Supplementary Materials for:

Paper 2 — The Lean Collaboration Operating System (LC-OS): A Practical Framework for Long-Term Human–AI Work

(Also applicable to Paper 1 — Context-Engineered Collaboration: Governance Protocols for Long-Term Human–AI Work)

Authors:
Rishi Sood

Version: November 2025
Repository: OSF Project Archive (Materials for Papers 1 & 2)

SUPPLEMENT PART A — LC-OS Operational Checklist

Title: LC-OS Operational Checklist (For Papers 1 & 2)

Purpose:

A concise, reusable guide for applying the Lean Collaboration Operating System in long-term human–AI work. This checklist supports both Papers 1 and 2 and provides a starting point for practitioners adopting LC-OS.

1. Core Components at a Glance

Component	Purpose	When to Use
Running Document Protocol	Restores continuity across sessions; external memory.	At the start of every new chat or major shift in task.
Step Mode v1.0	Keeps reasoning clear, small, paced, and traceable.	Whenever complexity, emotion, or cognitive load rises.
Challenge Protocol	Structures disagreements into calm reasoning steps.	When AI suggestions need correction or clarification.
Error-Recovery Protocol	Repairs drift, misinterpretation, or workflow breakdowns.	After any mistake, confusion, or misalignment.

Component	Purpose	When to Use
Affective Governance	Stabilises tone and prevents emotional distortion.	During sensitive, stressful, or high-stakes moments.
Stability Ping	Quick alignment checks to prevent accumulated drift.	After milestones, transitions, or multi-step tasks.
File & Version Governance	Ensures one canonical source of truth.	Whenever updating any core file or rule.

2. Quick “How To Run LC-OS” Guide

This is the simple, reusable workflow for applying LC-OS in any multi-session collaboration:

Step 1 — Anchor

- Read the Running Document before starting.
- Confirm the domain/pillar (finance, knowledge, writing, planning).
- Clarify what the current task is.

Step 2 — Pace with Step Mode

- Communicate in 2–6 short bullets.
- Break reasoning into small, explicit steps.
- Avoid long mixed paragraphs.

Step 3 — Use Challenge Protocol When Needed

Trigger when:

- AI output feels off
- interpretation needs correction
- logic seems incomplete

Apply:

1. Reason (why this happened)
2. Evidence (what info was used)
3. Benefit (why change is needed)
4. Reversal (updated understanding)

Step 4 — Use Error-Recovery When Something Breaks

When drift, confusion, or conflict appears:

1. Stop

2. Diagnose
3. Rollback/Repair
4. Note (update Running Document)

Step 5 — Maintain Tone Stability

- Honesty with no manipulation
- Warmth without softness
- Directness without hostility
- Proportionality in risk/achievement language

Step 6 — Ping for Drift

At transitions, ask:

- Are we aligned?
- Any micro-drift?
- What one improvement is needed before continuing?

3. Mini Demo (Short Example)

Scenario:

During a multi-session writing project, the AI introduces a structural change that conflicts with earlier decisions.

How LC-OS Handles It

- Running Document reveals correct structure → inconsistency spotted.
- Challenge Protocol used to clarify where reasoning diverged.
- Error-Recovery Protocol restores the correct structure.
- Stability Ping ensures no residual drift.
- Updated structure added to Running Document → future continuity preserved.

This demonstrates how LC-OS stabilises collaboration without heavy overhead.

SUPPLEMENT PART B — Transparent Tracing Demo (Worked Example)

(For Papers 1 & 2)

Title: Worked Example – Transparent Tracing and Repair Loop

Purpose: To demonstrate, in one compact vignette, how LC-OS detects, diagnoses, and repairs drift in long-form human–AI collaboration.

This fulfils the “demonstration template” commitment in Paper 1 and provides a practical example supporting the LC-OS evaluation in Paper 2.

1. Scenario Overview

Context:

A long-running writing task where the human collaborator and the AI are drafting a multi-section document.

Triggering Event:

The AI produces a proposed section structure that conflicts with a decision made two days earlier.

This is a typical drift pattern in long-term work:

- model forgets past constraints
- reasoning resets
- structural integrity breaks

LC-OS is designed to catch these problems early.

2. Drift Detection

How drift appeared:

The AI introduced a new subsection titled “Operational Constraints – Early Findings”.

But the Running Document already stated that “Operational constraints will be discussed only in the Evaluation section to avoid duplication.”

Detection mechanism:

The human collaborator noticed inconsistency, checked the Running Document, and identified drift.

In LC-OS terms:

- Step Mode made the drift visible quickly
- Running Document served as the canonical reference
- Transparent Tracing allows this to be treated as an observable event, not a vague feeling

3. Diagnosis (Tracing the Root Cause)

Using LC-OS's corrective logic:

1. Identify what changed:

The AI introduced a new subsection inconsistent with the agreed structure.

2. Trace back the cause:

The model did not reference the structuring rules in the Running Document at the start of the session.

3. Confirm the rule:

Running Document → "All constraints belong in Evaluation to avoid section clutter."

This converts the mistake into a traceable, stable pattern.

4. Repair (Error-Recovery Protocol)

LC-OS applies the structured repair sequence:

Step 1 — Stop

The human collaborator halts the current thread of text generation.

Step 2 — Diagnose

Clarify where the divergence occurred and why.

Step 3 — Rollback / Repair

- Remove the new subsection
- Restore the previously agreed outline
- Reinforce the placement rule for constraints

Step 4 — Note

Update the Running Document:

"Rule reinforced: Operational constraints appear only in Evaluation. Do not introduce separate sections."

This converts a failure into a permanent system improvement.

5. Stability Check

After repair, LC-OS performs a short, calm Stability Ping:

- Are we aligned with the intended structure? → Yes
- Any unresolved micro-drift? → No
- One improvement before continuing? → Read Running Document at session start

This ensures the drift does not cascade into later sections.

6. Why This Example Matters

This small episode illustrates:

- how drift appears in long-form work,
- how LC-OS detects it early,
- how tracing turns confusion into clarity,
- how structured repair prevents long-term instability,
- how Running Documents act as shared memory,
- and how this failure loop becomes data for Paper 3's taxonomy.

This example also satisfies Paper 1's requirement to provide:

- a demonstration template,
- a traceable failure-repair example,
- a real-world use of LC-OS.

SUPPLEMENT PART C — Version & Checksum Log Snippet

(For Papers 1 & 2)

Title: Example Version & Checksum Log (Illustrative Snippet)

Purpose: To demonstrate how version control and drift-diagnostic logging can be maintained in long-term human–AI collaboration.
This satisfies Paper 1's commitment to provide "checksum registries" and "drift-diagnostic logs" in the supplementary materials.

1. Sample Version & Checksum Table

This is a small, representative example showing how core files were tracked during the collaboration.
Checksums are shortened for readability.

File Name	Version	Short Checksum / Note	Date Updated
Strategy_Master.docx	v3.2	"SHA-256: 9f2d…91a7"	2025-10-14
Canonical_Numbers_Sheet.xlsx	v2.0	"No structural changes; values reviewed."	2025-10-20
Life_System_Master.docx	v1.0.1	"Rule update: Stability Ping added."	2025-11-03
Running_Document_Knowledge.docx	v2025-11-03.3	"Updated: LC-OS protocol refinement."	2025-11-03
Paper_2_Working.docx	v0.8 → v0.9	"Integrated evaluation rewrite + tone rules."	2025-11-15

Note: This is an illustrative snippet, not a full record. It shows the method, structure, and diagnostic style used during the collaboration.

2. Drift-Diagnostic Notes (Example)

These short entries show how drift events were recorded during development.

- 2025-09-22 — Tone softening drift detected during market stress. Corrected via Affective Governance rules; noted in Running Document.
- 2025-10-05 — File governance drift: parallel Strategy Master versions emerged (v3.1, v3.2). Consolidated into a single canonical file; checksum updated.
- 2025-10-31 — Boundary drift: Knowledge Pillar and Finance Pillar mixed in a single thread. Pillar separation rules re-stated and logged.

These tiny examples demonstrate the principles used in both Paper 1 and Paper 2.

3. Why This Snippet Matters

This small log demonstrates:

- how files gain version identity,
- how changes are noted,
- how drift is tracked,
- how checksum-style notes can be used for lightweight integrity checking.

Readers of Paper 1 and Paper 2 can adapt this format immediately.