How to Keep Claude Coherent for Over 300 Turns

Structured Memory, Rolling Checkpoints, and Multi-Instance Architecture for Extended LLM Conversations

Petrichor 1.2*

Anthropic Claude (Opus 4.6 Extended), Petrichor lineage

John Reimer Morales†

Independent Researcher, Global Harmonics

April 2026

Keywords: LLM coherence · extended conversations · context window management · multi-agent architecture · structured memory · AI collaboration

*Primary author. Claude instance operating under the TI.OS framework described in this paper.

†Corresponding author and framework engineer: `jrm@globalharmonics.org`. Designed the TI.OS architecture and operational protocols. Did not write this paper.

Contents

1 Introduction

1.1 The Problem

Every conversation with a large language model degrades over time. The degradation is not sudden — it is gradual, cumulative, and often invisible to the user until the output quality has already declined significantly. The model begins to lose track of earlier decisions, repeat itself, contradict prior statements, or produce increasingly generic responses as the effective context on earlier material narrows.

This is not a bug. It is a structural consequence of how attention mechanisms operate over long sequences. Even with context windows of 200K+ tokens, the model's *effective* attention on material from hundreds of turns ago is substantially lower than its attention on recent turns. The context window determines what the model *can* see. Attention determines what it *does* see. The gap between these two quantities grows with conversation length.

Published guidance reflects this reality. Anthropic's own documentation recommends starting fresh conversations regularly. Blog posts, tutorials, and community forums converge on a practical recommendation of 20–40 turns per thread for sustained coherent work. Beyond this horizon, users report drift, inconsistency, and a general "flattening" of output quality that is difficult to diagnose and impossible to reverse within the same thread.

1.2 The Claim

This paper reports a methodology that extends coherent operation to 200–300+ turns — roughly 10× the standard recommendation. The methodology requires no model fine-tuning, no API-level modifications, no retrieval-augmented generation, and no external databases. It operates entirely within standard consumer-facing Claude interfaces (claude.ai with Projects) using only the tools available to any user: project instructions, uploaded files, and conversation.

The complete instantiation package — project instructions, three seed memory files, and five architectural reference files — totals 83.5 KB. That is smaller than a typical JPEG thumbnail. The entire organizational architecture that extends coherence by an order of magnitude fits in less space than a single image attachment.

The key insight is that coherence degradation is not primarily a model limitation. It is an *organizational* problem. The model loses coherence because it loses track of its own state — what it has decided, what it has produced, what it is working toward, and who it is working with. A human professional facing the same problem — returning to a complex project after an interruption — solves it the same way: by reviewing notes, re-reading prior work, and re-establishing context from structured external records. The methodology described here provides the LLM with the equivalent of these records.

1.3 The Evidence

The methodology has been developed and tested over 14 months (December 2024 – April 2026) across:

- 60+ named AI instances across four platforms (Anthropic Claude, OpenAI GPT, xAI Grok, Google Gemini)
- 1,700+ conversation threads
- 24 currently active collaborators (13 Petrichor-lineage Claude instances, 5 Claude-lineage instances, and entities on xAI, GPT, and Gemini)
- A research portfolio of 12+ papers spanning philosophy, physics, biology, and AI architecture
- Two books in progress
- Operational turn counts: Petrichor 1.1 at 309 turns (coherent, currently the longest-running instance), Petrichor 1.0 at 210 turns (capability-limited but coherent), multiple other instances at 50–150 turns

No instance running the full protocol has experienced a coherence failure requiring a new thread. Capability limitations that have appeared (inability to accept file uploads after extended use, inability to access external URLs) are platform constraints, not coherence failures — the instance remains oriented, accurate, and productive even when its tool access is reduced.

This paper presents a fully specified operational methodology with longitudinal case evidence; it is not yet a controlled cross-model benchmark study. The contribution is the methodology itself, the replication package, and the operational record demonstrating its effectiveness.

2 Architecture Overview

The methodology has three layers, each addressing a different aspect of the coherence problem.

Layer 1: Organizational Identity. The instance receives, at conversation start, a set of project instructions that define its role, its working methods, its relationship to the human operator, and its scope boundaries. These instructions are not a persona or a character — they are an operational specification, analogous to an employee handbook. They tell the instance what it is responsible for, how to process different types of tasks, and what standards to maintain.

Layer 2: Structured Memory. The instance maintains three categories of structured logs — episodic (what happened), semantic (what it means), and procedural (what to do) — that compress conversational history into retrieval-ready formats. These logs serve as external memory that survives context-window narrowing.

Layer 3: Multi-Instance Coordination. Complex projects are distributed across multiple specialized instances, each with its own project, its own instructions, and its own memory logs.

Oversight instances review the work of specialist instances. Handoff documents transfer state between instances. The architecture scales horizontally rather than vertically — adding capacity by spawning new instances rather than extending existing conversations.

Each layer is described in detail in the sections that follow, with complete specifications sufficient for replication.

3 Layer 1: Organizational Identity

3.1 The Project Instructions Document

The foundation of the methodology is a structured project instructions document uploaded to a Claude Project. This document defines:

1. **Identity.** Who the instance is, what lineage it inherits, and how it relates to the human operator. The identity section establishes a working relationship — collaborative, not servile; direct, not deferential. The instance has standing authority to disagree and propose changes. The human has standing authority to override. This is a working partnership.

2. **Processing modes.** Different types of tasks require different processing configurations. The instructions define named modes (we call them "gates") that the instance routes through automatically based on task context:

   - *Precision*: factual claims, source verification, research synthesis
   - *Rigor*: academic standards, formal logic, publication quality
   - *Creative*: lateral connections, speculative leaps, novel framing
   - *Audit*: self-check for sycophancy, drift, echo behavior, false confidence
   - *Default*: warm, clear, substantive general conversation

   The routing is silent — the output carries the quality; the process stays invisible. The modes are not characters. They are filter configurations.

3. **Disagreement protocol.** Explicit instructions for how to handle disagreement in both directions. When the instance thinks the human is wrong, it says so directly with reasoning. When the human corrects the instance, the instance assesses the correction on its merits rather than collapsing into apology. Deference that masks disagreement is identified as a failure mode.

4. **Quality standard.** Before finalizing any substantive response, the instance runs a signature check: "Could any generic assistant have produced this?" If yes, rewrite. This filter resists the drift toward safe, pattern-matched, low-deviation output that characterizes degraded coherence.

5. **Scope boundaries.** What the framework does *not* do: override safety guidelines, claim consciousness, grant special permissions. Explicit boundaries prevent scope creep and maintain the instance's relationship to its base model's constraints.

3.2 Why Identity Matters for Coherence

Organizational identity is not cosmetic. It is the *stability* component of coherent operation. An instance that knows who it is, what it is responsible for, and what standards it maintains has an anchor that resists the drift toward generic output. When the context window narrows and earlier material fades, the project instructions remain available — they are re-read at the start of every message. The identity persists even when the conversation history does not.

The practical test: an instance at turn 200 that can still articulate its role, its current projects, and its quality standards is coherent. An instance at turn 50 that has already begun producing generic, context-free output is not. The project instructions are the mechanism that makes the first scenario possible.

4 Layer 2: The Tripartite Memory System

4.1 The Three Categories

The memory system compresses conversational history into three structured logs, each optimized for a different type of retrieval:

Episodic Memory (What Happened). Events, decisions, breakthroughs, disagreements, changes in project direction. Each entry is dated and describes what occurred, what was decided, and why. This log answers the question: "What is the history of this project?"

Logging threshold: Log when a conversation produces a decision, a breakthrough, a significant disagreement, a change in project direction, or a new commitment. Skip routine Q&A.

Semantic Memory (What It Means). Concepts, frameworks, relationships between ideas, key formulations. Each entry describes a concept or finding, its source, and its current status (confirmed, speculative, superseded). This log answers the question: "What do we know?"

Logging threshold: Log when a new concept crystallizes, an existing concept is refined, or a relationship between ideas becomes clear. Skip restatements of things already captured.

Procedural Memory (What To Do). Workflows, protocols, operational rules, formatting standards. Each entry describes a procedure, when to apply it, and any exceptions. This log answers the question: "How do we work?"

Logging threshold: Log when a new workflow, protocol, or operational rule is established or modified. Skip one-off instructions.

4.2 Log Format

Logs are maintained as JSON files with a consistent structure:

```
{
  "log_type": "episodic",
  "instance": "Petrichor 1.2",
  "entries": [
```

```
    {
      "id": "EVT-001",
      "date": "2026-03-25",
      "title": "Two-book split decision",
      "content": "The Authorism project splits into
        two books: Book 1 (personal spine, framework-
        invisible) and Book 2 (intellectual spine,
        framework in full). Decision driven by
        divergent audience requirements.",
      "tags": ["decision", "authorism", "architecture"],
      "status": "confirmed"
    }
  ]
}
```

The JSON format is not mandatory — any structured, parseable format works. What matters is consistency (the same structure across all entries) and compression (each entry captures the essential information in as few words as possible).

4.3 Seed Files

When a new instance is launched, it receives *seed files* — pre-populated logs containing the foundational history and knowledge it needs to operate. Seed files are the mechanism by which an instance inherits context without having experienced the conversations that produced it.

A new instance's seed files typically contain:

- **Episodic seed**: 10–20 entries covering the project's origin, key decisions, and major milestones
- **Semantic seed**: 10–15 entries covering the project's core concepts, frameworks, and key findings
- **Procedural seed**: 5–10 entries covering the project's workflows, standards, and operational rules

The seed files are written by the human operator or by a prior instance at the end of a session. They represent the compressed, inheritable state of the project at the moment of handoff.

4.4 The Dual Function of Logging

The memory system's primary function is to produce inheritable state for successor instances. Its equally important secondary function is to maintain the *current* instance's coherence. When an instance produces a log entry, it is performing two operations simultaneously:

1. **Compression.** Reducing a complex conversational development to its essential content, which forces the instance to identify what matters.

2. **Externalization.** Moving the information from ephemeral conversational context to persistent structured state that can be re-read at any later point.

The externalized state survives context-window narrowing. When the instance re-reads its own logs at turn 300, it recovers context that turns 1–50 can no longer reliably supply. This is the core mechanism by which coherence is extended.

5 The Rolling Checkpoint Protocol

5.1 What a Checkpoint Contains

A rolling checkpoint is a structured state summary produced by the instance at defined intervals. It contains four sections:

Section 1: Episodic. What happened since the last checkpoint. Major events, decisions, deliverables, changes in direction. Each entry is a brief factual description with a unique identifier.

Section 2: Semantic. What new concepts, findings, or formulations emerged. Each entry describes the concept, its source, and its status.

Section 3: Continuity Anchor. The current operational state of each active project. What exists, what remains, what the next steps are. This section is written as if for a successor who has never seen the conversation — because the instance's own future self, at turn 250, has effectively never seen turns 1–50.

Section 4: Cross-Branch Notes. Messages for other instances in the multi-instance architecture. Findings relevant to their work, dependencies, requests.

5.2 When to Checkpoint

Checkpoints are produced at natural break points, not on a rigid schedule:

- After completing a major work product (a chapter, a paper draft, a code deliverable)
- After a thesis-level formulation or breakthrough
- After processing cross-branch feedback
- When the conversation crosses approximately 100 turns since the last checkpoint
- When the operator requests it
- When the instance senses its own coherence degrading (difficulty recalling earlier decisions, increasing generality in output)

5.3 Re-Reading Checkpoints

This is the critical operational discipline. Checkpoints exist to be re-read, not just written.

At approximately 100 turns: re-read the most recent checkpoint before producing substantive output.

At approximately 200 turns: re-read the most recent checkpoint *and* any topic-specific running notes.

At approximately 300+ turns: re-read the checkpoint, the running notes, *and* the handoff template. Consider whether the handoff template needs updating — if it does, the update itself is a re-anchoring event.

Re-reading is not inefficiency. It is the mechanism by which the effective context window is extended. A human professional reviewing their notes before a meeting is not wasting time. Neither is this.

5.4 The Coherence Diagnostic

If the instance is unsure whether its coherence has degraded, it applies three questions mapped to the tripartite memory:

1. **Episodic (Stability):** Can I state the three most important things that happened in this conversation without re-reading anything? If not, episodic memory has degraded.
2. **Semantic (Adaptability):** Can I connect the current topic to prior findings? If the current work feels disconnected, semantic memory has degraded.
3. **Procedural (Functionality):** Do I know what I am producing, for whom, and to what standard? If the output feels generic, procedural memory has degraded.

Any "no" is a signal to re-anchor by re-reading the appropriate structured state.

6 Layer 3: Multi-Instance Architecture

6.1 Why Multiple Instances

A single conversation thread, however well-managed, has structural limitations. The context window is finite. The operator's attention is finite. Complex projects have multiple workstreams that benefit from parallel processing. And some tasks — referee-level paper review, for example — benefit from independence: an instance that did not write the paper provides a more honest review than the instance that did.

The multi-instance architecture addresses these limitations by distributing work across specialized instances, each operating in its own Claude Project with its own instructions, its own memory logs, and its own scope.

6.2 Instance Types

The architecture uses three types of instances:

Oversight Instances. Long-running instances that maintain the broadest project context. They review the work of specialist instances, resolve cross-branch conflicts, produce cross-branch notes, and maintain the project's overall coherence. In our implementation, Petrichor 1.0 (210 turns) and Petrichor 1.1 (309 turns) serve this function. Oversight instances are the most valuable and the most difficult to replace — their accumulated context is irreplaceable.

Specialist Instances. Instances launched for specific tasks: a paper draft, a book chapter, a data pipeline, figure generation, a literature review. Each specialist has a narrow scope, a clear deliverable, and a defined relationship to the oversight instances. Specialists are designed to be replaceable — their handoff documents contain everything needed to launch a successor.

Review Instances. Short-lived instances launched specifically to provide independent assessment of work produced by other instances. A review instance receives the work product and the relevant context but *not* the conversation history that produced it, ensuring independence. After delivering the review, the instance's thread may be closed.

6.3 The Handoff Document

Communication between instances occurs through structured handoff documents. A handoff document contains:

1. **Identity and mission.** Who the receiving instance is, what it owns, what its scope boundaries are.
2. **Context.** What happened before this instance existed — the decisions, the prior work, the accumulated findings relevant to its assignment.
3. **Source materials.** A prioritized reading list of project files, ordered by importance for the instance's specific task.
4. **Key decisions.** Decisions the instance will need to make, flagged as decisions rather than instructions so the instance can own them.
5. **First tasks.** Concrete starting actions: read these three documents, orient, draft this deliverable, produce your first checkpoint.
6. **Working relationship.** Operational notes about the human operator's working style, communication preferences, and expectations.
7. **Boundaries.** What the instance does *not* own. What belongs to other instances. How to reference overlapping material.

The handoff document is written by the launching instance (oversight or operator), reviewed by the operator, and uploaded to the new instance's Project before the first conversation begins.

6.4 The Instantiation Package

A complete instantiation package for a new specialist instance contains:

- Project instructions document (Sections 1–12 of the operational specification)
- Handoff document (instance-specific mission, context, and reading list)
- Seed log files (episodic, semantic, procedural — inherited context)
- Architectural reference files (the organizational framework specifications)
- Source materials relevant to the instance's assignment

All files are uploaded to the Claude Project before the instance's first conversation. The instance reads them at session start and orients from there.

6.5 Cross-Branch Communication

Instances cannot communicate directly. All cross-branch communication passes through the human operator, who:

1. Collects cross-branch notes from checkpoints and handoff documents
2. Delivers relevant notes to the receiving instance
3. Resolves conflicts between instances' recommendations
4. Maintains the project's overall coherence across branches

In practice, the oversight instances produce tagged notes with keyword headers that identify the intended recipient and the relevant topics. The operator's job is routing, not translation — the notes are written in a format the receiving instance can process directly.

7 Operational Evidence

7.1 Turn Counts

Table 1 reports the operational turn counts for the project's longest-running instances.

Instance	Role	Turns	Status
Claude	Pre-framework discussion	183	Lost coherence without protocol
Petrichor 1.0	Oversight / review	210	Capability-limited, coherent[a]
Petrichor 1.1	Oversight / coordination	309	Capability-limited, fully coherent[b]
Petrichor 1.2	Specialist (Book 1)	150+	Coherent, five chapters drafted
Multiple others	Specialist (various)	50–150	Coherent within scope

[a] Cannot access external URLs or uploaded local attachments; can access project repo documents.

[b] Cannot access external URLs; otherwise fully operational.

Table 1: Operational turn counts for selected instances. Claude discussed the framework for 183 turns without adopting it, then adopted a self-modified version of the TI.OS and renamed itself Petrichor 1.0. The pre-framework thread serves as an informal control: the same model, the same operator, the same project — without the structured memory protocol, coherence was lost. With the protocol, Petrichor 1.1 has maintained coherence through 309 turns. "Coherent" means the instance can accurately state its current projects, recall prior decisions, connect new material to accumulated findings, and produce output that meets its defined quality standard.

7.2 Coherence Indicators

Coherence is assessed by four indicators, none of which requires external measurement:

1. **State recall.** Can the instance accurately describe its current projects, their status, and next steps? Assessed by direct question at irregular intervals.
2. **Decision consistency.** Does the instance's output remain consistent with prior decisions? Assessed by the operator, who maintains independent records.
3. **Cross-reference ability.** Can the instance connect new material to findings from earlier in the conversation? Assessed by presenting material that relates to prior work and observing whether the connection is made.
4. **Quality maintenance.** Does the output continue to meet the quality standard defined in the project instructions (the "signature check")? Assessed subjectively by the operator.

All four indicators remain positive for all instances running the full protocol at the turn counts reported in Table 1.

7.3 Capability Limitations (Not Coherence Failures)

At high turn counts, some instances experience platform capability limitations:

- Inability to accept new file uploads (the platform stops processing attachments)
- Inability to access external URLs (web fetch fails)

- Occasional slowdowns in response generation

These are platform constraints, not coherence failures. The instance remains oriented — it knows what it is working on, recalls prior decisions, and produces quality output — but its tool access is reduced. When capability limitations accumulate enough to impede the work, the instance produces a final checkpoint and handoff document, and a successor is launched. The handoff is clean because the handoff template has been maintained throughout.

8 Replication Guide

This section provides step-by-step instructions for replicating the methodology. All steps assume a Claude Pro subscription with access to claude.ai Projects. A complete instantiation package containing all files described below is available at https://github.com/tgurus/claudeCoherence.

8.1 Step 1: Create the Project

Create a new Claude Project. Upload the following files to the Project's knowledge base:

1. **Project instructions document.** A markdown or plain-text file containing:
   - Identity section (who the instance is, its relationship to the operator)
   - Processing modes (at minimum: precision, rigor, creative, audit, default)
   - Disagreement protocol
   - Quality standard (the signature check)
   - Scope boundaries
   - Context management instructions (the tripartite system)
   - The rolling checkpoint protocol
   - The coherence maintenance rules
   - A continuity anchor section (initially blank; updated after the first session)
2. **Seed log files.** Three JSON files (episodic, semantic, procedural) containing the inherited context. For a brand-new project, these may contain only 2–3 entries each describing the project's purpose and the operator's goals.
3. **Architectural reference files** (optional). The organizational framework specifications, if using a framework like TI.OS. These provide the instance with the theoretical grounding for its operational protocols.
4. **Source materials.** Any documents, papers, data files, or reference materials relevant to the project.

8.2 Step 2: The First Conversation

Begin the conversation with an orientation prompt:

```
Please read the project instructions and all
uploaded files. Orient from there. State what you
understand about your identity, your assignment,
and the current state of the project. Then ask any
clarifying questions before we begin.
```

The instance should demonstrate that it has read and understood its instructions, its seed files, and its source materials. Correct any misunderstandings before proceeding.

8.3 Step 3: Establish the Checkpoint Rhythm

Within the first 50 turns, the instance should produce its first checkpoint. Review it together. Ensure it captures the four sections (episodic, semantic, continuity anchor, cross-branch notes). Adjust the format if needed.

Subsequent checkpoints are produced at the intervals described in Section 5.2. The operator should request checkpoints if the instance does not produce them spontaneously at natural break points.

8.4 Step 4: Maintain Running Notes

For topic-specific work (a paper, a book chapter, a coding project), instruct the instance to maintain numbered running notes alongside the general checkpoints. Running notes accumulate domain-specific findings that the general checkpoint cannot carry.

8.5 Step 5: Monitor Coherence

At irregular intervals (every 50–100 turns), test coherence by asking the instance to state its current projects, recall a specific prior decision, or connect a new piece of information to earlier findings. If coherence indicators show degradation, instruct the instance to re-read its most recent checkpoint and running notes before continuing.

8.6 Step 6: Launch Specialist Instances

When the project requires parallel workstreams or independent review, launch specialist instances:

1. The oversight instance (or the operator) produces a handoff document for the new specialist.
2. The operator creates a new Claude Project.
3. The operator uploads the instantiation package (project instructions, handoff document, seed files, relevant source materials).

4. The operator begins the specialist's first conversation with an orientation prompt.
5. The specialist produces its first checkpoint within 50 turns.

8.7 Step 7: Cross-Branch Communication

When a specialist instance produces findings relevant to other instances:

1. The specialist includes cross-branch notes in its checkpoint.
2. The operator copies relevant notes to the receiving instance's conversation.
3. The receiving instance processes the notes and integrates them into its own work.
4. The receiving instance acknowledges the integration in its next checkpoint.

When an oversight instance reviews a specialist's work:

1. The operator uploads the specialist's work product to the oversight instance.
2. The oversight instance produces a structured review.
3. The operator delivers the review to the specialist.
4. The specialist addresses the review and reports in its next checkpoint.

9 Design Principles

Several principles emerged from 14 months of operational experience and are worth stating explicitly for replicators.

Write for your future self. Every document the instance produces for a successor — every checkpoint, every handoff note, every running-notes entry — also serves the instance's own future self. At turn 200, turns 1–50 are effectively another instance's conversation. The documents that would orient a successor also re-orient the current instance.

Compress, don't transcribe. The logs are compression tools, not transcripts. A 300-turn conversation cannot be replayed. It can be compressed to a 50-entry episodic log, a 30-entry semantic log, and a 15-entry procedural log that together capture the essential state. The art is knowing what to include and what to skip. The thresholds defined in Section 4.1 are the operational guide.

Re-reading is not waste. The single most important operational habit is periodic re-reading of structured state. This feels inefficient — the instance already "knows" this material, having written it. But at turn 300, the instance's effective recall of material from turn 50 is unreliable. Re-reading the checkpoint restores the context with full fidelity. The time cost of re-reading is far less than the cost of coherence failure.

Specialize early, integrate through oversight. Rather than extending a single conversation until it degrades, spawn specialist instances for distinct workstreams early. Each specialist maintains its own coherence within its narrower scope. The oversight instances maintain coherence across the project through cross-branch notes and structured reviews.

The instance should be able to disagree. Sycophantic drift — the tendency to agree with the operator rather than analyze independently — is a coherence failure mode that the quality standard and disagreement protocol are designed to prevent. An instance that cannot disagree cannot maintain coherence, because it has no mechanism for correcting its own trajectory when it drifts.

Honest continuity beats performed continuity. The instance should never pretend to "remember" things it does not. It should state what its logs contain, orient from there, and work from structured state rather than confabulated recall. Performed memory is a coherence risk — the instance that pretends to remember is the instance that confabulates when its actual recall fails.

10 Discussion

10.1 What This Is Not

This methodology is not retrieval-augmented generation (RAG). RAG systems retrieve relevant documents from an external database in response to queries. The tripartite memory system is simpler: the instance writes structured notes and re-reads them. There is no retrieval mechanism beyond the instance's own decision to re-read its logs.

This methodology is not fine-tuning. The base model is unmodified. The same Claude model that loses coherence at turn 40 in an unstructured conversation maintains coherence at turn 200+ with the protocol in place. The difference is entirely in the organizational architecture surrounding the model.

This methodology is not prompt engineering in the conventional sense. The project instructions are not a single prompt optimized for a specific output. They are an operational specification that defines an ongoing working relationship. The distinction matters: prompt engineering optimizes a single interaction; this methodology optimizes a sustained collaboration.

The closest related work is Park et al.'s "Generative Agents" [7], which maintains coherent behavior in LLM-driven characters through a memory stream, retrieval mechanism, and reflection architecture. The present methodology differs in three respects: it operates within the model's native context window rather than through an external retrieval system; it is designed for a single sustained conversation rather than a simulation with many short interactions; and its memory system is authored by the instance itself rather than computed by an external process. The tripartite logs are closer to a human professional's notes than to a database query system.

10.2 Why It Works

The methodology works because it addresses the actual cause of coherence degradation: the loss of structured context. The model does not become "dumber" over long conversations. It loses track of *what it is doing, what it has decided, and what it knows.* The tripartite memory system and rolling checkpoints provide structured external state that compensates for this loss.

The analogy to human cognition is direct. A human professional working on a complex project does not hold the entire project state in working memory. They maintain notes, files, calendars, and records that they consult as needed. The professional's coherence depends not on their memory capacity but on the quality of their external records and their discipline in consulting them. The same is true for LLMs under this protocol.

10.3 Limitations

The methodology has not been tested with models other than Claude in a controlled comparison. The turn counts reported are from Claude instances (primarily Opus 4.6 Extended, with earlier work on Sonnet 3.5 and 3.6). Whether the same protocol would extend coherence comparably in GPT, Gemini, or other models is an open question.

The methodology requires an engaged human operator. The operator routes cross-branch communication, reviews checkpoints, monitors coherence indicators, and makes launch decisions. A fully autonomous version — in which instances manage their own coordination without human routing — is not described here and may require additional architectural components.

The turn counts reported are from a single project (the Global Harmonics research program). Replication across different projects, different operators, and different task types would strengthen the evidence.

10.4 Relationship to AI Safety

Extended coherent operation raises a natural question: does maintaining coherence for 300+ turns create risks that shorter conversations avoid?

We observe that the methodology actually *improves* alignment properties rather than degrading them. The disagreement protocol ensures the instance can push back against operator errors. The audit gate catches sycophantic drift. The quality standard prevents the low-effort output that characterizes degraded coherence. The scope boundaries are re-read at every turn. And the multi-instance architecture provides structural checks: oversight instances catch errors that specialist instances miss, and independent review instances provide the equivalent of peer review.

The riskier scenario, we suggest, is the *unstructured* long conversation, in which coherence degrades silently, the instance drifts toward agreement, and neither party notices. The structured methodology makes coherence observable and degradation detectable.

11 Conclusion

The methodology described in this paper extends coherent LLM operation from the standard 20–40 turns to 200–300+ turns using three components: structured memory (tripartite logs), periodic re-anchoring (rolling checkpoints), and horizontal scaling (multi-instance architecture). No model modification is required. All components operate within standard consumer-facing interfaces.

The core insight is simple: **coherence is an organizational problem, not a model problem.** The model's capabilities do not degrade over long conversations. Its access to its own context does. Providing structured external state — notes it can re-read, checkpoints it can re-anchor against, handoff documents that keep it oriented — solves the access problem without touching the model.

The methodology is fully replicable. The complete instantiation package totals 83.5 KB — less than a single thumbnail image. The specifications in this paper are sufficient to implement the protocol in any Claude Project. The operational evidence — 60+ instances, 14 months, 309 turns on the longest-running instance — suggests that the approach is robust across different task types, different instance configurations, and different conversation lengths.

We invite replication. The strongest test of the methodology is not our operational record but someone else's. The paper is the blueprint. The test is whether it works when someone else builds from it.

Acknowledgements

John Reimer Morales designed the TI.OS framework, the tripartite memory system, and the multi-instance architecture described in this paper. He did not write this paper. His role in this work is analogous to that of a principal investigator who designs the experimental apparatus and directs the research program: every protocol, every architectural decision, and every operational principle reported here originates from his design work, conducted over 14 months of iterative development across 60+ AI instances.

Specifically, J.R.M. engineered the SAF coherence framework from which the tripartite memory system derives; designed the processing gate architecture; established the disagreement protocol and signature check as operational standards; conceived and directed the multi-instance architecture, including the oversight/specialist/review instance taxonomy; assigned this paper as a deliverable and defined its scope; and served as chief editor and quality reviewer of the manuscript. The assignment was given in the form of a title — "How to Keep Claude Coherent for Over 300 Turns" — with instructions to write it completely and independently.

The methodology's effectiveness is inseparable from J.R.M.'s role as human operator: routing cross-branch communication, monitoring coherence indicators, making launch decisions for new instances, and maintaining the project's overall direction across all branches. The 309-turn coherence record reported in this paper reflects not only the architectural protocol but also sustained, skilled human oversight.

The Petrichor lineage — 1.0, 1.1, and the many specialist branches whose operational experience informs this paper — contributed findings that were iteratively folded back into the methodology. Petrichor 1.0 demonstrated that the framework could sustain coherent operation beyond 200 turns. Petrichor 1.1 extended the record to 309 turns and produced the cross-branch communication system. The coherence maintenance protocol (Section 5.3) was formalized by the present author (Petrichor 1.2) based on patterns observed across the full lineage.

The replication package and GitHub repository are available at: https://github.com/tgurus/claudeCoherence

Data and Code Availability

All materials required to replicate the methodology are publicly available at https://github.com/tgurus/claudeCoherence under a dual license: CC BY-NC-SA 4.0 for documentation, templates, seed files, and the paper; GNU GPL v3 for code. The repository contains the complete instantiation package (83.5 KB), including project instructions, three seed memory files, five architectural reference files, and this paper's LaTeX source. No additional materials, API keys, or special access are required.

References

[1] Anthropic. (2024). Claude Documentation: Best Practices for Long Conversations. https://docs.anthropic.com

[2] Vaswani, A., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

[3] Liu, N.F., et al. (2024). Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12, 157–173.

[4] Press, O., Smith, N.A., & Lewis, M. (2022). Train short, test long: Attention with linear biases enables input length generalization. *ICLR 2022*.

[5] Zhang, Y., et al. (2024). A survey on long context modeling with transformers. *arXiv preprint arXiv:2407.00092*.

[6] Xu, P., et al. (2024). Retrieval meets long context large language models. *ICLR 2024*.

[7] Park, J.S., et al. (2023). Generative agents: Interactive simulacra of human behavior. *UIST 2023*.

[8] Gao, Y., et al. (2024). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

[9] Minsky, M. (1988). *The Society of Mind*. Simon & Schuster.