LUXCRYPTA Technologies LLC

Prompt Accelerator

Technical White Paper

Founder and CEO: Thomas Burton Coleman

14781 Memorial Drive, Suite 253
Houston, Texas 77079
Email: tcoleman@luxcrypta.ai
Website: https://luxcrypta.ai

Version 1.0

May 18, 2026

Abstract. LUXCRYPTA Prompt Accelerator is a local-first continuity runtime for AI workflows. Its purpose is not merely to rewrite prompts, but to reduce workflow entropy across long-running interactions with large language models. The system stabilizes objectives, constraints, decisions, unresolved questions, and model-targeted handoff state; compresses redundant prompt material; separates stable from provisional state; preserves open questions where appropriate; and packages continuity for transfer across model ecosystems including ChatGPT, Claude, Gemini, and Grok. This paper defines the product philosophy, formal architecture, mathematical abstractions, runtime state model, capsule schema, governance model, transformation pipeline, provider routing strategy, browser-extension realization, and forward path toward desktop continuity infrastructure.

Contents

1. Executive Summary

Prompt Accelerator is a continuity system for AI work. Its core proposition is simple: most AI workflows degrade over time due to context drift, repeated setup, prompt sprawl, forgotten constraints, and weak transfer between sessions or models. Prompt Accelerator addresses those failures by acting as a lightweight operating layer over AI interactions.

The central thesis is:

> **LUXCRYPTA owns continuity, not cognition.**

The external model still performs cognition and generation. Prompt Accelerator manages continuity, compression, workflow state, transferability, and governance over evolving session state.

In practical terms, the system:

- reduces redundant prompt material,
- preserves high-confidence workflow state,
- isolates new or provisional instructions,
- preserves open questions and unresolved risks,
- generates continuity capsules for later reuse,
- adapts handoff structures across model ecosystems,
- and presents the resulting state in human-readable form.

The system evolved from a browser extension into a broader continuity architecture. The browser extension remains a key surface, but the underlying logic is properly understood as continuity infrastructure.

2. Problem Definition

2.1. Failure Modes in Practical LLM Usage

Real-world LLM workflows tend to fail in recurring ways:

1. **Context Drift:** the active objective degrades or is replaced by nearby but not identical goals.
2. **Constraint Loss:** output requirements, prohibitions, style rules, or boundary conditions fall out of the active prompt frame.
3. **Decision Amnesia:** decisions made earlier in the workflow are forgotten, re-litigated, or contradicted later.

4. **Prompt Sprawl:** repeated reminders, caveats, and structural restatements cause the prompt to grow without improving clarity.
5. **Rebuild Overhead:** users repeatedly reconstruct the same context by hand across sessions.
6. **Cross-Model Fragmentation:** moving from one provider to another destroys workflow state continuity.
7. **Memory Pollution:** low-confidence speculative or temporary instructions contaminate the stable operating state.
8. **Open-Question Collapse:** unresolved items disappear rather than being preserved as intentionally open.

2.2. Design Response

Prompt Accelerator is explicitly designed to address these failures with:

- a continuity state model,
- a stable/provisional/open partition,
- always-on redundancy reduction and objective prioritization,
- a capsule persistence mechanism,
- governance logic over evolving state,
- and provider-aware continuity handoff generation.

3. Product Definition

3.1. What Prompt Accelerator Is

Prompt Accelerator is:

- a continuity runtime,
- a workflow persistence engine,
- a prompt/session stabilizer,
- a capsule generator,
- a provider-aware handoff system,
- a local-first AI workflow layer.

3.2. What Prompt Accelerator Is Not

Prompt Accelerator is *not*:

- a standalone chatbot,
- a model,
- hidden control over third-party model internals,
- a generic note-taking app,
- a template marketplace,
- or a passive prompt library.

4. Core Product Principles

4.1. Principle 1: Continuity over Prompt Ornamentation

Prompt rewriting is useful but insufficient. The system optimizes for continuity preservation, not superficial prompt polish.

4.2. Principle 2: Stable State Must Be Explicit

The system must know which parts of the workflow are:

- stable and accepted,
- provisional or newly introduced,
- unresolved and intentionally open.

4.3. Principle 3: Compression Must Preserve Meaning

Compression must reduce redundancy and cognitive noise while preserving essential state.

4.4. Principle 4: Focus Must Be Native

Objective prioritization is not a user-selected mode. It is a runtime obligation.

4.5. Principle 5: Human-Readable Before Raw

Review surfaces must present continuity state in human-readable form first, with raw/internal diagnostics accessible but secondary.

4.6. Principle 6: Local-First by Default

Core continuity functions must work without backend dependence in the browser-extension and desktop-MVP contexts.

5. High-Level Architecture

5.1. Conceptual Stack

The conceptual architecture is:

User Interface Layer → Continuity Runtime → Governance Layer → Capsule Persistence → Compression / Pric

5.2. Browser-Extension Realization

For the browser extension, the realized stack becomes:

Web Surface → Content/Action Layer → Continuity Pipeline → Review Layer → Apply / Handoff

5.3. Desktop Realization

For the desktop runtime, the realized stack becomes:

Workspace UI → Continuity Engine → Governance → Capsule / Workflow Store → Provider Routing

6. System Modules

6.1. Continuity Engine

Responsibilities:

- track active objective,
- track stable constraints,
- track accepted decisions,
- track unresolved questions and risks,

- update continuity state over time,
- produce human-readable state summaries.

6.2. Governance Layer

Responsibilities:

- prevent stable-state corruption,
- isolate novelty,
- preserve useful openness,
- avoid memory pollution,
- score session health,
- recommend promotions or suppressions.

6.3. Capsule System

Responsibilities:

- generate carry-forward capsules,
- version and persist capsules,
- support model-agnostic internal representation,
- support provider-targeted external packaging.

6.4. Compression / Prioritization Layer

Responsibilities:

- reduce redundancy,
- normalize duplicated requirement phrasing,
- prioritize toward the primary objective,
- preserve fidelity and output contracts.

6.5. Provider Routing Layer

Responsibilities:

- adapt continuity handoff for ChatGPT,
- adapt continuity handoff for Claude,
- adapt continuity handoff for Gemini,
- adapt continuity handoff for Grok,
- remain provider-agnostic internally.

7. Formal State Model

7.1. Workflow State

Define a workflow state at time t as:

$$W_t = (O_t, C_t, D_t, U_t, P_t, M_t, H_t)$$

where:

- O_t: active objective,
- C_t: stable constraints,
- D_t: accepted decisions,
- U_t: unresolved items,
- P_t: provisional items,
- M_t: preferred mode / model routing metadata,
- H_t: health and diagnostics state.

7.2. Partitioned Continuity State

We partition continuity into three disjoint regions:

$$S_t \cup P_t \cup U_t = X_t$$

where:

- S_t: stable accepted state,
- P_t: new or provisional state,
- U_t: open / unresolved state.

with:

$$S_t \cap P_t = \varnothing, \quad S_t \cap U_t = \varnothing, \quad P_t \cap U_t = \varnothing$$

This partition is central to governance and review generation.

7.3. Stable Core

The stable core is defined as:

$$\mathcal{K}_t = (O_t^\star, C_t^\star, D_t^\star, Q_t^\star)$$

where:

- $O_t^\star$: normalized active objective,
- $C_t^\star$: deduplicated stable constraints,
- $D_t^\star$: accepted decisions,
- $Q_t^\star$: persistent output contract / quality rules.

8. Mathematical Structures

8.1. Redundancy Reduction

Let the input text sequence be $T = (u_1, u_2, \ldots, u_n)$, where each u_i is a meaningful instruction unit. Define a redundancy relation $R(u_i, u_j)$ such that:

$$R(u_i, u_j) = \begin{cases} 1 & \text{if } u_i \text{ and } u_j \text{ are semantically redundant} \\ 0 & \text{otherwise} \end{cases}$$

The redundancy burden is:

$$\mathcal{R}(T) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} R(u_i, u_j)$$

The compression substep seeks a transformed sequence T' such that:

$$\mathcal{R}(T') \ll \mathcal{R}(T)$$

subject to preservation of stable constraints and accepted decisions.

8.2. Objective Prioritization

Let $G = \{g_1, \ldots, g_m\}$ denote candidate goals extracted from the prompt plus prior state.

Assign each goal a priority score:

$$\pi(g_i) = \alpha \cdot \text{relevance}(g_i, O_t) + \beta \cdot \text{stability}(g_i) + \gamma \cdot \text{recency}(g_i) - \delta \cdot \text{conflict}(g_i, \mathcal{K}_t)$$

The active objective selected for handoff is:

$$O_{t+1} = \arg\max_{g_i \in G} \pi(g_i)$$

8.3. Constraint Preservation Score

Let C_{orig} be the set of hard constraints extracted from incoming state and C_{out} the set preserved after transformation.

Then:

$$\text{CPS} = \frac{|C_{\text{orig}} \cap C_{\text{out}}|}{|C_{\text{orig}}| + \varepsilon}$$

with $\varepsilon > 0$ small to avoid division by zero.

The runtime should target:

$$\text{CPS} \approx 1$$

for non-permissive transformations.

8.4. Continuity Health

Define continuity health as a vector:

$$H_t = (\chi_t, \delta_t, \nu_t, \omega_t, \kappa_t)$$

where:

- χ_t: continuity score,
- δ_t: drift score,
- ν_t: novelty load,
- ω_t: openness score,
- κ_t: compression density.

A practical scalar health summary may be:

$$\mathcal{H}_t = a\chi_t - b\delta_t - c\nu_t + d\omega_t + e\kappa_t$$

with positive weights a, b, c, d, e.

8.5. Promotion Heuristic

For a provisional item $p \in P_t$, define promotion likelihood:

$$\Pi(p) = \eta_1 \cdot \text{repeat}(p) + \eta_2 \cdot \text{consistency}(p, \mathcal{K}_t) + \eta_3 \cdot \text{utility}(p) - \eta_4 \cdot \text{conflict}(p, \mathcal{K}_t)$$

Promote p into stable state only if:

$$\Pi(p) \geq \tau$$

for threshold τ chosen conservatively.

9. Capsule Model

9.1. Definition

A carry-forward capsule is a portable, versioned continuity artifact preserving enough workflow state to continue a project without rebuilding context from scratch.

9.2. Canonical Capsule Schema

Listing 1: Canonical carry-forward capsule schema

```
{
  "capsule_version": 1,
  "id": "capsule-uuid",
  "title": "Launch strategy continuity capsule",
```

```
  "active_objective": "Pressure-test launch positioning and produce next-step guidance.",
  "stable_constraints": [
    "Stay aligned to workflow continuity positioning",
    "Avoid drifting into generic prompt-tool language"
  ],
  "accepted_decisions": [
    "Prompt Accelerator is positioned as continuity infrastructure",
    "Advanced view is inspection-first, not mode-selection"
  ],
  "open_questions": [
    "How should Grok support be prioritized in extension messaging?"
  ],
  "unresolved_risks": [
    "Marketing may still undersell workflow continuity value"
  ],
  "preferred_mode": "continuity_runtime",
  "next_actions": [
    "Refine homepage positioning",
    "Validate Grok support surface behavior"
  ],
  "provider_target": "chatgpt",
  "notes": "Keep review window human-readable first.",
  "created_at": "2026-01-01T00:00:00Z",
  "updated_at": "2026-01-01T00:00:00Z"
}
```


9.3. Capsule Processing Rule

The runtime must never treat the raw capsule as the final review representation.

Instead:

$$\text{Capsule JSON} \rightarrow \text{Semantic Extraction} \rightarrow \text{Human Readable State} \rightarrow \text{Review/Handoff}$$

10. Always-On Continuity Runtime

10.1. Core Idea

Compression and focus are no longer explicit user modes. They are native runtime obligations.

The runtime always performs:

- semantic state extraction,
- redundancy reduction,

- active-objective prioritization,
- stable/provisional/open separation,
- continuity shaping.

10.2. Processing Pipeline

The always-on pipeline is:

Context Detect → Capsule Detect → Capsule Parse → State Partition → Redundancy Reduction → Objective

In algorithmic form:

Listing 2: Always-on continuity pipeline pseudocode

```
function continuity_pipeline(input):
    raw_context      = detect_context(input)
    capsule_state    = detect_capsule(raw_context)
    parsed_state     = parse_capsule_semantically(capsule_state)
    merged_state     = merge_with_new_user_input(parsed_state, input)
    partitions       = partition_state(merged_state)
    reduced_state    = reduce_redundancy(partitions)
    prioritized      = prioritize_main_objective(reduced_state)
    review_model     = build_human_review(prioritized)
    final_handoff    = build_final_handoff(prioritized)
    diagnostics      = build_advanced_diagnostics(prioritized)
    return (review_model, final_handoff, diagnostics)
```

11. Review Window Design

11.1. UX Principle

The review surface must be:

- inspection-first,
- human-readable,
- executive-friendly,
- raw-data-second.

11.2. Required Section Order

The review surface should render in this order:

1. Clean Summary
2. Active Objective
3. Stable Core
4. New / Provisional
5. Open / Unresolved
6. What Changed
7. Recommended Next Actions
8. Advanced Diagnostics (collapsed)

11.3. Advanced Diagnostics

Advanced Diagnostics must:

- be collapsed by default,
- contain raw/internal state only,
- never dominate the primary reading experience.

12. Governance Model

12.1. Purpose

Governance ensures that not every new instruction becomes stable truth.

12.2. Governance Responsibilities

- Protect stable core from accidental overwrite
- Hold new instructions provisionally
- Preserve unresolved items without forcing premature closure
- Suppress stale or redundant provisional items
- Score session health
- Recommend promotion only when justified

12.3. Stable vs Provisional vs Open

- **Stable Core:** high-confidence accepted workflow state
- **New / Provisional:** changes not yet accepted
- **Open / Unresolved:** risks, questions, uncertainties intentionally preserved

13. Provider Routing Strategy

13.1. Supported Provider Targets

Prompt Accelerator currently targets:

- ChatGPT
- Claude
- Gemini
- Grok

13.2. Internal vs External Representation

Internally, continuity state must remain provider-agnostic.

Externally, handoff rendering may differ by provider:

$$\text{Provider-Agnostic State} \rightarrow \text{Provider Profile} \rightarrow \text{Targeted Handoff Format}$$

13.3. Provider Profiles

A provider profile may specify:

- structural preference,
- instruction density,
- section ordering,
- response framing,
- compactness bias.

13.4. Routing Function

Define provider routing as:

$$\mathcal{T}_p : W_t \mapsto H_t^{(p)}$$

where $p \in \{\text{chatgpt}, \text{claude}, \text{gemini}, \text{grok}\}$ and $H_t^{(p)}$ is the provider-targeted handoff artifact.

14. Browser-Extension Blueprint

14.1. Extension Role

The extension is the live surface layer for:

- surface detection,
- draft read,
- default runtime processing,
- human-readable Advanced review,
- apply/writeback,
- workflow/capsule save actions.

14.2. Supported Surface Requirements

For each supported surface:

- detect active draft area,
- read current draft safely,
- apply transformed output safely,
- avoid duplicate insertion,
- avoid brittle runtime assumptions,
- preserve review/apply integrity.

14.3. Always-On Runtime Gate

The browser extension is blessable only when:

- default UX contains no Compress/Focus concept,
- Advanced is inspection-first,
- repeated Advanced/Apply passes do not create duplication or drift,
- saved/reloaded continuity state remains clean,
- constraints normalize across slight wording changes,
- stale provisional items do not keep resurfacing,
- ChatGPT, Claude, Gemini, and Grok each pass:
  - surface detection,
  - draft read,
  - Advanced open,
  - Apply writeback,
  - no duplicate insertion,
  - no console/runtime errors.

15. Desktop-MVP Blueprint

15.1. Desktop Role

The desktop application extends the same shared continuity core into:

- persistent workspaces,
- local versioned persistence,
- provider handoff generation,
- capsule/workflow inspection and management,
- continuity console UX.

15.2. Shared-Core Extraction

The core architectural move is:

$$\text{Extension-Specific Shell} \oplus \text{Shared Continuity Core}$$

where:

- the extension shell handles browser surfaces,
- the shared core handles continuity logic,
- the desktop app reuses the same continuity core.

16. Monorepo / Module Blueprint

A practical monorepo blueprint:

Listing 3: Recommended repository shape

```
apps/
  extension/
  desktop/

packages/
  continuity-types/
  continuity-core/
  continuity-governance/
  continuity-storage/
  continuity-domain/
  continuity-routing/

docs/
  technical-white-paper.tex
```

16.1. Package Responsibilities

Package	Responsibility
`continuity-types`	canonical shared contracts and schemas
`continuity-core`	pipeline, parsing, extraction, review generation
`continuity-governance`	stable/provisional/open logic, promotion, health
`continuity-storage`	local persistence envelopes and IO
`continuity-domain`	higher-level services and orchestration rules

`continuity-routing`	provider profiles and handoff generation

17. Data Persistence Blueprint

17.1. Local-First Rule

For the browser extension and desktop MVP, persistence is local-first:

- no mandatory backend,
- no required account,
- no hidden telemetry,
- no silent cloud sync.

17.2. Versioned Envelopes

Persisted artifacts should use versioned envelopes:

Listing 4: Versioned persistence envelope

```
{
  "version": 1,
  "kind": "workspace",
  "id": "workspace-uuid",
  "updated_at": "2026-01-01T00:00:00Z",
  "payload": { }
}
```

Kinds include:

- workspace
- session
- capsule
- workflow
- diagnostics
- export

18. Algorithms

18.1. Semantic Capsule Parsing

Listing 5: Capsule parsing algorithm sketch

```
function parse_capsule_semantically(capsule):
    objective   = summarize_objective(capsule.active_objective)
    constraints = normalize_constraints(capsule.stable_constraints)
    decisions   = normalize_decisions(capsule.accepted_decisions)
    open_items  = normalize_open_questions(capsule.open_questions,
                                           capsule.unresolved_risks)
    next_steps  = normalize_next_actions(capsule.next_actions)
    metadata    = extract_metadata(capsule)
    return {
        objective,
        constraints,
        decisions,
        open_items,
        next_steps,
        metadata
    }
```

18.2. Stable/Provisional Separation

Listing 6: Partition algorithm sketch

```
function partition_state(prior_state, new_input):
    stable = derive_stable(prior_state)
    provisional = derive_new_or_changed(new_input, stable)
    unresolved = derive_unresolved(prior_state, new_input)
    return {
        stable_core: stable,
        provisional: provisional,
        open_unresolved: unresolved
    }
```

18.3. Human Review Build

Listing 7: Human-readable review assembly sketch

```
function build_human_review(partitions, diagnostics):
    return {
        clean_summary: summarize_state(partitions),
        active_objective: summarize_objective(partitions.stable_core),
        stable_core: render_bullets(partitions.stable_core),
```

```
        provisional: render_bullets(partitions.provisional),
        open_unresolved: render_bullets(partitions.open_unresolved),
        what_changed: summarize_deltas(partitions),
        next_actions: recommend_next_actions(partitions, diagnostics),
        advanced_diagnostics: diagnostics
    }
```

19. Success Criteria

19.1. Runtime Success

The system is succeeding when:

- users no longer think in terms of manual modes,
- the runtime always tightens and stabilizes the workflow,
- review surfaces are easier to read than raw prompt state,
- workflow state survives across sessions,
- continuity can be transferred across providers.

19.2. UX Success

A user should never think:

> "Which mode should I choose?"

They should feel:

> **LuxCrypta is already preserving, tightening, and stabilizing the workflow.**

20. Security, Privacy, and Legal Posture

20.1. Privacy Posture

The intended privacy posture is:

- local-first,
- no hidden telemetry,

- no silent sync,
- manual export/import,
- explicit consent where applicable.

20.2. Operational Caution

Prompt Accelerator is a workflow tool, not a guarantee of model correctness. It should not be represented as deterministic control over model internals.

21. Roadmap

21.1. Phase 1: Browser Extension Standardization

- always-on runtime behavior
- human-readable Advanced review
- Grok support parity
- tighter capsule parsing

21.2. Phase 2: Shared Core + Desktop Runtime

- core extraction
- desktop continuity console
- local workspace persistence
- provider handoff preview/copy

21.3. Phase 3: Distribution Hardening

- desktop installers
- signing / notarization
- platform smoke validation

21.4. Phase 4: Expanded Continuity Infrastructure

- optional sync
- richer capsule diffing
- deeper continuity analytics
- broader enterprise workflow surfaces

22. Conclusion

Prompt Accelerator is best understood not as a prompt utility, but as continuity infrastructure for AI workflows. Its long-term value lies in reducing workflow entropy: preserving objectives, constraints, decisions, and unresolved questions across time, surfaces, and providers.

The architecture described here is intentionally modular and local-first. The browser extension proves the concept at the edge of existing LLM interfaces. The desktop runtime extends that concept into a persistent workspace model. The shared core defines the actual product.

In final form, Prompt Accelerator transforms AI interaction from disposable chat into structured continuity.

Appendix A: Canonical Terminology

Term	Definition
Continuity Runtime	The always-on processing layer that preserves and shapes workflow state
Stable Core	High-confidence accepted objective, constraints, decisions, and output rules
Provisional State	New or changed material not yet promoted into stable state
Open / Unresolved	Questions, risks, or uncertainties intentionally kept open
Carry-Forward Capsule	Portable continuity artifact for later reuse or transfer
Redundancy Reduction	Automatic removal of repeated or low-information phrasing
Objective Prioritization	Automatic shaping toward the most important current objective
Advanced Diagnostics	Secondary raw/internal diagnostic view, collapsed by default
Workflow Entropy	Drift, sprawl, fragmentation, or instability in an AI workflow over time

Appendix B: Representative Review Window Blueprint

```
Advanced Review
---------------
1. Clean Summary
2. Active Objective
3. Stable Core
4. New / Provisional
5. Open / Unresolved
6. What Changed
7. Recommended Next Actions
8. Advanced Diagnostics (collapsed)
```

Appendix C: Minimal Blessing Rule

A user should never think "which mode should I choose?" They should feel LuxCrypta is already preserving, tightening, and stabilizing the workflow, with Advanced available only for visibility or deeper control.