural behavior**.

Every navigation action is wrapped in a **compact symbolic envelope**, providing deterministic structure without touching, inspecting, or interpreting the underlying value.

Two-Layer Architecture

Each browsing action carries **exactly two layers**:

1. Value — The Original Browsing Action

URL visit, search term, click, scroll, resource load, form submit, or interaction.
Untouched.
Never modified, ranked, filtered, interpreted, boosted, or demoted.

2. Envelope — A Transparent Structural Description

A small, deterministic record defining posture, coherence, and declared rules:

• a bounded symbolic posture `a ∈ (-1,+1)`
• optional coherence `q` for multi-tab stability
• a transparent structural band
• a declared manifest (structural ruleset)
• an optional deterministic navigation stamp (`stamp = sha256(prev || payload)`) — providing tamper-visible ordering without requiring time, identity, or online connectivity
• an optional signature field (never influencing alignment)

This symbolic envelope provides the missing layer in modern browsing:

• same visit → same structural outcome everywhere
• ranking behavior becomes explainable
• suggestions become structurally traceable
• navigation history becomes provable and tamper-visible
• browsing becomes mathematically fair and auditable

Structural Guarantees

SSM-Browse ensures:

• no hidden weighting
• no silent personalization
• no inference or profiling
• no platform-dependent behavior
• no undisclosed correlations
• no semantic interpretation of user actions

It is:

• **open-standard**
• **platform-neutral**
• **non-semantic**
• **deterministic**
• **safe by design**
• **compatible with all existing browsers**

A Lightweight Foundation for Ethical Browsing

One of the defining characteristics of **SSM-Browse** is its **exceptionally small and transparent footprint**.
From the beginning, the framework was designed to prove that **determinism, fairness, and structural truth do not require heavy engines, ML, telemetry, or large dependencies**.

Today, SSM-Browse is available in **four ultra-compact editions**, each demonstrating a different integration path while remaining fully deterministic and non-semantic:

• **Core Edition (~5 KB)** — minimal symbolic engine with alignment & Quero lanes, drift computation, deterministic stamps, and replay.
• **Research Edition (~8 KB)** — adds envelope blocks, manifest IDs, ZETA-0 indicators, Quero coherence, and richer structural metadata for platform or academic R&D.
• **WebExtension Edition (~6 KB)** — a pure sidecar extension using the WebExtensions API, capturing real tab events without touching URLs, content, or DOM. Fully product-agnostic.
• **DevTools-Class Edition (~5 KB)** — a single-file, embedded, interactive DevTools-style analyzer with multi-panel visualization, drift matrices, instant stamping, and structural replay controls.

All four editions are:

• **fully local** (no network, no backend)
• **deterministic** (identical behavior across devices)
• **non-semantic** (zero content interpretation)
• **dependency-free** (HTML + JS only)
• **product-agnostic** (compatible with any browser or platform)

This extreme minimalism enables:

• seamless embedding into existing browsers
• deployment even on low-resource or fully offline environments
• portable research, compliance, and audits
• structural transparency without computational overhead
• safe experimentation without modifying browser internals

Together, the four editions form a **complete demonstration suite**, proving that true verifiability, determinism, and structural integrity can be achieved with **only a few kilobytes of symbolic logic** — not with heavyweight engines or opaque algorithms.

A New Class of Browsing

SSM-Browse transforms browsing from a black-box environment into a structurally truthful, replayable, and audit-ready digital journey — anchored not in inference or prediction, but in mathematical clarity.

It introduces a browsing world where:

• structure is visible
• alignment is deterministic
• coherence is measurable
• drift is detectable
• stamps make navigation tamper-evident

In essence, SSM-Browse brings mathematical honesty to the browsing experience, enabling a new era of transparent, trustworthy, and ethically aligned digital navigation.

Incredibly, this entire structural transformation is achieved with engines measured in **kilobytes**, not megabytes.

Such compactness makes SSM-Browse uniquely suited for **low-resource devices, offline environments, intranet systems, regulatory sandboxes, and minimal-compute platforms**, ensuring structural truth even where full browsers or ML systems cannot operate.

Beyond transparency, SSM-Browse also establishes **structural universality**:
the guarantee that the same browsing action — on any device, browser, region, or future version — produces the *exact* same structural outcome, as long as the manifest remains unchanged.

This elevates browsing from vendor-defined behavior into a consistent, standards-grade structural experience that survives platforms, versions, and environments.

TABLE OF CONTENTS

1. The SSM-Browse Model

SSM-Browse introduces a simple but profound change to digital navigation:

Every browsing action must carry both the event itself and the structural truth describing it.

Traditional browsers transmit only the *event*:

- a URL request
- a click
- a search query
- a tab switch
- a scroll
- a resource load

Everything else—ranking decisions, visibility shifts, redirect paths, tab stability, and suggestion behavior—is generated later by complex and non-transparent systems. Because these systems operate privately, each device or session may produce a different outcome, making browsing:

- unpredictable
- non-reproducible
- non-auditable
- non-verifiable

SSM-Browse fixes this mismatch by ensuring that **every event** is accompanied by a deterministic symbolic envelope that exposes its structural posture, coherence, and declared rules.

The model operates on **two explicit layers**:

1.1 Value Layer — The Original Browsing Action (Untouched)

The **Value Layer** represents the exact action the user performs, such as:

- URL visit
- search term
- click event
- tab activation
- scroll position
- resource fetch
- navigation intent

This value is **never** modified, interpreted, transformed, ranked, or filtered by SSM-Browse.

It is:

- **not re-ranked**
- **not personalized**
- **not redirected**
- **not boosted or demoted**
- **not semantically analysed**
- **not rewritten or enriched**

This preserves:

- **authenticity** — the event remains exactly what the user did
- **neutrality** — no weighting, scoring, or bias is introduced
- **intent clarity** — the meaning of the action is not inferred or altered
- **platform independence** — the structure remains identical on all systems

The browsing action therefore remains **pure, deterministic, and intact**, forming the foundation on which the structural envelope operates.

SSM-Browse does not perform real browsing or rendering.
It does **not**:

- load pages
- execute scripts
- fetch network resources
- interpret URLs
- evaluate content

Its role is to **record**, **stabilize**, and **standardize** the **structural description** of browsing events, not to simulate or execute web activity.

It is a **symbolic layer**, not a browser engine.

1.2 Envelope Layer — The Structural Description of the Visit

The **Envelope Layer** is a compact symbolic structure that accompanies every browsing action.
It does **not** describe meaning, intent, or interpretation.
It describes **structural posture, structural coherence, and declared rules**, and is processed **identically across all systems**.

Its purpose is simple:
Every browsing event must carry a **deterministic structural truth** that can be **replayed, verified, and audited** without accessing or interpreting content.

The envelope contains **five required elements**, plus **one optional non-core field**.

---

1. Posture Signal (a) — bounded in (-1,+1)

The posture signal `a` expresses the structural posture of the browsing event.

It is derived from a declared raw value `a_raw`, which must conform to one manifest-declared mode:

- **user-declared**
- **browser-assigned** (rule-based, deterministic)
- **hybrid** (user input constrained by manifest rules)

Bounded transform:

```
a_c = clamp(a_raw, -1+eps, +1-eps)
a   = tanh(atanh(a_c))
```

Strict rules:

- no inference
- no behavioral prediction
- no hidden profiling
- no tracking-based adjustments

Only **declared, deterministic posture logic** is permitted.

---

2. Optional Coherence Signal (q)

The coherence signal `q` provides a structural measure of stability across navigation flows such as:

- multi-tab workflows
- redirect chains
- sequential research sequences
- long branching navigation paths

`q` expresses **structural coherence**, not semantic coherence.
It never inspects or interprets content.

---

3. Band (Structural Category)

A **band** is a transparent structural category derived entirely from manifest thresholds:

- `NAVIGATE`
- `NEUTRAL`
- `RESTRICT`
- `VERIFY`
- `WARN`

Bands are **purely structural**.
They **never** trigger ranking, filtering, personalisation, or behavioral interpretation.

---

4. Manifest ID

Every envelope identifies its **active manifest**, the deterministic rulebook that declares:

- posture computation rules
- clamping logic
- band thresholds
- coherence evolution rules
- replay rules
- optional ordering semantics

All reconstruction depends only on the **manifest + envelope sequence**.
Nothing may exist outside the manifest.

---

5. Optional Stamp (Tamper-Visible Ordering)

A stamp provides deterministic ordering:

```
stamp = sha256(prev || payload)
```

It enables:

- tamper-evident ordering
- structural continuity
- cross-device replay
- offline verification
- complete session reconstruction

Strongly recommended for audit-grade or regulatory environments.

Optional Field (Non-Core): Signature (sig)

`sig` may act as a lightweight authenticity seal.
It:

- never influences posture
- never influences coherence
- never influences banding
- never influences replay behavior

It is **safe to ignore**.

The Role of the Envelope

The envelope creates a **portable, platform-independent structural truth** for browsing — without inspecting content, interpreting meaning, ranking results, or influencing visibility.

It guarantees:

- **fairness**
- **reproducibility**
- **structural neutrality**
- **cross-platform consistency**
- **survivability across browser versions and architectures**

As long as the manifest remains unchanged, the same browsing action always produces the **same envelope everywhere**.

Mini-Browser Demonstrator (Reference Only)

In the reference demonstrator, envelope fields appear visually as:

- lane panels
- event lists
- drift matrices
- replay integrity indicators

These panels are **developer-only** and never modify browsing actions.
They are **not** part of the structural standard.

1.3 Why Two Layers Matter in Browsing

The dual-layer architecture of SSM-Browse solves structural problems that have persisted since the earliest days of digital navigation.

• Ranking unpredictability disappears

Structural posture determines the structural outcome everywhere.
Same event → same classification → same envelope → same behavior.

• Cross-device consistency becomes guaranteed

`Value + Envelope` produces identical structural responses across all systems, independent of device settings, personalization models, or hidden heuristics.

• Navigation sequences become provable

With symbolic stamps, any sequence of browsing actions can be reconstructed exactly as it occurred, ensuring structural continuity and tamper visibility.

• Browsers cannot introduce silent weighting or personalization

All structural rules arise only from the declared manifest.
Nothing outside the manifest may influence posture, coherence, or banding.

• Institutions gain a replayable, tamper-visible structural record

Compliance, research, archival, and regulatory use cases can reconstruct structural flows **without collecting or interpreting content**.
This meets transparency needs without compromising privacy.

Together, these properties shift browsing from **opaque algorithmic interpretation** to **deterministic symbolic structure**, establishing a universal standard for reproducible digital navigation.

1.4 Envelopes Are Universal Across Browsing Contexts

Because SSM-Browse is non-semantic, deterministic, and rule-bound, envelopes can accompany any browsing-related event, including:
• direct navigation
• search results
• link following
• tab creation
• page refresh

• resource loads
• app-like embedded views
• iframe transitions
• redirects
• autocompletes
• background fetches or network requests

No redesign of the underlying platform is required.

Overlay Mode

Envelopes sit beside normal browsing events, forming a sidecar symbolic track.
The existing value layer and browsing engine remain untouched.

Native Mode

Envelopes become structural browsing objects that participate in:
• alignment evolution (`a_lane`)
• Quero coherence updates (`q_lane`)
• ZETA-0 stabilization when drift crosses thresholds
• deterministic stamps and lineage
• drift computation across tabs and sessions

Both modes preserve:
• compatibility
• neutrality
• determinism
• platform independence

The envelope's universality is what makes SSM-Browse a lightweight, portable, and structurally consistent standard across all browsing environments.

Glossary (Local to This Section)

To maintain clarity across all browsing contexts, the following symbolic terms apply:

Envelope — a structural object containing alignment, coherence, stamps, drift, and manifest-declared fields.
Alignment Lane (`a_lane`) — bounded posture describing how structural alignment evolves.
Quero Lane (`q_lane`) — bounded coherence describing smoothness or shocks across interactions.

Drift — symbolic divergence between tabs or across session timelines, expressed using canonical thresholds.
ZETA-0 — a neutral reset event (`a_lane = 0`, `q_lane = 0`) triggered when cumulative drift exceeds the manifest's declared limit.
Manifest — the rule document specifying envelopes, lane behavior, drift limits, resets, and structural guarantees.

1.5 Example (Human-Readable)

Value (Original Browsing Action)

Visit: `https://example.com/research/energy`

Envelope (Structural Description)

```
a_lane   = +0.31
q_lane   = +0.12
band     = NAVIGATE
manifest = B1
stamp    = sha256(...)
sig      = <optional>
```

Interpretation

- **The URL remains exactly as visited**; no semantics, keywords, or content are read or interpreted.
- **`a_lane` and `q_lane`** are deterministic, bounded outputs computed using the alignment kernel and manifest-declared rules.
- **`band = NAVIGATE`** is derived solely from structural thresholds in the manifest — never from meaning or intent.
- The **stamp** is generated only from structural lineage inputs and provides **tamper-visible ordering** without identity or tracking.
- The **signature (`sig`)** is optional and has **zero influence** on posture, coherence, banding, or replay behavior.
- **ZETA-0** will apply only if cumulative drift exceeds manifest-defined structural limits (not triggered here).

Because the envelope is **non-semantic**, **bounded**, and **manifest-driven**, any compliant implementation can replay this event and reproduce identical:

- structural posture (`a_lane`)
- structural coherence (`q_lane`)
- band classification
- structural lineage

- stamp ordering

…across **all devices, browsers, architectures, languages, or future versions**, as long as the manifest is unchanged.

This guarantees **structural universality** — the same browsing event produces the **same envelope everywhere**, independent of platform behavior or interpretation.

1.6 Deterministic Navigation Stamps (Optional but Strongly Recommended)

SSM-Browse supports an optional *deterministic navigation stamp* that allows any platform or auditor to verify the exact ordering of browsing actions—without using clocks, identity, or external data sources.

A navigation stamp is a small, deterministic hash derived only from:

- the current action's symbolic envelope
- the manifest's declared rules
- (optionally) the previous stamp in the chain

This produces a tamper-visible sequence where:

- **reordering becomes detectable**
- **missing steps become detectable**
- **injected steps become detectable**
- **session reconstruction becomes trivial**

Importantly, stamps are:

- **non-semantic** — no URLs, keywords, or content included
- **stateless** — do not require servers or clocks
- **offline-safe** — work fully inside intranets or air-gapped networks
- **privacy-safe** — contain no user information
- **lightweight** — only a few bytes, derived from deterministic rules

Because they are optional, platforms may adopt SSM-Browse in pure envelope mode or envelope-plus-stamp mode.
But when used, stamps provide a major advantage:
they transform browsing into a fully reconstructable and tamper-visible structural record, while remaining mathematically neutral and entirely non-intrusive.

Stamps never influence ranking, ordering, visibility, personalization, or browsing behavior. They exist only to support integrity, replayability, and trust.

2. The Alignment Kernel

Modern browsing is shaped by a dense network of **hidden decision systems**.
Every search result, autocomplete suggestion, reordered item, visibility change, or prioritization is influenced by structural logic that users cannot see.

The problem is not that structure exists.
The problem is that:

- users cannot verify it
- institutions cannot audit it
- browsers cannot replay it
- platforms silently modify it
- no two devices interpret structure the same way

SSM-Browse solves this by eliminating all hidden structural interpretation.
In its place, it introduces one transparent, deterministic mathematical engine:

The Alignment Kernel

The Alignment Kernel has a single purpose:
convert raw posture inputs `a_raw` into a bounded, safe, reproducible structural signal `a ∈ (-1,+1)` for browsing.

This signal:

- does **not** describe meaning
- does **not** judge content
- does **not** infer preferences
- is a **structural representation**, not a semantic one

It is the foundation of deterministic structural browsing.

What the Alignment Kernel Never Does

The kernel does **not**:

- judge or classify content
- interpret meaning or intent
- score user behavior
- derive relevance
- adjust based on history
- profile or personalize

- use machine learning
- adapt automatically or “learn” over time

The kernel is **pure math** — deterministic, rule-bound, transparent, and predictable.

What the Alignment Kernel Always Does

The kernel takes a **declared** posture value `a_raw`, applies manifest-defined rules, and produces a stable structural posture:

```
a_c = clamp(a_raw, -1+eps, +1-eps)
u   = atanh(a_c)
U   = DECAY_W * U + w * u
W   = DECAY_W * W + w
a   = tanh( U / max(W, eps_w) )
```

Where:

- `a_c` is the clamped posture
- `u` is the rapidity-space representation
- `U` is cumulative rapidity
- `W` is cumulative manifest-declared weight
- `eps_w` prevents numerical instability
- `a` is the final bounded posture of the event

Properties of the Kernel

• Boundedness

The structural posture always stays within `(-1,+1)`.
It cannot overflow, saturate, or destabilize, even during long browsing sessions.

• Determinism

The same envelope sequence always produces the same posture output on:

- any device
- any browser
- any language
- any architecture
- any future version

• Transparency

All rules are declared explicitly in the manifest:
no inference, no hidden heuristics, no private logic.

• Reproducibility

Any compliant system can replay the envelope sequence and compute **identical posture values** purely from:

- the envelope
- the manifest
- the alignment kernel formulas

In effect, the Alignment Kernel transforms browsing from a system of invisible, non-reproducible heuristics into a system of transparent, mathematically governed structure.
It is the first universal, content-agnostic engine that provides deterministic structural truth across the entire browsing ecosystem.

2.1 What the Kernel Solves in Browsing

The Alignment Kernel solves the structural problems that have made browsing unpredictable for decades. It replaces opaque interpretation with deterministic, mathematically provable behavior.

1. Consistency

The same navigation envelope always produces the same structural posture.
No device-level differences. No environment-based variations.

2. Fairness

No silent weighting, personalization, or implicit ranking can influence outcomes.
Structure arises **only** from declared manifest rules.

3. Stability

Structural drift disappears.
Tab interactions, redirects, search flows, and multi-step navigation sessions maintain predictable alignment.

4. Replayability

Any browsing sequence can be reconstructed:

- deterministically
- platform-independently
- without relying on hidden browser logic
- without interpreting content

5. Auditability

Institutions can validate navigation structure without collecting or accessing page data. Structural envelopes provide transparency while preserving privacy.

Together, these properties give browsing the same **structural discipline** that fields like engineering and formal verification rely on—something traditional browsing has never possessed.

2.2 How It Works

The Alignment Kernel transforms `a_raw` into a bounded structural posture using a clear, transparent, three-step process.
No semantics. No inference. No machine learning.
Just deterministic mathematical rules.

Step 1 — Clamp

The raw posture value is clamped to prevent instability, overflow, or manipulation:

```
a_c = clamp(a_raw, -1+eps_a, +1-eps_a)
```

Clamping ensures the posture always remains safely inside the open interval `(-1,+1)`.

Step 2 — Transform

The clamped value is mapped into rapidity space so that posture contributions combine smoothly:

```
u = atanh(a_c)
```

This transformation makes posture evolution stable even during long sessions.

Step 3 — Aggregate

The kernel combines posture contributions using declared manifest rules:

```
U = DECAY_W * U + w * u
W = DECAY_W * W + w
a_out = tanh( U / max(W, eps_w) )
```

Where:

- `U` is the cumulative rapidity
- `W` is the cumulative declared weight
- `eps_w` ensures numerical safety

The final output is:

- **smooth**
- **safe**
- **deterministic**
- **identical across languages, devices, and platforms**

No system may alter, enrich, or reinterpret these steps outside the manifest.
All structural behavior must be explicitly declared, making the Alignment Kernel the first fully auditable structural engine for browsing.

2.3 Why a Bounded Signal Matters in Browsing

Browsing naturally produces **long, branching, and structurally volatile** chains of symbolic actions:

- search → click → scroll → open next tab
- redirects → dynamic loads → nested transitions
- parallel multi-tab research
- iterative exploration loops with structural shocks

Without a **bounded** symbolic lane, structural drift accumulates indefinitely.
This leads to:

- instability
- device-dependent posture divergence
- unpredictable tab behavior
- irreproducible navigation paths

A bounded signal is essential for structural safety, replayability, and fairness.

A bounded posture ensures that SSM-Browse:

- **prevents posture explosions** during deep or long sessions
- **caps extreme fluctuations** in symbolic alignment
- **keeps Quero coherence stable** across transitions
- **maintains fairness** across multi-tab or parallel exploration
- **ensures mathematical safety** during cumulative navigation
- **guarantees deterministic replay** across devices and environments

With bounding, no matter how long, complex, or multi-threaded the browsing becomes, the structural envelope remains:

- **predictable**
- **reproducible**
- **platform-neutral**
- **safe for deterministic reconstruction**

Both alignment (`a_lane`) and coherence (`q_lane`) therefore remain strictly within `(-1, +1)` using the kernel's bounded transform:

```
lane = tanh( Σ w_i * atanh(raw_i) / Σ w_i )
```

This bounded formulation is the **cornerstone of deterministic structural browsing**.
It enables:

- stable Quero coherence
- controlled structural drift
- verifiable replay
- device-independent structural consistency

Bounding is what makes SSM-Browse mathematically disciplined, safe over arbitrarily long sessions, and universally reproducible across architectures.

2.4 Example — Kernel in a Browsing Session

Consider a typical navigation sequence:

- perform a search for "energy data"
- click a result
- follow an internal link
- open a related tab
- scroll deeply
- navigate back

Each event carries a declared posture, such as:

```
+0.12
-0.25
+0.07
```

The Alignment Kernel processes each posture value using the manifest's deterministic rules:

1. **Clamp the posture:**

```
a_c = clamp(a_raw, -1+eps_a, +1-eps_a)
```

2. **Transform to rapidity space:**

```
u = atanh(a_c)
```

3. **Aggregate using declared weights:**

```
U = DECAY_W * U + w * u
W = DECAY_W * W + w
a_out = tanh( U / max(W, eps_w) )
```

The result is a **smooth**, **bounded**, **stable** alignment signal that reflects the structural posture of the entire sequence.

Because the kernel is deterministic and transparent, **any system replaying the same envelope sequence will compute exactly the same output**.

This is the foundation of **structurally truthful browsing** in SSM-Browse—every structure can be reproduced, inspected, verified, and audited without accessing or interpreting content.

---

3. Modes of Operation

SSM-Browse is designed to work **immediately** with any existing browsing environment while also providing a clear path toward a fully deterministic future browsing architecture.

For this reason, SSM-Browse supports **two complementary modes**:

- **Overlay Mode — zero friction, instant adoption**
- **Native Mode — full structural browsing with posture memory, coherence evolution, and replayable tab logic**

Both modes use **the same envelope format**, the same Alignment Kernel, and the same manifest rules.
The difference lies only in **how deeply** the browsing environment integrates structural behavior.

This dual-mode design allows SSM-Browse to be:

- deployable today
- extensible tomorrow
- compatible with all browsers
- independent of any rendering engine
- lightweight enough to run even in constrained environments

3.1 Overlay Mode — Zero Friction, Works With Any Browser

Overlay Mode enables SSM-Browse to operate **without modifying** any fundamental browser components.
No changes are required to:

- browser engines
- rendering pipelines
- resource loaders
- tab systems
- search interfaces

Users interact with the browser **exactly as they do today**:

- URLs open normally
- search results appear normally
- scrolling behaves normally
- tabs behave normally

Nothing about the visual or functional browsing experience changes.

The only addition is that **each browsing action receives an SSM-Browse envelope**, which is stored or transmitted as structural metadata.

Immediate Benefits of Overlay Mode

- **deterministic structural posture (`a`)**
- **transparent navigation categories (band)**
- **platform-independent ordering through stamps**
- **fully replayable browsing sequences**
- **audit-ready structural navigation trails**
- **predictable and verifiable tab behavior**
- **no dependence on browser internals**

Overlay Mode is ideal for:

- desktop or mobile browsers
- enterprise browsing portals
- research workflows
- compliance and observability dashboards
- educational platforms
- archival systems that require structural integrity

Three-Line Integration Contract

Overlay Mode requires only a minimal, universal integration pattern:

1. Attach envelope

```
event.env = { a, q?, band, manifest_id, stamp?, sig? }
```

2. Store envelope

```
save(JSON.stringify(event.env))
```

3. Replay envelope

```
sort_by(sequence_number)
then recompute structural values
```

This lightweight contract ensures that **any platform can adopt SSM-Browse instantly**, without disruption and without redesigning existing components.

Overlay Mode delivers structural determinism today while preserving full compatibility with future Native Mode implementations.

3.2 Native Mode — Full Structural Browsing Layer

Native Mode elevates the browser into a **deterministic symbolic engine**, where structural behavior is **first-class, mathematically governed, and fully replayable**.
In this mode, the browser **computes, maintains, and reconstructs** the structural state of the entire session using:

- the **Alignment Kernel**
- the **Envelope rules**
- the **declared manifest**

Native Mode introduces persistent, session-wide structural properties:

- **session alignment (a)** — bounded symbolic posture accumulated across actions
- **tab coherence (q)** — stability signals across parallel and sequential workflows
- **redirect stability** — deterministic posture preservation across multi-hop navigation

- **branch stability** — convergence/divergence behavior across related tabs
- **sequence drift detection** — identifying oscillatory or unstable navigation paths
- **structural resets (ZETA-0)** — deterministic stabilization when drift exceeds safe limits

In Native Mode:

- **every tab carries a structural identity**
- **every redirect is structurally interpretable**
- **every multi-step journey behaves predictably**
- **long research sessions remain stable and reproducible**
- **institutions gain provable structural trails** independent of content

Native Mode is the first environment where browsing becomes **structurally truthful by design**.

Quero (q-lane): Multi-Tab Structural Awareness

The optional **Quero lane (q)** enables deterministic multi-tab structural analysis.

It provides signals for:

- tab-to-tab coherence
- structural divergence and convergence
- workflow consistency across parallel navigation
- stability of long, deep research journeys

Quero is not behavioral tracking.
It never inspects:

- content
- intention
- identity
- meaning

It captures only **pure structural coherence** across tabs.

ZETA-0: Structural Resets for Drift Stability

ZETA-0 introduces a **neutral posture event** (`a = 0`) whenever cumulative drift crosses a manifest-declared threshold.

It is triggered during:

- deep multi-tab workflows
- heavy redirect chains

- extensive scrolling
- parallel topic expansion

ZETA-0 restores structural order without touching:

- browsing history
- visited URLs
- user actions
- search behavior

It is a **purely symbolic stabilization mechanism** that preserves structural predictability in complex, evolving navigation environments.

Native Mode Enables

- **deterministic tab flows**
- **predictable redirects**
- **stable multi-tab browsing**
- **reproducible structural categorization** (if declared)
- **safe, long-form research workflows**
- **transparent, audit-ready navigation trails** for high-trust institutions

Native Mode is the path to the first structurally truthful browser.

3.3 Choosing the Right Mode

SSM-Browse supports both present-day adoption and future deterministic browsing.

- **Overlay Mode** — ideal for immediate deployment, no redesign
- **Native Mode** — ideal for advanced environments requiring structural guarantees

A system may choose any combination:

- Overlay for standard browsing
- Native for sensitive or research workflows
- Hybrid: Overlay for general tabs and Native for structural tabs

Both modes remain:

- interoperable
- envelope-compatible
- manifest-aligned

The envelope is the common language between them.

3.4 Example — How Both Modes Interpret a Single Visit

Value (original browsing action):
Search: `"renewable energy statistics"`

(As always, the value layer remains untouched and non-interpreted.)

Overlay Mode

Overlay Mode adds symbolic structure *beside* the browsing action:
• attaches the envelope to the event
• stores alignment and coherence lanes in bounded form
• generates deterministic stamps using pure structural inputs
• enables identical replay of the event sequence on any device
• guarantees stable posture and ordering without modifying content

Structural logic remains external, minimal, and non-intrusive.
The user's action is preserved exactly as performed.

Native Mode

In Native Mode, the envelope actively participates in symbolic computation:
• updates session alignment (`a_lane`) using bounded transforms
• updates Quero coherence (`q_lane`) to reflect smoothness or shocks
• influences redirect posture and structural continuity
• contributes to cross-tab drift values for multi-tab analysis
• strengthens deterministic replay across independent sessions

The search text stays untouched.
Only the **structural interpretation** (alignment, coherence, drift, lineage) becomes deterministic and reproducible.

---

4. ZETA-0 and Quero — Structural Extensions for Browsing Stability and Coherence

Modern browsing is **non-linear by nature**.
A single user journey may involve:

- multiple tabs
- rapid switching
- deep scroll + backtracking loops
- redirects and auto-loads
- parallel research paths
- partial overlaps between tabs
- dynamic page updates
- intermittent pauses and returns

Traditional browsers treat these patterns as **noise** or incidental behavior.
SSM-Browse recognizes them as **structural signals** that must be modeled, stabilized, and replayed deterministically.

To achieve this, SSM-Browse extends the envelope with two symbolic mechanisms:

- **ZETA-0** — the structural zero-alignment event that stabilizes cumulative drift
- **Quero (𐤒)** — the multi-tab coherence lane that detects divergence and restores structural order

Together, they transform SSM-Browse from a symbolic overlay into a **full structural browsing architecture**, where even complex journeys remain:

- stable
- predictable
- deterministic
- safe
- audit-ready

These mechanisms ensure that browsing remains mathematically coherent even when user behavior is fluid, multi-threaded, or deeply exploratory.

4.1 ZETA-0 — The Zero-Event for Browsing Drift Reset

Every browsing session naturally accumulates **structural drift**.
This drift emerges from symbolic navigation patterns such as:

- long multi-page research
- rapid tab switching
- deep scroll → return loops
- multi-step redirect chains
- branching exploratory sequences
- overlapping or cascading tab workflows

Without stabilization, drift leads to:

- unpredictable tab posture
- inconsistent cross-tab relationships
- oscillatory lane behavior
- divergent multi-tab evolution
- replay sequences that differ across devices

ZETA-0 introduces a transparent, deterministic, mathematically bounded reset that restores structural stability.

What is ZETA-0?

ZETA-0 is a **neutral symbolic event** that re-centers **alignment** and **Quero coherence** when cumulative drift crosses a manifest-declared threshold.

It is:

- **neutral** (`a_lane = 0`, `q_lane = 0`)
- **bounded** (always stabilized via the kernel)
- **declared** (rules defined in the manifest)
- **transparent** (never inferred or hidden)
- **deterministic** (identical outcomes everywhere)

Nothing in the **Value Layer** changes.
Only the **structural posture** is safely re-centered.

The reset is performed using the bounded kernel:

```
lane = tanh( Σ w_i * atanh(raw_i) / Σ w_i )
```

Under ZETA-0, all `raw_i = 0`.

When ZETA-0 Is Invoked

Once manifest conditions are met, ZETA-0 ensures:

- navigation drift is neutralized
- session posture re-centers cleanly
- Quero coherence recovers to stability
- tab groups regain predictable alignment
- redirect sequences stabilize
- replayability remains exact and platform-neutral

ZETA-0 guarantees that even the most complex browsing sessions remain:

- **structurally safe**
- **predictable**
- **reproducible**
- **mathematically recoverable**

across devices, environments, versions, and time.

4.2 Why Zero-Events Are Necessary in Browsing

Browsing is inherently **recursive, branching, and non-linear**:

- a single search can open ten tabs
- one tab can branch into several parallel paths
- scroll → click → back → click sequences form structural loops
- embedded content can produce secondary browsing threads

Over time, this naturally generates **cumulative structural drift**.
Without reset mechanisms like ZETA-0, drift pushes the browsing system into instability.

Consequences of not using zero-events:

- **extreme posture accumulation** → structural bias builds over time
- **tab coherence collapses** → parallel tabs diverge unpredictably
- **replay becomes inconsistent** → structural sequences vary across platforms
- **alignment becomes unbounded** → posture signals lose interpretability
- **session stability degrades** → long workflows stop behaving deterministically

ZETA-0 acts as a **structural anchor**, restoring equilibrium by:

- clearing accumulated drift
- resetting unstable state transitions
- stabilizing branching processes
- re-centering session posture
- enforcing bounded symbolic behavior

ZETA-0 is essential for:

- **deep research workflows**
- **enterprise-grade browsing**
- **regulatory and compliance environments**
- **trusted digital archives**
- **multi-hour or multi-day navigation sessions**

It guarantees that browsing remains structurally truthful, predictable, and stable even under the most complex usage patterns.

4.3 Quero — The Structural Coherence Lane for Browsing

If the **alignment lane (a)** measures the *posture* of individual browsing events,
the **Quero lane (q)** measures *coherence*: the stability, harmony, and divergence patterns across the browsing journey.

Quero never interprets meaning or content.
It evaluates **structural behavior only**.

Quero tracks:

- structural evolution of parallel tabs
- stability or instability in sequential navigation
- divergence patterns across multi-tab research paths
- consistency and integrity of redirect chains
- structural rhythm and drift across the session

Quero is **not behavioral analytics**.
It is a **symbolic structural signal** that captures how stable, predictable, or divergent a browsing workflow becomes.

It enables deterministic multi-tab awareness while maintaining complete neutrality, privacy, and non-semantic operation.

4.4 What Quero Enables in Browsing

With Quero (q), the browser gains deterministic structural intelligence that has never existed in traditional browsing.

Quero enables the browser to:

- detect unstable or diverging tab groups

- identify when a browsing session branches beyond safe coherence
- detect irregular or oscillatory redirect loops
- highlight structurally unstable navigation sequences
- offer optional structural insights to advanced users
- support safe, predictable workflows for enterprises and institutions

All of this happens **without reading URLs or page content**.
Quero operates on pure structure — a symbolic understanding of navigation, not the meaning behind it.

4.5 ZETA-0 + Quero Together

ZETA-0 and Quero operate as a **closed structural feedback loop** that maintains browsing stability in real time.

Their roles are complementary:

- **Quero (ϙ)** continuously evaluates structural coherence
  Detects divergence, instability, or oscillatory behavior in navigation patterns.
- **ZETA-0** intervenes precisely when needed
  Neutralizes accumulated drift and restores structural balance through a deterministic zero-alignment event.

Together, they ensure that:

- tab groups maintain coherent structure
- long, multi-step research sessions remain stable
- navigation paths follow predictable symbolic transitions
- replay and audit remain identical across devices and platforms
- structural posture does not accumulate uncontrolled bias

This transforms browsing from an ad-hoc behavioral sequence into a **mathematically governed structural system** with deterministic rules, bounded signals, and transparent corrections.

4.6 Example

Scenario
A 2-hour research session involving:

- multiple search iterations
- 15 open tabs
- deep scroll exploration
- redirect chains
- parallel investigation of related subtopics

As the session progresses:

1. **Quero detects rising divergence**
   Parallel tabs begin to drift apart structurally as posture oscillations accumulate.
2. **ZETA-0 is invoked**
   A neutral zero-alignment event is inserted because drift crossed the manifest-defined threshold.
3. **Session posture resets**
   The alignment lane (`a`) returns to a stable, bounded center.
4. **All open tabs regain coherence**
   Structural consistency is restored across diverging tab paths.
5. **Replay becomes deterministic again**
   Any system can reconstruct the browsing sequence identically.

Nothing about the content changes.
Nothing is ranked, boosted, interpreted, or suppressed.

This is **pure symbolic correction**:
transparent, deterministic, and structurally truthful.

4.7 Live Structural Drift Demonstration (SSM-Browse v4)

(Authoritative Real Output — DevTools-Class Embedded Analyzer)

Unlike traditional documentation that uses simulated or conceptual flows, this section presents **actual**, **fully deterministic output** generated by **SSM-Browse v4**, the DevTools-Class embedded symbolic analyzer.

This is **real execution**, produced by the single-file v4 engine (`ssm_browse_full_v4.html`), running:

- with **zero network**,
- **zero libraries**,
- **zero external dependencies**,
- and **100% local deterministic logic**.

It demonstrates multi-tab structural evolution, kernel-driven lane computation, deterministic stamping, Quero behavior, and cross-tab drift mechanics exactly as the alignment kernel performs them internally.

Scenario

The following sequence was executed interactively on the live v4 engine:

1. **Open Tab × 4**
2. **Random navigation events** across tabs
3. **Scroll events** on selected tabs
4. **Idle events** added to complete the structural sequence

No URLs, no DOM, no semantics, no rendering.
Only **symbolic structure**.

Live Output (Direct From SSM-Browse v4)

Tabs

- tab_1
- tab_2
- tab_3
- tab_4

Lanes (a_lane / q_lane)

These are the fused alignment + coherence values after kernel transformation:

```
tab_1  →  a=0.073   q=0.004
tab_2  →  a=0.000   q=0.045
tab_3  →  a=0.056   q=0.083
tab_4  →  a=0.162   q=0.048
```

The gradual divergence in `q_lane` reflects cross-tab structural exploration patterns.

Event Log (with Deterministic Stamps)

Every event receives a stamp computed solely from symbolic lineage:

```
tab_1 | open_tab   | S826796
tab_1 | navigate   | S153960
tab_1 | scroll     | S694417
tab_1 | idle       | S253575

tab_2 | open_tab   | S991220

tab_3 | open_tab   | S737124
tab_3 | navigate   | S367353
```

```
tab_4 | open_tab   | S389162
tab_4 | scroll     | S446888
tab_4 | idle       | S336066
```

Even with varied event order, **stamp stability is absolute** across all devices.

Drift Matrix

Cross-tab symbolic drift computed by deterministic structural comparison:

```
           tab_1   tab_2   tab_3   tab_4
tab_1        0     0.408   0.742   0.972
tab_2      0.428     0     0.930   0.703
tab_3      0.102   0.053     0     0.105
tab_4      0.513   0.011   0.206     0
```

Interpretation:

- **tab_1 ↔ tab_4 = 0.972** → Highly divergent navigation
- **tab_2 ↔ tab_3 = 0.930** → Parallel but structurally drifting research
- **tab_3 ↔ tab_4 = 0.206** → Mild drift (coherent tab family)
- Diagonal zeros reflect required self-identity

This is mathematically real, not conceptual.

What This Demonstration Proves

1. Deterministic Behavior

Running this demo on:

- Windows
- macOS
- Linux
- offline devices
- air-gapped systems

…produces **identical stamps, identical drift, identical lanes**.

2. Product-Agnostic Architecture

Because the engine is purely symbolic:

- **no URL parsing**
- **no DOM hooks**
- **no browser APIs**

- **no rendering**

It works identically in any standards-compliant browser.

3. Structural Drift Detection Works

The drift matrix evolves solely from the event structure —
not from randomness, timing, content, or environment.

4. Alignment Kernel is Stable & Bounded

All lane values remain safely within **(−1, +1)** due to:

```
a = tanh( Σ w_i * atanh(raw_i) / Σ w_i )
```

5. v4 Delivers Professional-Grade Observability

With only **~5 KB of code**, SSM-Browse v4 offers:

- multi-panel DevTools-style visualization
- live event controls
- real-time posture evolution
- instant drift computation
- deterministic structural replay

This capability is unprecedented at this footprint.

Summary

This section is **not illustrative**.
It is **authoritative**, documenting a **real execution** of the symbolic browser engine.

It proves that:

- SSM-Browse is mathematically grounded
- alignment, coherence, and drift are observable
- all behavior is deterministic and reproducible
- the engine is product-agnostic
- the symbolic envelope model works in live browsing contexts
- ~5 KB is enough to deliver structural truth at professional depth

This output serves as **official verification** of **SSM-Browse v4** correctness.

---

5. Safety & Licensing

SSM-Browse is engineered with uncompromising safety boundaries.
Its purpose is **structural truth**, not influence — and its architecture enforces this by design.

SSM-Browse never reads, interprets, learns from, or reacts to page content.
It never profiles users or alters browsing outcomes.
All it provides is a **transparent, deterministic structural envelope** for navigation events.

This section defines:

- **what SSM-Browse explicitly allows**
- **what it strictly forbids**
- **how its open-standard license protects neutrality and public trust**

These rules are essential for global adoption across research, enterprise, education, compliance, and long-term digital archiving.

5.1 Safety Principles (Non-Semantic, Zero-Inference, Zero-Manipulation)

SSM-Browse is built entirely on **structural mathematics**, not behavioral or semantic interpretation.

It processes only:

- **posture** `a`
- **coherence** `q`
- **manifest-declared structural rules**

Nothing else.
The framework operates under **five non-negotiable safety principles**, each foundational to its open-standard mission.

---

1. No Interpretation of Meaning

SSM-Browse **never performs semantic analysis**.
It does **not** inspect, read, infer, or classify any aspect of the browsing value layer.

It never interprets:

- search keywords
- URLs

- page titles
- page content
- scroll behavior
- user goals
- inferred preferences
- viewing habits
- interaction patterns

There is **zero content intelligence** and **zero behavioral inference**.

SSM-Browse processes only:

- **posture (`a`)**
- **coherence (`q`)**
- **manifest-defined rules**

This eliminates:

- personalization
- content inference
- predictive ranking
- behavioral scoring
- hidden bias mechanisms

Browsing remains **non-semantic**, **non-interpretive**, and **mathematically neutral**.

---

2. No Influence on Ranking, Visibility, or Personalization

The envelope is **structural metadata**, not a scoring or ranking system.
It **never affects**:

- search rankings
- result visibility
- ordering
- boosts or demotions
- suggestion patterns
- recommendation systems
- autocomplete behavior

There are **no hidden calculations**.

Any structural band must be:

- **declared**
- **visible**
- **manifest-driven**
- **identical for all users**

This guarantees:

- **fairness**
- **neutrality**
- **verification**
- **reproducibility**

Browsing becomes **immune to silent manipulation**.

---

3. No Content Modification

The **Value Layer** is sacrosanct.
SSM-Browse never alters:

- URLs
- content
- metadata
- scripts
- cookies
- redirects
- search results
- page rendering
- page logic
- network behavior

The value layer remains:

- **100% original**
- **100% unmodified**
- **100% replayable**
- **100% neutral**

The envelope sits **beside** the action — never inside it, never changing it.

---

4. Transparent, Deterministic Structural Behavior

Every symbolic outcome is:

- **mathematically defined**
- **deterministic**
- **platform-independent**
- **reproducible across devices**
- **identical across programming languages**
- **auditable and visible**
- **declared in the manifest**

Posture `a`, coherence `q`, structural bands, drift rules, Quero behavior, and ZETA-0 triggers are all **explicitly defined**.

There is **no black-box logic** anywhere.

If a rule exists, it is:

- **declared**
- **inspectable**
- **identical across all systems**

This is the foundation of **verifiable structural truth**.

---

5. Optional Stamp Chain — Never Identity, Never Tracking

The stamp mechanism uses:

```
stamp = sha256(prev || payload)
```

It is **ordering-only** — not identity, not tracking.

The stamp never encodes:

- user identity
- session identifiers
- personal attributes
- behavioral correlations

It cannot be used for:

- profiling
- authentication
- targeted ranking
- user reconstruction
- fingerprinting

The stamp serves only as:

- a **deterministic sequence anchor**
- a **tamper-visible replay mechanism**
- a **content-agnostic ordering tool**

This ensures:

- **privacy**
- **neutrality**
- **anonymity**
- **long-term audit integrity**

5.2 What SSM-Browse Forbids

To preserve neutrality, structural truth, and long-term public trust, SSM-Browse explicitly prohibits certain uses and practices.
These prohibitions are absolute, non-negotiable, and apply to all implementations.

Prohibited Uses

SSM-Browse must **never** be used to derive, infer, score, or correlate anything about user identity or behavioral meaning.

It prohibits:

- profiling or inferring personal traits
- correlating posture (`a`) or coherence (`q`) with demographics or identity
- tracking browsing history for behavioral analytics
- deriving preferences, intentions, sentiments, or motivations
- merging structural envelopes with private identifiers
- training predictive models using `a` or `q` to infer behavior
- applying hidden weighting rules to influence browsing outcomes
- manipulating search results, rankings, or suggestions using envelope values
- using structural bands to punish, reward, restrict, filter, or redirect content

The envelope exists solely as **structural metadata** — never as input for behavioral interpretation.

Prohibited Technical Practices

SSM-Browse must never incorporate or enable:

- undisclosed inference mechanisms
- sentiment scoring
- contextual or semantic interpretation
- computational fingerprinting
- covert personalization
- hidden browser biases
- silent rewriting of structural rules
- undeclared ranking adjustments
- envelope-based behavioral prediction
- content-dependent structural decisions

Any structural rule must be **declared**, **visible**, and **replayable**.

These prohibitions ensure that structural browsing remains:

- fair
- neutral
- mathematically transparent
- universally reproducible
- completely free of manipulation

SSM-Browse protects the user, the developer, and the institution by eliminating ambiguity, bias, and silent influence.

5.3 Licensing (Short Summary + Ecosystem Clarity)

SSM-Browse is released as a **fully Open Standard**, enabling any browser, platform, tool, or system to adopt the structural envelope without friction, permission, or proprietary constraints.

This licensing model is intentionally different from other Shunyaya projects that use CC BY 4.0, such as the core symbolic mathematics layers. Those projects require attribution because they define foundational mathematical constructs. SSM-Browse does not — because it is designed as a **public structural protocol** that must remain universal, neutral, and freely deployable everywhere.

License Type

Open Standard (as-is, observation-only, no warranty)

Meaning:
• no mandatory attribution requirements
• optional reference to the concept name for clarity of lineage
• no implementation restrictions
• no exclusivity or lock-in
• no registration
• no fees of any kind
• no legal barriers to modification or redistribution

This stands in contrast to CC BY 4.0 projects in the Shunyaya ecosystem, where explicit attribution is required.
SSM-Browse removes that requirement to enable frictionless, worldwide adoption and to ensure that the symbolic envelope becomes a **universal structural layer** for browsing.

Permitted

You may freely:

- use SSM-Browse in personal, institutional, research, or commercial systems
- embed envelopes beside any browsing actions
- build overlays, extensions, native integrations, proxies, or developer tools
- modify, extend, repackage, or redistribute original or modified versions
- integrate posture (`a`) and coherence (`q`) computation into other symbolic layers
- create visualizations, drift matrices, audits, or replay tools

No part of the original browsing event (the Value layer) is restricted.
All implementation choices remain yours.

Conditions

To maintain structural integrity and global trust:

• attribution is **optional**, but when describing your implementation, you may reference the concept name for clarity:
"Shunyaya Symbolic Mathematical Browse (SSM-Browse)"

• envelopes must remain transparent, inspectable, and reproducible
• manifests must be declaratively published within their scope
• no hidden logic may operate behind envelope fields
• posture (`a`) and coherence (`q`) rules must remain mathematically explicit
• no semantic or contextual interpretation may be added
• the Value layer must remain **100% untouched**
• envelope fields must never be repurposed for ranking, profiling, inference, or moderation
• any rule changes must be declared in a manifest or structural appendix

These conditions ensure that all SSM-Browse implementations remain deterministic, fair, neutral, and structurally truthful.

Prohibited

To preserve neutrality and prevent misuse, SSM-Browse must not be used for:

Structural Manipulation

- using envelope values to influence ranking, ordering, or visibility
- applying posture or coherence as signals to boost, demote, or filter content

Semantic or Behavioral Interpretation

- interpreting sentiment, intent, topic, emotion, or preference
- deriving meaning from URLs, content, or keywords
- combining envelope values with contextual inference

Identity or Profile Construction

- correlating `a` or `q` with identity or demographic attributes
- predicting behavior, preference, or personality
- linking envelopes to identifiable user histories

Machine-Learned or Predictive Inputs

- sourcing posture or coherence signals from ML inference
- embedding classifiers or predictive features behind envelope computation

Undeclared Logic

- modifying posture rules without publication in a manifest
- adding hidden weighting mechanisms
- obscuring or rewriting envelope logic

These prohibitions ensure total neutrality, mathematical consistency, and platform-independent reproducibility.

Disclaimer

SSM-Browse is provided **as-is** and intended for:

- research
- observation
- structural browsing
- transparency
- reproducibility
- auditability

It must not be used for safety-critical or high-risk decision-making.
Responsibility for deployment rests with the implementer.

Intent (Why SSM-Browse Uses Open Standard Licensing)

The Open Standard model exists to:

- enable barrier-free global adoption
- ensure structural browsing remains independent of commercial interests
- prevent hidden reinterpretation or proprietary distortion
- support interoperability across devices and architectures
- keep posture, coherence, and reset logic mathematically explicit
- preserve neutrality and public trust for decades

In practice:

- you may integrate SSM-Browse anywhere, including in commercial products
- you may innovate and extend the model
- but structural transparency must be preserved
- users and auditors must always be able to reproduce envelope outputs exactly

---

Practical Attribution (Lightweight and Optional)

Recommended (but not required) phrasing:

"This system implements the Shunyaya Symbolic Mathematical Browse (SSM-Browse) concepts."

This maintains clarity of lineage while keeping adoption globally frictionless.

---

6. Developer Integration (High-Level Overview)

SSM-Browse is designed for **instant integration** into any existing browsing environment.
It does not modify rendering, networking, JavaScript execution, or page content.
Instead, it adds a **structural envelope**—a compact, deterministic, and fully transparent record that travels beside each browsing action.

This makes the model:

- light-weight
- portable
- non-intrusive
- easy to embed in any architecture

Integration is defined by four minimal components:

1. Envelope Generation

Compute posture (`a`), coherence (`q`), band, and optional stamp.

2. Manifest Lookup

Retrieve the declared rulebook for clamping, weighting, thresholds, resets, and replay logic.

3. Optional Stamp Chain

Produce transparent, tamper-visible ordering of navigation events.

4. Minimal API Surface

Provide a simple method for attaching, storing, retrieving, and replaying structural envelopes.

No machine learning.
No content interpretation.
No personalization.
Just deterministic structural metadata that behaves identically everywhere.

This allows SSM-Browse to integrate with:

- browsers
- extensions
- proxies
- research tools
- enterprise observability systems
- institutional audit frameworks

without disrupting the underlying browsing model.

6.1 Envelope Generation

Envelope generation is the core operation of SSM-Browse.
It produces the **structural truth** of each browsing event while leaving the **Value layer completely untouched**.

A platform generates the envelope using the following components:

Inputs

- **`a_raw`** — declared posture signal
- **`q_raw`** — optional coherence input (Native Mode only)
- **manifest-declared rules** for:
  - clamping
  - weighting
  - Quero behavior
  - band thresholds
  - structural reset triggers (**ZETA-0**)

All inputs must be **declared**, **transparent**, and **non-inferred**.

Alignment Kernel (Core Formula)

The kernel transforms `a_raw` into a bounded, deterministic posture signal:

```
a_c   = clamp(a_raw, -1+eps_a, +1-eps_a)
u     = atanh(a_c)
a_out = tanh(U / max(W, eps_w))
```

Where:

- **`a_c`** – clamped posture
- **`u`** – rapidity representation
- **`U`** – cumulative rapidity
- **`W`** – declared weight
- **`eps_w`** – prevents division instability

This computation is:

- deterministic
- stable
- mathematically explicit
- reproducible across all platforms

Band Assignment

Bands are derived strictly from **manifest-defined thresholds**, such as:

- `NAVIGATE`
- `NEUTRAL`
- `RESTRICT`
- `VERIFY`
- `WARN`

Bands are *structural categories*, not semantic interpretations.

Optional Quero (q-lane)

If coherence tracking is enabled:

- `q_raw` is clamped and transformed using manifest rules
- `q_out` becomes the structural coherence signal
- no content, behavior, or meaning is inspected

Optional Stamp (Recommended)

```
stamp = sha256(prev || payload)
```

This provides:

- deterministic ordering
- tamper visibility
- cross-device replay consistency

The stamp:

- **never encodes identity**
- contains no URLs, content, or behavioral attributes

Minimal Envelope Structure

```
{
  a: a_out,
  q: q_out?,              // optional coherence (Quero)
  band: <label>,
  manifest_id: <id>,
  stamp: <optional>,
  sig: <optional>         // optional authenticity seal; never structural
}
```

Properties

The envelope remains:

- timezone-independent
- platform-neutral
- architecture-agnostic
- fully reproducible

- mathematically explicit
- deterministic across all languages and devices

Envelope Generation is the backbone of structurally truthful browsing.

6.2 Manifest Lookup

The manifest is the **rulebook** that governs how structural posture, coherence, and stability are computed.
SSM-Browse requires *all* structural logic to be **declared**, **visible**, and **non-inferred**.

A manifest typically defines:

- posture assignment rules
- clamping parameters
- Quero semantics (if enabled)
- band thresholds and structural categories
- weighting logic for alignment
- replay specifications
- ZETA-0 reset triggers
- stamp chain guidelines (optional but recommended)

Nothing inside the manifest may be hidden or dynamically inferred.

Developer workflow:

1. **load** the manifest
2. **apply** the declared rules exactly
3. **compute** the envelope

This guarantees that every system — regardless of device, OS, language, or architecture — produces **identical structural outputs** for the same inputs.

SSM-Browse ensures that manifests become a **transparent, reproducible foundation** for structural browsing.

6.3 Optional Stamp Chain (Strongly Recommended)

The **stamp chain** provides **tamper-visible**, **deterministic** ordering of browsing events.
It is optional, but **strongly recommended** for:

- auditing
- replay
- compliance

- multi-device consistency
- long-session reconstruction

The formula for the k-th event is:

```
stamp_k = sha256(stamp_(k-1) || payload_k)
```

Where:

- **`stamp_(k-1)`** — previous stamp in the chain
- **`payload_k`** — symbolic browsing event (structural, non-semantic)

Because timestamps are not used, the stamp chain is:

- **timezone-independent**
- **region-independent**
- **clock-independent**
- **fully deterministic across all devices**

The stamp chain enables:

- **provable navigation ordering**
- **deterministic replay** across devices and environments
- **transparent integrity verification**
- **drift and tamper detection**
- **complete structural session reconstruction**

Important:
Stamps never identify users.
They carry **no personal data**, **no content**, **no URLs**, and **no behavioral attributes**.
They provide only a **structural ordering signal**, nothing more.

6.4 Minimal API Surface

SSM-Browse is intentionally simple to implement.
A complete integration requires only **four functions**:

1. **generate_envelope(event, manifest)**
   Computes posture, coherence, band, and optional stamp.
2. **attach_envelope(event, envelope)**
   Binds structural metadata to the browsing action.
3. **replay_session(records)**
   Reconstructs structural navigation using envelopes and manifest logic.
4. **verify_stamp_chain(records)**
   Validates the integrity and ordering of the session.

Everything else — visualization panels, drift matrices, event logs, developer dashboards, or research tools — is optional and outside the core specification.

The API remains minimal, deterministic, and platform-neutral, enabling SSM-Browse to function in:

- native browsers
- extensions
- proxies
- archival systems
- institutional audit platforms

with zero dependency on machine learning, semantics, or proprietary components.

6.5 Storage & Serialization

SSM-Browse does not impose any storage format or backend requirements.
Envelopes may be stored in **any medium**, as long as their structural fields remain **exact and unmodified**.

Common storage formats:

- JSON objects
- sidecar files written per event
- browser or extension metadata blocks
- structured log entries
- inline combined objects: `{ value, envelope }`

Developers may store envelopes:

- in memory
- in local application data
- in browser storage
- in secure institutional logs
- in long-term archival systems

The only invariant is that structural fidelity must be preserved.

Because envelope computation is deterministic and manifest-driven, **replay always produces identical outcomes**, regardless of storage format or location.

6.6 Overlay & Native Integration Paths

SSM-Browse provides two equally supported integration paths.

Overlay Mode (zero friction)

- no engine or rendering redesign
- envelopes simply attach to existing browsing events
- ideal for:
  - research prototypes
  - compliance dashboards
  - enterprise observability
  - browser extensions
  - educational tools

Overlay Mode enables platforms to adopt SSM-Browse immediately.

Native Mode (structural browsing engine)

- tabs maintain structural state
- redirects and navigation chains become deterministic
- Quero lanes provide multi-tab coherence tracking
- session alignment becomes stable over long workflows
- ZETA-0 prevents uncontrolled structural drift

Native Mode unlocks the complete symbolic architecture of SSM-Browse, enabling deterministic browsing across entire sessions.

Interoperability

Both modes:

- use the same envelope
- use the same manifest
- use the same alignment kernel
- produce identical structural semantics

A platform may mix both modes, using Overlay Mode for standard browsing and Native Mode for trusted, research, or enterprise workloads.

6.7 Development Philosophy

SSM-Browse is built on five foundational principles:

- **lightweight** — designed to be micro-thin and negligible in overhead
- **deterministic** — all core behavior follows explicit mathematical rules
- **transparent** — every structural transformation is inspectable and reproducible
- **non-semantic** — no interpretation of content, meaning, emotion, or intent
- **safe by design** — envelopes cannot be used for ranking, profiling, or influence

Additional goals:

- globally interoperable across devices, architectures, and languages
- trivial to integrate using a minimal four-function API
- compatible with existing browsers, proxies, and research pipelines
- capable of long-term archival and institutional audit

SSM-Browse does not predict, judge, or infer.

It encodes **pure structural mathematics** for browsing — nothing more, nothing less.

---

7. Verification & Trust

SSM-Browse is designed for environments where trust **cannot depend on black-box logic**, opaque personalization, or vendor-specific interpretation.
Every component of the standard is:

- deterministic
- reproducible
- portable
- structurally explicit

Verification in SSM-Browse never requires:

- reading content
- inspecting URLs or search terms
- tracking users
- linking identities
- inferring meaning

Verification requires only:

- the **envelope**
- the **manifest**
- the **alignment kernel formula**
- the **optional stamp chain**

With these, any system can reconstruct structural behavior **exactly**, across devices, languages, and architectures — today or decades in the future.

This shifts browsing from an interpretive system to a **mathematically verifiable structure**.

7.1 Deterministic Replay

Deterministic replay is one of the core guarantees of SSM-Browse.

A browsing session containing envelopes can be replayed to regenerate:

- exact posture values (`a_out`)
- exact structural bands
- exact Quero coherence (`q_out`), when enabled
- exact ZETA-0 reset placements
- exact navigation ordering (with stamps)
- exact structural tab and redirect states

Replay must always yield the same results, regardless of:

- device
- programming language
- operating system
- browser engine
- hardware architecture

If two systems produce different results, **one of them is not applying the manifest correctly**.

Deterministic replay turns structural browsing into an **audit-ready**, provable, mathematically governed process.

7.2 Clamp & Safety Tests

Because posture computation uses clamping and rapidity transforms, developers can verify correctness using purely mathematical tests.

Core formulas:

```
a_c = clamp(a_raw, -1+eps_a, +1-eps_a)
u   = atanh(a_c)
```

These allow any system to confirm that:

- no overflow occurs
- no numeric instability arises
- no undefined or indeterminate states appear
- `a_out` always remains strictly within `(-1, +1)`
- extreme or repetitive navigation patterns remain structurally safe
- posture lanes do not diverge, saturate, or accumulate bias

This ensures that SSM-Browse maintains **bounded structural behavior**, even in long, multi-tab, or deeply nested navigation flows.

7.3 Stamp Chain Integrity

When stamps are used, navigation ordering becomes **tamper-visible** and **mathematically verifiable**.

The deterministic rule is:

```
stamp_k = sha256(stamp_(k-1) || payload_k)
```

Any deviation from this sequence immediately exposes attempts to alter structural history.

Detectable tampering includes:

- reordering events
- deleting events
- inserting events
- altering payloads
- modifying structural fields inside the envelope
- replaying only part of a session

The stamp chain enables:

- deterministic session integrity checks
- cross-device auditing
- transparent history verification
- trusted logs without centralized control
- long-term archival of structural navigation sequences

No signatures, identity markers, or personal data are involved.
Stamps provide **pure structural truth**, not authentication or user tracking.

7.4 Manifest Consistency

Every SSM-Browse manifest is:

- public
- human-readable
- deterministic
- versioned only when changed
- free from hidden parameters or adaptive logic

Verification ensures that:

- band thresholds exactly match declared boundaries
- Quero semantics match the declared structural rules
- clamp parameters match specification
- the alignment kernel is applied precisely as defined
- ZETA-0 insertion rules follow the manifest without deviation

If a manifest is **missing, altered, or incomplete**, envelopes must not be interpreted.

This prevents:

- silent manipulation
- implicit reinterpretation
- vendor-specific extensions
- ambiguity between implementations

A manifest-driven system guarantees **universal interpretability** and **global consistency**.

7.5 Alignment Kernel Consistency

The alignment kernel can be validated through four deterministic steps:

1. **Apply clamp**
2. **Apply rapidity transform (`atanh`)**
3. **Apply declared cumulative or weighted combine (`U/W`)**
4. **Apply `tanh` to recover posture (`a_out`)**

If a system reproduces the reference posture output vector from known inputs, the implementation is correct.

Alignment kernel verification eliminates:

- platform-specific bias
- hidden adjustments

- inconsistent ranking or prioritization
- unpredictable variance in posture computation

This ensures that structural truth is not a matter of interpretation — it is a matter of **mathematics**.

7.6 Quero Coherence Checks (Optional)

When enabled, Quero (`q`) follows the same structural reproducibility model as posture:

- clamp
- rapidity transform
- weighted or streaming combine
- bounded recovery using `tanh`

These checks guarantee that multi-tab coherence remains:

- stable
- deterministic
- safe
- identical across platforms and architectures

Quero coherence captures **structural relationships**, not behavior, preference, or intent.

7.7 Verification Packs

To streamline adoption and auditing, platforms may distribute complete, public verification packs containing:

- sample envelopes
- golden vectors for posture
- golden vectors for Quero
- deterministic replay test sets
- example ZETA-0 placements
- serialized stamp chain sequences
- reference manifests

These allow:

- developers
- research teams
- institutions
- auditors
- compliance groups

to validate SSM-Browse integrations quickly and confidently.

Verification packs contain **no personal data**, only **structural metadata**.

7.8 Trust Philosophy

Trust in SSM-Browse emerges from:

- absolute transparency
- deterministic structural outcomes
- zero hidden logic
- zero personalization
- zero inference
- zero behavioral weighting
- a single universal alignment kernel
- strict structural neutrality

Trust is not created through authority, policy, or centralized control.

Trust is created by mathematics.

SSM-Browse elevates the global trust baseline by ensuring that browsing interactions remain:

- structurally truthful
- fully auditable
- reproducible everywhere
- safe by design

This transforms browsing from an opaque behavioral system into a **transparent structural protocol**.

7.9 Replay Integrity Failure Example (Optional Diagnostic Pattern)

Deterministic replay guarantees that any sequence of browsing events can be re-constructed exactly from envelopes, lane values, stamps, and the declared manifest.
A replay failure does **not** indicate user error or semantic mismatch — it indicates a structural inconsistency that must be surfaced transparently.

This diagnostic example illustrates how replay failure is detected, localized, and logged.

Example Scenario

A browsing sequence is recorded as:

```
E1 → E2 → E3 → E4
```

Where each `E*` contains:

```
a_lane
q_lane
band
manifest_id
stamp
```

During replay, the verification engine reconstructs:

```
E1' → E2' → E3' → E4'
```

A replay failure occurs when:

```
E3 ≠ E3'
```

and the mismatch falls into one of the manifest-defined failure classes.

Failure Class: Stamp Chain Divergence

If the stamp chain is deterministic, a replay mismatch may be flagged as:

```
stamp_mismatch:
    expected = sha256(E2 | manifest | …)
    got      = sha256(E2 | manifest | <different inputs>)
```

This indicates that one of the following occurred:

• the original structural inputs were altered
• the manifest used for replay differs from the original
• a non-deterministic component was introduced upstream
• a missing or injected event invalidated the chain

The envelope itself remains untouched; only structural checks have failed.

Failure Class: Alignment Lane Violation

A mismatch in posture evolution is flagged as:

```
a_lane_violation:
    expected = clamp( f(E2, manifest) )
    got      = <value outside expected deterministic range>
```

This means:

• the kernel parameters or thresholds differ between recording and replay
• the alignment evolution rule was modified
• an unbounded posture update was attempted outside the manifest
• clamp application is inconsistent across environments

Failure Class: Quero Coherence Drift

A coherence mismatch appears as:

```
q_lane_violation:
    expected = clamp( g(E2, manifest) )
    got      = <non-matching coherence progression>
```

This signals:

• tab-to-tab sequences diverged
• the replay manifest applies different drift constraints
• a ZETA-0 reset occurred in one system but not the other

Failure Class: Missing Event

If replay attempts to reconstruct:

```
E1 → E2 → E4
```

but the original chain included `E3`, the engine flags:

```
missing_event:
    expected = E3
    got      = <gap detected>
```

This ensures no silent removal of structural history.

Failure Class: Injected Event

If an extra event appears:

```
E1 → E2 → E2a → E3 → E4
```

but never existed in the original chain:

```
E2a:
    band != expected
    stamp != expected
    manifest_id != declared
```

The engine flags it as:

```
injected_event_detected
```

because the deterministic stamp chain cannot be satisfied.

Manifest-Level Reporting

Replay failure never interprets content.
It produces a purely structural report:

```
replay_status = FAILED
reason        = <failure_class>
location      = E3
details       = { expected: <...>, got: <...> }
```

This ensures:

• transparency
• determinism
• auditability
• zero inference
• zero user profiling
• zero dependency on browsing content

Replay failure is therefore a **safety surface**, not a semantic surface.

Purpose of This Diagnostic Pattern

This example exists to help:

• implementers
• auditors

• regulators
• developers

understand how SSM-Browse surfaces inconsistencies **strictly at the structural level**, without involving identity, personalization, or semantics.

It does not influence ranking, visibility, search behavior, or navigation.
It exists purely to ensure that the structural chain remains:

• deterministic
• tamper-visible
• platform-neutral
• replayable across time and environment

---

8. Closing Summary & Adoption

Modern browsing carries a fundamental structural flaw:

- the **browsing action** is visible,
- but the **structural interpretation** of that action is hidden.

As a result:

- search results shift silently
- suggestions evolve unpredictably
- ranking logic cannot be explained
- redirect paths differ across devices
- tab flows drift in non-deterministic ways
- institutions cannot reliably reconstruct navigation
- users cannot verify why one result appeared instead of another

SSM-Browse corrects this structural opacity.
It introduces a transparent, deterministic symbolic envelope for every browsing event—
a universal structural layer that restores clarity, fairness, and reproducibility without modifying content or behavior.

- The **Value layer** (the browsing action) remains pure.
- The **Envelope layer** makes its structural treatment consistent, replayable, and device-independent.

This dual-layer architecture enables five foundational corrections:

1. **No hidden algorithms**
   Structural posture `a` is declared, not inferred.
2. **No platform-dependent distortion**
   Same visit → same structural outcome everywhere.

3. **No unpredictable structural drift**
   All transformations follow one transparent mathematical kernel.
4. **No silent reordering**
   When stamps are used, navigation sequences become provably ordered.
5. **No semantic bias**
   SSM-Browse never interprets URLs, keywords, content, or intent.

Browsing becomes:

- fair
- deterministic
- audit-ready
- mathematically grounded
- safe by design
- compatible with all existing systems

SSM-Browse works immediately in **Overlay Mode** and may be extended to **Native Mode** if a platform elects to treat structure as a first-class concept.

This makes the framework suitable for:

- browser developers
- enterprises
- research institutions
- regulatory bodies
- observability and compliance tools
- archival systems
- new browser architectures
- high-integrity or long-term reproducibility environments

SSM-Browse provides the missing structural foundation—
a neutral, mathematical, reproducible framework describing how browsing actions are structurally treated.

The modern web requires a model that is:

- fair
- deterministic
- universally reproducible

SSM-Browse supplies that structural model.

Adoption

Adoption of SSM-Browse involves three transparent steps:

1. **Generate an envelope using the alignment kernel**

```
a_c   = clamp(a_raw, -1+eps_a, +1-eps_a)
u     = atanh(a_c)
a_out = tanh(U / max(W, eps_w))
```

2. **Attach the envelope to browsing events (Overlay Mode)**
   No redesign, no disruption—simply place the envelope beside the action.
3. **Publish a manifest declaring the structural rules**
   (clamps, weights, bands, Quero semantics, optional stamps)

Everything beyond this is optional.

With these steps, browsing becomes:

- predictable
- inspectable
- structurally aligned
- globally interoperable
- free of personalization bias
- suitable for research, observability, and compliance workflows

SSM-Browse is not an enhancement.
It is a **structural correction**—a foundational upgrade that restores mathematical truth and neutrality to how the web handles browsing actions.

The standard is **open, neutral, minimal**, and available for use today.

Appendix A — Alignment Kernel Specification (Authoritative Reference)

The Alignment Kernel is the mathematical heart of SSM-Browse.
It defines how a browsing action is transformed into a **structural posture value** that is:

- deterministic
- reproducible
- platform-independent
- non-semantic
- bounded within (-1,+1)

This appendix formalizes the kernel in a complete and implementation-ready manner.

A.1 Kernel Overview

Every envelope computes:

```
a_out  ←  alignment_kernel(a_raw, manifest)
```

Where:

- `a_raw` is the declared input from the manifest's assignment rule
- `alignment_kernel` is the 4-step symbolic process
- `a_out` is the final bounded posture included in the envelope

No interpretation of meaning, keywords, content, URLs, identity, or behavior is involved. The kernel transforms structure only.

A.2 Mathematical Steps (Canonical Sequence)

The alignment kernel consists of **four mandatory steps**.

Step 1 — Clamp

```
a_c = clamp(a_raw, -1 + eps_a, +1 - eps_a)
```

Where:

- `eps_a` is a very small positive constant (declared in the manifest)
- ensures inputs never hit ±1 exactly
- prevents infinite rapidity during atanh()

Properties:

- `a_c` always satisfies `-1 < a_c < +1`
- identical across languages (JS, Python, Rust, C++, etc.)

Step 2 — Rapidity Transform

```
u = atanh(a_c)
```

Interpretation:

- Converts posture into hyperbolic rapidity (unbounded)
- Makes sequential combinations linearly stable
- Ensures structural effects compose properly across long browsing sessions

Properties:

- `u` is unbounded (−∞, +∞)
- well-defined for all clamped values

Step 3 — Weighted Combination

The manifest defines:

- a cumulative term `U`
- a cumulative weight `W`

After processing the new event:

```
U_new = U_prev + w * u
W_new = W_prev + w
```

Where:

- `w` is a positive weighting factor declared in the manifest
- `eps_w` ensures weight is never zero
- weights may encode simple structural rules (not semantics)

This produces a normalized rapidity:

```
r = U_new / max(W_new, eps_w)
```

Step 4 — Recover Posture

```
a_out = tanh(r)
```

Properties:

- `a_out` ∈ (-1, +1)
- bounded and continuous
- smooth under high-frequency browsing actions
- identical across systems and languages

`a_out` is the final structural posture published in the envelope.

A.3 Kernel Pseudocode (Reference Form)

```
function alignment_kernel(a_raw, manifest, state):
    eps_a = manifest.eps_a
    eps_w = manifest.eps_w
    w     = manifest.weight
```

```
# 1. Clamp
a_c = clamp(a_raw, -1 + eps_a, +1 - eps_a)

# 2. Rapidity transform
u = atanh(a_c)

# 3. Weighted accumulation
U_new = state.U_prev + w * u
W_new = state.W_prev + w

r = U_new / max(W_new, eps_w)

# 4. Bounded recovery
a_out = tanh(r)

return a_out, {U_new, W_new}
```

This pseudocode is manifest-driven and does not depend on language, device, or browser engine.

A.4 Numerical Stability Guarantees

The kernel is stable because:

- clamp prevents undefined atanh(±1)
- epsilons prevent division by zero
- tanh guarantees final values remain bounded
- the weighted structure prevents explosive growth in long sessions
- cross-language floating-point variations cannot break consistency (due to bounded outputs)

This ensures structural posture is safe even with:

- rapid tab switching
- deep redirect chains
- heavy browsing
- long multi-hour research flows

A.5 Kernel Determinism Requirements

An implementation is correct only if:

1. Steps 1–4 occur in the exact sequence above
2. No additional adjustments are added
3. No semantic, behavioral, or predictive signals modify `a_raw`
4. epsilons and weights match the manifest
5. tanh and atanh follow standard mathematical definitions

Any deviation produces a non-compliant kernel.

A.6 Alignment Kernel as a Universal Structural Contract

The kernel provides:

- **structural truth** (same inputs → same posture)
- **cross-device consistency** (phones, laptops, servers)
- **cross-language reproducibility**
- **transparent replayability**

This is the mathematical foundation that makes SSM-Browse a **global open structural standard**.

Appendix B — Reference Envelope Formats (Minimal & Extended)

This appendix defines the **canonical SSM-Browse envelope formats**.
They specify how structural metadata must be represented, stored, and exchanged.

Envelopes never contain:

- URLs
- content
- identity
- intent
- browsing history
- semantics of any kind

They carry only **structural mathematics**.

Two reference forms are provided:

1. **Minimal Envelope** — mandatory baseline
2. **Extended Envelope** — optional fields for advanced integrations

Any system that implements at least the Minimal format is considered compliant.

B.1 Envelope Philosophy

The envelope is:

- structural
- deterministic
- portable
- inspectable
- replayable

It attaches alongside any browsing event without altering the Value layer.

Example:

```
{
  value:   <raw browsing action>,
  envelope: { ...structural fields... }
}
```

Browsing remains exactly the same.
Only structural posture becomes visible.

B.2 Minimal Envelope (Mandatory Standard)

The minimal envelope is intentionally compact:

```
{
  "a": <float>,            // alignment posture output
  "band": "<label>",       // structural band declared by manifest
  "manifest_id": "<id>"    // pointer to declared rules
}
```

Field Requirements

Field	Meaning	Required
`a`	final posture from the alignment kernel, bounded in (-1,+1)	Yes
`band`	label from manifest thresholds (e.g., NAVIGATE / NEUTRAL / RESTRICT / WARN)	Yes
`manifest_id`	key linking to the manifest used for structural rules	Yes

Minimal Envelope Guarantees

- deterministic posture
- manifest-driven thresholds
- transparent structural interpretation
- cross-device reproducibility
- zero semantic inference

This is the **baseline envelope** for all SSM-Browse integrations.

B.3 Extended Envelope (Optional Advanced Fields)

Systems may add fields to support:

- coherence tracking
- tab structure
- drift stability
- reset events
- stamp chains
- signatures

All additional fields are optional and transparent.

Reference Extended Envelope

```
{
  "a": <float>,
  "q": <float or null>,          // Quero coherence (optional)
  "band": "<label>",
  "zeta0": <bool>,               // true if a ZETA-0 reset event
  "manifest_id": "<id>",
  "stamp": "<sha256hex>",        // optional chain element
  "sig": "<signature?>",         // optional verification
  "meta": {                      // optional browser-side structural notes
    "tab_id": "<id?>",
    "redirect_depth": <int?>,
    "sequence": <int?>           // recommended for ordering UI
  }
}
```

Meaning of Optional Fields

Field	Purpose
`q`	Quero structural coherence value (bounded like `a`)
`zeta0`	indicates a neutralizing ZETA-0 event was applied
`stamp`	SHA-256-based tamper-visible structural ordering
`sig`	optional cryptographic signature — *never identity*
`meta`	implementation-specific structural hints (not content)

None of these are required.
All must remain structural and non-semantic.

B.4 Envelope Extension Rules

Any extension must satisfy:

1. **No semantic or behavioral inference**
   Fields must never encode meaning, keywords, user intent, predictions, etc.
2. **No identity**
   No field may identify or correlate users.
3. **No hidden logic**
   All fields must be declared in a manifest or documentation.
4. **Reproducibility**
   Any added field must produce identical results across devices and languages.
5. **Neutrality**
   Structural data may not influence ranking, visibility, content selection, or personalization.

Envelope extensions should improve clarity, not reinterpret behavior.

B.5 Envelope Serialization Models

Envelopes may be serialized as:

- JSON (preferred)
- CBOR / MessagePack (if needed for compression)
- line-delimited JSON logs
- browser-internal metadata blocks
- sidecar files

Serialization must **not** alter:

- numeric precision
- ordering semantics
- clamping thresholds
- posture values

Replay must regenerate identical posture.

B.6 Envelope Versioning

The envelope **does not carry a version number**.

Versioning is handled by:

- `manifest_id`
- manifest schema evolution
- optional appendix identifiers

This keeps envelopes lightweight and future-proof.

B.7 Security & Integrity Notes

Envelopes must:

- never include sensitive data
- never leak browsing history
- never embed full URLs
- remain fully inspectable
- support offline replay without network calls

The envelope is purely mathematical.

Appendix C — Manifest Schema (Standard & Extension Guidelines)

A manifest is the **rulebook** that defines how a system must interpret SSM-Browse envelopes.

It does *not* contain:

- URLs
- content logic
- behavioral models
- personalization rules
- ML signals
- user identifiers

It contains only **structural mathematics and thresholds**.

A manifest must be:

- transparent
- public
- deterministic
- minimal
- versioned only when changed

- easy to audit and reproduce

Any system implementing SSM-Browse depends on the manifest for structural truth.

C.1 Purpose of the Manifest

The manifest defines:

- how posture inputs are clamped
- how rapidity values combine
- how Quero behaves (if used)
- how bands are assigned
- how ZETA-0 triggers
- whether stamps are used
- replay semantics
- drift rules for tabs and redirects
- optional structural extensions

The manifest **never** includes:

- personalization
- hidden weights
- dynamic inference
- behavioral scoring
- semantic interpretation
- device-specific rules

It is the single source of structural determinism.

C.2 Standard Manifest Schema (JSON)

A reference manifest looks like this:

```
{
  "id": "B1",
  "name": "Default Structural Manifest",
  "kernel": {
    "eps_a": 1e-6,
    "eps_w": 1e-6,
    "weights": {
      "default": 1.0
    }
  },
  "bands": [
    { "label": "NAVIGATE", "min": -0.25, "max": 1.00 },
    { "label": "NEUTRAL",  "min": -0.05, "max": 0.05 },
    { "label": "RESTRICT", "min": -1.00, "max": -0.25 },
```

```
    { "label": "WARN",     "min": -1.00, "max": -0.75 }
  ],
  "quero": {
    "enabled": true,
    "eps_q": 1e-6,
    "combine_mode": "weighted",
    "weight": 1.0
  },
  "zeta0": {
    "enabled": true,
    "threshold": 0.85
  },
  "stamps": {
    "enabled": true,
    "hash": "sha256"
  },
  "replay": {
    "ordering": "stamp",
    "strict_mode": true
  }
}
```

This schema contains *only* structural rules.

C.3 Field-by-Field Explanation

1. id

A stable, unique identifier.
Never re-use IDs; add new ones for rule changes.

2. kernel

Defines all posture mathematics.

- `eps_a` — prevents edge overflow during clamp
- `eps_w` — prevents division by zero
- `weights` — how to combine rapidity values

3. bands

A list of non-overlapping posture ranges with human-readable labels.

Rules:

- must be contiguous
- must cover the full (-1, +1) space
- must not overlap
- must be documented

4. quero (optional)

Defines structural coherence behavior.

Fields include:

- whether Quero is enabled
- clamping epsilon
- combination mode (e.g., streaming, weighted)
- Quero weight for multi-event workflows

5. zeta0 (optional but recommended)

Defines:

- when ZETA-0 is inserted
- how thresholds are interpreted (absolute or smoothed)

6. stamps

Defines whether session stamping is required.

- `enabled`: true/false
- `hash`: currently `sha256` (other hashes must be aligned across systems)

7. replay

Defines:

- whether ordering uses stamps or sequence numbers
- strict mode: systems MUST error if envelopes violate rules

C.4 Minimal Manifest Schema

For lightweight or embedded systems:

```
{
  "id": "B1",
  "kernel": {
    "eps_a": 1e-6,
    "eps_w": 1e-6
  },
  "bands": [
    { "label": "NAVIGATE", "min": -0.25, "max": 1.00 },
    { "label": "NEUTRAL",  "min": -0.05, "max": 0.05 },
    { "label": "RESTRICT", "min": -1.00, "max": -0.25 }
  ]
}
```

This is the smallest valid manifest.

C.5 Manifest Design Principles

All manifests must satisfy these rules:

1. Determinism

Two systems using the same manifest must produce identical posture.

2. Transparency

Nothing may be hidden, inferred, or dynamic.

3. Structural Neutrality

No semantics, no ranking, no personalization.

4. Reproducibility

Replay on any machine must yield identical output.

5. Auditability

Manifests must be readable by humans.

6. Simplicity

Manifests should avoid unnecessary fields.

C.6 Manifest Versioning Rules

Versioning occurs only when:

- thresholds change
- Quero logic changes
- clamp parameters change
- ZETA-0 behavior changes
- stamps are added/removed

Versioning must be:

- monotonic (B1 → B2 → B3)
- documented

- backward-compatible unless explicitly broken

The manifest ID is *not* semantic — it is structural.

C.7 Forbidden Manifest Behaviors

A manifest must **never**:

- introduce hidden weights
- embed ML models
- include semantic rules
- encode content types
- include device-specific logic
- adjust posture based on behavior
- incorporate personal identifiers
- create vendor-specific ranking logic

Manifests must remain pure.

C.8 Publishing Manifests

Manifests must be:

- published publicly
- downloadable
- verifiable
- archived
- immutable once referenced

This ensures global consistency.

Appendix D — Numerical Stability Notes (Clamp Zones, Overflow Safety, Edge Behavior)

SSM-Browse uses bounded posture values and rapidity transforms.
This introduces specific numerical stability considerations that every implementation must respect to ensure:

- deterministic results

- cross-device consistency
- platform neutrality
- long-term reproducibility
- safety under extreme browsing patterns

This appendix provides precise guidance for clamp zones, overflow prevention, and stable rapidity behavior.

D.1 Why Numerical Stability Matters

The alignment kernel depends on two mathematical operations that can become unstable if not controlled:

1. **clamp** — prevents posture escaping (-1,+1)
2. **atanh** — has asymptotes near -1 and +1

When browsing sessions involve:

- deep navigation chains
- high-frequency events
- parallel tab workflows
- repeated posture accumulation
- long redirect sequences

unstable posture can drift into edge zones unless the implementation respects strict bounds.

SSM-Browse solves this with controlled numerical safety rules.

D.2 Core Safety Guarantees

Every SSM-Browse implementation must guarantee:

1. `a_raw` is clamped before transformation
2. `a_c` always satisfies:
3. `-1 + eps_a <= a_c <= +1 - eps_a`
4. `atanh(a_c)` cannot overflow
5. weighted rapidity combination uses safe denominators
6. posture recovery using `tanh` always returns values inside (-1,+1)
7. Quero (if used) follows the same safety guarantees

These rules eliminate:

- floating-point drift
- platform-dependent overflow
- unstable edge values

- inconsistent posture recovery

D.3 Clamp Zones and eps_a

The clamp step is defined as:

```
a_c = clamp(a_raw, -1+eps_a, +1-eps_a)
```

Where:

- `eps_a` prevents approaching atanh singularities
- typical value: `1e-6`
- minimum recommended: `1e-12`
- smaller values must be used only if platform precision is guaranteed

Why clamp exists:
`atanh()` diverges at exactly -1 and +1.
Clamping ensures finite rapidity values in all implementations.

Implementation Note:
Clamping must occur *before* any streaming combine, not after.

D.4 Rapidity Overflow Avoidance

The transform step:

```
u = atanh(a_c)
```

Rapidity (`u`) remains finite because `a_c` is always strictly inside (-1,+1).
This guarantees:

- no infinite rapidity
- no undefined outputs
- no platform-to-platform divergence

Even in extreme browsing sequences (e.g., thousands of events), the combined rapidity will remain mathematically stable.

D.5 Safe Weight Division

The kernel uses:

```
a_out = tanh(U / max(W, eps_w))
```

Where:

- `U` is cumulative rapidity
- `W` is the declared weight
- `eps_w` prevents division by zero

Stability guarantees:

- posture recovery always stays within (-1,+1)
- large rapidity values get smoothly compressed
- zero or extremely small weights cannot destabilize output

Recommended `eps_w` values:

- typical: `1e-6`
- minimum: `1e-12`

D.6 Floating-Point Cross-Platform Consistency

Platforms differ by:

- float vs double precision
- math library implementations
- rounding rules
- compiler optimizations

SSM-Browse avoids these inconsistencies because:

- inputs are clamped
- outputs are bounded
- rapidity is stable
- posture is recovered via `tanh`

Any differences remain far below symbolic thresholds and cannot affect:

- band assignment
- Quero stability
- ZETA-0 triggers
- structural replay

D.7 Extreme Scenario Behavior

Scenario A — Very Long Browsing Sessions

Thousands of events → cumulative rapidity grows → `tanh` compresses output → posture remains stable.

Scenario B — Rapid Tab Switching

Posture remains bounded.
Quero (if used) stabilizes multi-threaded divergence.

Scenario C — Parallel Exploration

Large differences in rapidity still compress into meaningful, bounded posture.

Scenario D — Repeated Redirect Loops

Structural drift cannot escape the boundaries.
ZETA-0 can reset posture by manifest declaration.

D.8 Zero-Drift Guarantees

Because posture is always recovered using:

```
a_out = tanh(...)
```

We guarantee:

- no unbounded growth
- no monotonic drift
- no cumulative instability
- no cross-platform divergence

This ensures that:

- envelopes remain valid
- manifests remain reliable
- replay remains deterministic

D.9 Why Numerical Stability Enables Global Interoperability

Numerical stability ensures that every device in the world:

- computes the same posture
- assigns the same band
- detects the same ZETA-0 thresholds
- observes the same Quero coherence
- reconstructs identical sessions

This creates:

- universal reproducibility
- platform neutrality
- deterministic trust
- zero hidden drift

Mathematics becomes the global equalizer.

Appendix E — Example Manifests (Reference Templates for Implementers)

These manifests are illustrative. They define structural rules only — never content interpretation.

Each manifest below demonstrates a different style of SSM-Browse deployment:

- **Base Manifest** — minimal, safe defaults
- **Quero-Enabled Manifest** — adds coherence tracking
- **ZETA-0–Weighted Manifest** — suitable for long research sessions
- **Minimal Manifest** — tiny footprint for lightweight systems
- **Compliance-Strict Manifest** — for regulated or archival environments

All examples are deterministic, transparent, and content-independent.

E.1 Base Manifest (Recommended Starting Point)

A simple envelope configuration suitable for most systems.

```
manifest_id: B1
description: "Baseline structural manifest for SSM-Browse"
```

```
alignment:
  eps_a: 1e-6
  eps_w: 1e-6
  weight_default: 1.0

bands:
  NAVIGATE:   [-0.25, +0.25]
  NEUTRAL:    [-0.50, +0.50]
  RESTRICT:   [-0.75, +0.75]
  VERIFY:     [-0.90, +0.90]
  WARN:       [-1.00, +1.00]

quero:
  enabled: false

zeta0:
  enabled: false

stamps:
  enabled: true
  timestamp: utc
```

Use cases:
General browsing, research tools, institutional dashboards.

E.2 Quero-Enabled Manifest (Multi-Tab Structural Coherence)

Adds coherence tracking without affecting values or user behavior.

```
manifest_id: Q1
description: "Quero-enabled coherence manifest"

alignment:
  eps_a: 1e-6
  eps_w: 1e-6
  weight_default: 1.0

bands:
  NAVIGATE:   [-0.20, +0.20]
  NEUTRAL:    [-0.45, +0.45]
  VERIFY:     [-0.80, +0.80]
  WARN:       [-1.00, +1.00]

quero:
  enabled: true
  eps_q: 1e-6
  weight_q: 1.0

zeta0:
  enabled: false

stamps:
  enabled: true
```

Use cases:
Browsing assistants, multi-tab research, investigation workflows.

E.3 ZETA-0–Weighted Manifest (High Stability for Deep Sessions)

Designed for long research, recursive navigation, or multi-hour browsing sessions.

```
manifest_id: Z1
description: "ZETA-0 heavy structural stabilizer manifest"

alignment:
  eps_a: 1e-6
  eps_w: 1e-6
  weight_default: 1.0

bands:
  NAVIGATE:   [-0.15, +0.15]
  NEUTRAL:    [-0.40, +0.40]
  VERIFY:     [-0.70, +0.70]
  WARN:       [-1.00, +1.00]

quero:
  enabled: true
  eps_q: 1e-6
  weight_q: 1.0

zeta0:
  enabled: true
  trigger_threshold: 0.65
  reset_posture: 0.0
  cooldown_events: 30

stamps:
  enabled: true
```

Use cases:
Regulated environments, academic research sessions, deep exploration.

E.4 Minimal Manifest (Tiny-Footprint, Lightweight Systems)

Designed for lightweight browsers, embedded systems, or low-resource environments.

```
manifest_id: M1
description: "Minimal envelope manifest"

alignment:
  eps_a: 1e-6
  eps_w: 1e-6
  weight_default: 1.0
```

```
bands:
  NAVIGATE: [-1.0, +1.0]

quero:
  enabled: false

zeta0:
  enabled: false

stamps:
  enabled: false
```

Use cases:
Small browsers, offline viewers, kiosk systems, micro-browsers.

E.5 Compliance-Strict Manifest (Audit, Governance, Archival)

Designed for environments requiring maximum structural reproducibility, audit trails, and deterministic replay.

```
manifest_id: C1
description: "Compliance strict manifest for audited browsing flows"

alignment:
  eps_a: 1e-12
  eps_w: 1e-12
  weight_default: 1.0

bands:
  NAVIGATE:   [-0.10, +0.10]
  NEUTRAL:    [-0.25, +0.25]
  RESTRICT:   [-0.50, +0.50]
  VERIFY:     [-0.75, +0.75]
  WARN:       [-1.00, +1.00]

quero:
  enabled: true
  eps_q: 1e-12
  weight_q: 1.0

zeta0:
  enabled: true
  trigger_threshold: 0.50
  reset_posture: 0.0
  cooldown_events: 10

stamps:
  enabled: true
  timestamp: utc
  hash_chain: required
```

Use cases:
Compliance centers, forensic browsing reconstruction, regulatory audits, archival preservation.

Appendix F — Developer Integration Patterns (Overlay, Native, Hybrid)

These patterns demonstrate how SSM-Browse integrates into real systems with minimal friction and maximum structural consistency.

SSM-Browse deliberately supports multiple integration styles because browsing ecosystems vary widely:

- some systems need zero-impact overlays
- some want full structural engines
- some require mixed modes for different workflows

This appendix provides clean implementation pathways for all three patterns.

F.1 Overlay Integration Pattern (Fastest Adoption)

Use when: you want SSM-Browse immediately, with no browser redesign.

Overlay Mode attaches the envelope *beside* standard browsing events:

- no engine modification
- no rendering change
- no script change
- no ranking or behavior impact

The browsing action flows exactly as before; the envelope travels alongside as structural metadata.

Overlay Architecture Diagram (Conceptual)

```
[event] → [attach envelope] → [store] → [replay as needed]
```

Minimal Integration Steps

1. **Capture Event**
2. `value = { type, url, query, tab_id, ts }`
3. **Generate Envelope**
4. `envelope = generate_envelope(value, manifest)`

5. **Attach Envelope**
6. `event.envelope = envelope`
7. **Store (Optional)**
    - local log
    - cloud store
    - institutional archive
8. **Replay (Optional)**
9. `replay_session(records)`

Benefits

- instant deployment
- minimal code changes
- replay-ready logs
- zero performance overhead
- safe for pilots and institutional dashboards
- works in any language or platform

Overlay Mode is the default recommended integration path for early implementers.

F.2 Native Integration Pattern (Full Structural Browsing Engine)

Use when: your platform requires structural coherence, multi-tab stability, or long-session determinism.

Native Mode turns the browser into a **symbolic structural engine**, enabling:

- posture memory
- Quero multi-tab coherence
- deterministic redirect handling
- ZETA-0 drift correction
- tab lineage stability
- structural session graphs

Native Architecture Diagram

```
[browser engine]
      ↓
[alignment kernel]
      ↓
[tab structural state]
      ↓
[quero coherence + zeta0]
      ↓
[structural session model]
```

Key Components

1. Structural Tab State

Each tab maintains:

```
tab_state = {
  alignment: a_out,
  coherence: q_out?,
  zeta0_pending: bool,
  redirect_chain: [...],
  posture_history: [...]
}
```

2. Quero Lane

Tracks cross-tab divergence:

```
q_out = tanh( (Σ rapidity_q) / max(weight_q, eps_w) )
```

3. ZETA-0 Enforcement

Ensures drift never exceeds manifest-defined limits:

```
if |a_out| > trigger_threshold:
    insert ZETA0
    a_out = 0.0
```

4. Deterministic Redirect Logic

Redirect chains follow structural rules, not engine-specific heuristics.

5. Native Replay

Playback reconstructs:

- tab states
- divergence patterns
- redirect structure
- session alignment
- ZETA-0 points

Benefits

- perfect reproducibility
- institutional-grade integrity
- stable multi-tab research

- deterministic workflows
- tab-level structural awareness
- safer long sessions

Native Mode is ideal for future browsers, enterprise systems, and structural research tools.

F.3 Hybrid Integration Pattern (Overlay + Native Coexistence)

Use when: some workflows require structural depth, others need minimalism.

Hybrid Mode allows:

- general browsing → Overlay Mode
- critical workflows → Native Mode
- mixed logs, unified manifests

Hybrid Architecture

```
[event] → choose mode:
    - overlay → attach envelope only
    - native  → compute full structural state
          ↓
[common storage & replay layer]
```

Both paths produce envelopes.
Both are replayable.
Both use the same manifest.
Differences appear only in depth of structural modeling.

Use Cases

- enterprises with multiple browsing profiles
- compliance systems with “structural tabs”
- browsers offering “research mode”
- tab-specific symbolic tracing

Benefits

- optional structural depth
- minimal friction
- unified data format
- flexible deployment strategy
- smooth migration toward Native Mode

F.4 Real-World Integration Examples (Conceptual Only)

No content, user behavior, or systems are ever interpreted — these are purely structural patterns.

Example 1 — Browser Extension

Overlay Mode envelope attached to navigation events using a lightweight script.

Example 2 — Institutional Proxy

SSM-Browse envelopes added at the proxy layer for research or compliance replay.

Example 3 — Native Browser Engine

The browser integrates Quero lanes and ZETA-0 using the declared manifest.

Example 4 — Structural Research Tool

A separate tool consumes stored envelopes to reconstruct browsing alignment graphs.

Appendix G — Reference Alignment Kernel (Platform-Neutral Pseudocode)

The Alignment Kernel is the mathematical heart of SSM-Browse.
It defines how posture and coherence evolve across browsing events using a bounded, fully deterministic transformation.
It is strictly non-semantic and structurally neutral.

It ensures:

• determinism
• neutrality
• safety
• reproducibility
• identical results across all systems
• zero semantic processing
• no hidden heuristics or personalization

This appendix provides the canonical pseudocode applicable to every platform and device.

G.1 Kernel Goals

Every implementation of SSM-Browse must preserve:

- `a_lane ∈ (-1,+1)`
- `q_lane ∈ (-1,+1)`
- deterministic clamp rules
- rapidity-space transformations
- manifest-declared weighted combination
- ZETA-0 reset behavior
- overflow-safe execution
- platform-independent reproducibility

No system may optimize away, reorder, or reinterpret these mathematical steps.
They must remain explicit and observable.

G.2 Core Formula (Canonical Form)

Given a declared raw structural posture:

`a_raw`

The kernel computes the bounded output `a_out` using **four invariant steps**:

1. **Clamp**
2. `a_c = clamp(a_raw, -1 + eps_a, +1 - eps_a)`
3. **Rapidity Transform**
4. `u = atanh(a_c)`
5. **Weighted Combine**
6. `u_out = U_total / max(W_total, eps_w)`
7. **Recover Bounded Signal**
8. `a_out = tanh(u_out)`

These steps must always be explicit.
No shortcuts.
No compressed approximations.
No machine-learned reinterpretations.

G.3 Platform-Neutral Pseudocode

```
function compute_posture(a_raw, manifest):

    eps_a = manifest.eps_a
    eps_w = manifest.eps_w

    # Step 1 — Clamp raw posture
    a_c = clamp(a_raw, -1 + eps_a, +1 - eps_a)
```

```
    # Step 2 — Map to rapidity space
    u = atanh(a_c)

    # Step 3 — Combine using declared weights
    U_total = manifest.U_prev + u
    W_total = manifest.W_prev + manifest.weight

    u_out = U_total / max(W_total, eps_w)

    # Step 4 — Recover bounded result
    a_out = tanh(u_out)

    return a_out
```

Notes
• `U_prev` and `W_prev` are manifest-declared cumulative values.
• Full-precision `atanh` and `tanh` must be used.
• Fixed-precision variants may only be used if the manifest explicitly declares them.
• All steps must be observable and testable.

G.4 Clamp Function (Exact Behavior)

```
function clamp(x, low, high):
    if x < low:  return low
    if x > high: return high
    return x
```

Clamp rules must never be modified.
Clamp is the first safety boundary.

G.5 Numeric Safety Requirements

All implementations must guarantee:

• no overflow in `atanh`
• no division by zero (`eps_w`)
• no boundary collapse (`eps_a`)
• `a_out ∈ (-1,+1)`
• identical results across backends

Suggested numeric parameters:

```
eps_a = 1e-9
eps_w = 1e-12
```

These may vary only through explicit manifest declaration.

G.6 Quero Kernel (Optional Coherence Lane)

The Quero lane (`q_lane`) uses the same canonical structure as posture:

```
q_c   = clamp(q_raw, -1 + eps_a, +1 - eps_a)
v     = atanh(q_c)
v_out = tanh( (V_total / max(Wq_total, eps_w)) )
```

Quero must remain:

• structural
• deterministic
• bounded
• non-semantic
• independent of meaning or content

It captures structural smoothness, divergence, or shocks across browsing interactions.

G.7 ZETA-0 Reset Logic (Drift-Aware Stabilization)

ZETA-0 is a deterministic zero-event invoked when cumulative drift exceeds manifest thresholds.

Manifest-declared drift condition:

```
if drift >= DRIFT_LIMIT:
    a_lane = 0
    q_lane = 0
    reset cumulative U_total and V_total if manifest declares it
```

Important properties:

• resets never interpret content
• resets never modify the value layer
• resets are declared, neutral, and transparent
• all environments reproduce the same reset at the same step
• ZETA-0 restores stability without changing the action itself

This ensures long sessions, multi-tab sequences, and research workflows remain structurally predictable.

G.8 Multi-Event Streaming Kernel

When multiple events contribute to posture evolution:

```
U_total = 0
```

```
W_total = 0

for event in events:
    u_i = atanh(clamp(a_raw_i, -1 + eps_a, +1 - eps_a))
    U_total += u_i
    W_total += event.weight_i

a_out = tanh(U_total / max(W_total, eps_w))
```

This guarantees:

• deterministic alignment across long research sessions
• consistent results even with hundreds of interactions
• identical replay on any system
• resilience to complex tab + redirect flows

G.9 Multi-Tab Stability Kernel (Optional but Recommended)

For platforms supporting multi-tab structural merging:

```
function merge_tabs(tab_A, tab_B, manifest):

    uA = atanh(clamp(tab_A.a_lane))
    uB = atanh(clamp(tab_B.a_lane))

    U_merged = uA + uB
    W_merged = tab_A.weight + tab_B.weight

    a_out = tanh(U_merged / max(W_merged, eps_w))

    return a_out
```

This ensures tabs evolve with coherent structural alignment during:

• multi-tab research
• branching exploration
• cascading redirects
• partial or full tab merges

G.10 Structural Guarantees

The kernel guarantees:

• same inputs → same posture
• monotonic deterministic behavior
• no silent reweighting
• no hidden adjustments
• no personalization

• no semantic interference
• tamper-visible lineage
• perfect replay across devices, environments, and time

This is what makes SSM-Browse:

• reproducible
• trustworthy
• structurally safe
• platform-independent
• mathematically auditable

Appendix H — Safety, Ethics, and Non-Semantic Boundaries

This appendix consolidates all safety constraints, ethical guardrails, and prohibited uses that preserve SSM-Browse's neutrality, fairness, and structural integrity.

SSM-Browse is a structural standard.
It is **not** a behavioral system, predictive model, recommendation engine, or ranking layer.

This appendix defines the strict boundaries that keep the system:

- safe
- non-semantic
- deterministic
- globally interoperable
- mathematically neutral

These boundaries are mandatory for all implementations.

H.1 Core Ethical Principles

SSM-Browse is built on three non-negotiable principles:

1. Structure ≠ Meaning

- Posture (a) and Quero (q) describe *structure only*
- No part of the envelope may interpret:
  - keywords
  - queries
  - topics
  - sentiment

- user intent
- preferences
- behavior

2. Neutrality at All Layers

- The envelope must never alter ranking, visibility, or order
- The Value layer (the real browsing action) remains **100% untouched**
- No personalization is allowed
- No inference or hidden weighting may be added

3. Universal Transparency

- All structural rules must be declared
- Manifests must remain public and immutable within their scope
- No hidden parameters
- No unverifiable computation paths

These principles ensure SSM-Browse stays mathematically fair across systems and contexts.

H.2 Structural Safety Requirements

Every implementation must uphold:

- deterministic posture computation
- bounded signals using `(-1,+1)`
- stable Quero coherence under all browsing flows
- transparent ZETA-0 resets when drift exceeds declared thresholds
- full replayability based solely on structural metadata

SSM-Browse must function identically across:

- devices
- browsers
- architectures
- operating systems

If structural behavior diverges, the manifest is not being followed.

H.3 Prohibited Uses (Strict & Enforced)

To prevent misuse, SSM-Browse **must not** be used for:

Semantic or Behavioral Interpretation

- sentiment detection
- intent inference
- topic classification
- preference modeling
- behavioral profiling
- emotional analysis

Ranking or Influence

- hidden ranking
- silent boosts or demotions
- personalization of results
- manipulating visibility
- altering search outcomes
- filtering, adjusting, or rewriting browsing behavior

Identity or Surveillance

- user identification
- demographic inference
- constructing behavioral dossiers
- cross-site tracking
- combining a/q with external identifiers

Machine-Learned Interference

- inserting ML-generated signals into a or q
- optimizing posture through training
- replacing the kernel with a learned surrogate

Structural Distortion

- hidden thresholds
- undeclared ZETA-0 triggers
- proprietary envelope extensions that break reproducibility
- privately capturing, enclosing, or withholding parts of the standard

These prohibitions protect neutrality and prevent misuse of structural metadata.

H.4 Allowed Uses (Safe & Encouraged)

SSM-Browse may be used for:

- structural browsing metadata
- deterministic replay

- session integrity verification
- drift analysis
- compliance auditing
- multi-tab stability tracking
- research reproducibility
- debugging toolchains
- visualization overlays
- educational material on deterministic browsing

These use-cases preserve the standard's non-semantic nature.

H.5 Value Layer Preservation

The Value (the actual browsing action) must always remain:

- untampered
- unaltered
- unfiltered
- unread
- unscored
- uninterpreted

This is the foundation of SSM-Browse.

No field in the envelope may influence what the user sees.
The envelope describes *structure*, never the browsing experience itself.

H.6 Manifest Responsibility

All structural behavior must be declared in the manifest:

- clamps
- epsilons
- weight parameters
- Quero behavior
- reset logic
- band thresholds
- stamp chain rules

Nothing may be added outside the manifest.

Manifests must be:

- public
- readable

- deterministic
- traceable
- versioned only when changed

This ensures long-term trust and interoperability.

H.7 Structural Fairness

SSM-Browse guarantees global fairness by ensuring:

- identical structural results everywhere
- no device-dependent behavior
- no browser-specific drift
- no personalization
- no back-channel influence
- no embedded preferences

Fairness emerges not from policy — but from **mathematics**.

H.8 Trust Philosophy

Trust in SSM-Browse is built on:

- transparent mathematics
- deterministic alignment
- bounded signals
- zero semantic processing
- strict safety rules
- reproducible manifests
- verifiable stamp chains
- no hidden behavior

Trust is earned through **structure**, not opacity.

SSM-Browse raises the global baseline for digital integrity by giving users and institutions a deterministic, mathematically truthful model of browsing.

OMP