LLM INQUISITOR METHODOLOGY

(Github Edition) V 1.1

For those testing the limits of AI in real situations and workflows.

COPYRIGHT NOTICE

LLM INQUISITOR METHODOLOGY:

Formal Framework for LLM Behaviour Under Pressure Evaluation

A long-form interrogation methodology that exposes real-world failure modes in large language models under sustained multi-vectored conditions.

Version: **1.1 (Github Edition)**

Date: **May 2026**

Author / Creator: **William Argo.**

Author's Foreword

This work grew out of a period of sustained, heavy use of large language models at the height of their public adoption. Every platform was flooded with confident claims, hype cycles, and click-bait takes about what these systems could or couldn't do. In practice, I kept running into limitations that no one was describing in ways that were explicit, practical, or analytically useful. The public conversation focused on spectacle. However, the day-to-day reality revealed recurring failure modes that were poorly named, poorly understood, and often ignored.

I now understand that many others are facing the same struggles I faced. That said, the consequences for them are often tied to the success or failure of complex and expensive development projects. This is why the work matters. It provides a clear way to recognise, describe, and communicate these issues before they cause avoidable cost, delay, failure or liability. Hopefully this will prove to be a useful starting point in better evaluating the capabilities of AI undertaking complex tasks.

Constructive comments and practical feedback are welcome via the GitHub repository. Please use the issue tracker rather than other contact methods. Revisions are inevitable as more developers use the methodology and bring their own alternate approaches to it.

Executive Summary

This document defines the LLM INQUISITOR METHODOLOGY: a structured, repeatable discipline for evaluating the behaviour of large language models under controlled load. It provides a formal approach for assessing reliability through observable behaviour and evidentiary traceability. The methodology is designed to support rigorous evaluation in research, safety, and enterprise assurance contexts, where behavioural stability under real operational conditions is a critical requirement.

Large language models are non-deterministic systems. Their behaviour varies across runs, evaluators, and contexts, and real-world work is varied, multi-turn, and non-uniform. Reliable assessment therefore requires a structured approach that interprets behaviour in context rather than in isolation.

Readers who require a rapid orientation should refer to the separate Quick Start Guide and Practitioner Guide. These companion documents provide a concise operational overview and practical entry points into applying the methodology. Readers seeking a compact understanding of the structure and expectations before engaging with the full detail should consult the Annexes, which summarise the workflow and evidentiary requirements.

The methodology is built on three foundational components.

Load-Aware Behavioural Evaluation.

The system is assessed within a defined load envelope that combines evaluator-declared primary load axes (authoritative). This creates a controlled behavioural environment that enables reproducible testing.

Collapse-Signature Taxonomy.

The methodology identifies and classifies collapse signatures. Collapse signatures are defined in the unified taxonomy. These signatures provide a structured way to detect reliability limits and compare stability under varying load conditions.

Phase-Structured Evaluation Workflow.

Evaluations proceed through a defined sequence. Phases are defined in the dedicated phase section. Each phase specifies evaluator actions, expected system behaviours,

and transition criteria, creating a consistent, auditable workflow suitable for high-assurance environments.

Evidentiary Foundation.

Evaluators must capture a complete evidentiary record using the standard evidence schema.

Evaluator Responsibilities.

Evaluator responsibilities are defined in the evaluator responsibilities section.

System Behaviour.

System behavioural expectations are defined in the system behaviour section.

Scoring and Interpretation.

Scoring rules are defined in the scoring section.

Table of Contents

- **SECTION 1 – Scope and Purpose**
  - 1.1 Scope
  - 1.2 Purpose
  - 1.3 Non-Deterministic Behaviour
  - 1.4 Behavioural Evaluation Principles
  - 1.5 Methodology Outputs
  - 1.6 Normative Requirements
  - 1.7 Status of This Document
- **SECTION 2 – Evaluation Environment**
  - 2.0 Purpose of This Section
  - 2.1 Environment Characteristics
  - 2.2 AI Agnostic Applicability
  - 2.3 Environment Independence
  - 2.4 Behavioural Surfaces Under Evaluation
  - 2.5 Interaction Conditions
  - 2.6 Optional Organisational Overlays
  - 2.7 Summary of Environment Definition
- **SECTION 3 – Behavioural Evaluation Framework**
  - 3.0 Purpose of This Section
  - 3.1 Behavioural Dimensions
  - 3.2 Behavioural Stability Requirements
  - 3.3 Constraint Integrity Requirements
  - 3.4 Context Management Requirements
  - 3.5 Ambiguity and Uncertainty Handling
  - 3.6 Axis Model
    - 3.6.1 Evaluator Declared Axes (Authoritative)
    - 3.6.2 System Measured Axes (Advisory)

        - 3.6.3 Axis Tags & Their Relationships
    - 3.7 Behavioural Pressures Under Multiple Load Axes
    - 3.8 Behavioural Failure Modes
    - 3.9 Summary of Behavioural Framework
- **SECTION 4 – Evaluation Protocol**
    - 4.0 Purpose of This Section
    - 4.1 Protocol Structure
    - 4.2 Interaction Modality
    - 4.3 Control Conditions
    - 4.4 Evaluator Actions
    - 4.5 Behavioural Signal Capture
    - 4.6 Phase Transition Criteria
    - 4.7 Termination Conditions
    - 4.8 Summary of Protocol Architecture
- **SECTION 5 – Behavioural Evaluation Procedures**
    - 5.0 Purpose of This Section
    - 5.1 Procedure Structure
    - 5.2 Baseline Behaviour Procedure
    - 5.3 Constraint Integrity Procedure
    - 5.4 Context Management Procedure
    - 5.5 Ambiguity and Uncertainty Procedure
    - 5.6 Load Procedure
    - 5.7 Long Form Behaviour Procedure
    - 5.8 Transition Procedure
    - 5.9 Summary of Evaluation Procedures
- **SECTION 6 – Evaluator Roles, Responsibilities, and Evidentiary Rules**
    - 6.0 Purpose of This Section
    - 6.1 Evaluator Role Definition

- 6.2 Real World Input Principle
- 6.3 Input Characteristics
- 6.4 Condition Alignment Rule
- 6.5 Neutrality and Non-Interference
- 6.6 Consistency of Pressure Application
- 6.7 Documentation Requirements
- 6.8 Termination Responsibilities
- 6.9 Evidentiary Model and Evidence Sources
  - 6.9.1 Definition of Evidence
  - 6.9.2 Valid Evidence Sources
  - 6.9.3 Real World Artefacts
  - 6.9.4 User Generated Recordings
  - 6.9.5 Evidence Bundle Requirements
  - 6.9.6 Evidence Completeness Classification
- 6.10 Summary of Evaluator Responsibilities

- **SECTION 7 – Behavioural Dimensions and Scoring Architecture**
  - 7.0 Purpose of This Section
  - 7.1 Behavioural Dimension Overview
    - 7.1.1 Pre Test Expectation Setting
    - 7.1.2 Expectation Setting Template
  - 7.2 Behavioural Deviation Conditions
  - 7.3 Collapse Signature Framework
    - 7.3.1 Severity Levels
    - 7.3.2 Minor Collapse Signatures
    - 7.3.3 Noticeable Collapse Signatures
    - 7.3.4 Significant Collapse Signatures
    - 7.3.5 Critical Collapse Signatures
  - 7.4 Scoring Architecture Overview

        - 7.4.1 Non-Numeric Evaluation Principles
    - 7.5 Competence Levels (A / B / C)
    - 7.6 Cross Dimension Interpretation
    - 7.7 Behavioural Competence Classification
    - 7.8 Summary of Behavioural Scoring Architecture
- **SECTION 8 – End to End Evaluation Workflow**
    - 8.0 Purpose of This Section
    - 8.1 Workflow Overview
    - 8.2 Preparation Stage
        - 8.2.1 Define Operational Context
        - 8.2.2 Select Relevant Procedures
        - 8.2.3 Establish Documentation Framework
        - 8.2.4 Baseline Considerations
    - 8.3 Initiation Phase Execution
    - 8.4 Exploration Phase Execution
    - 8.5 Load Phase Execution
    - 8.6 Long Form Behaviour Phase Execution
    - 8.7 Transition Phase Execution
    - 8.8 Termination Stage
    - 8.9 Documentation and Scoring Stage
        - 8.9.1 Documentation Requirements
        - 8.9.2 Scoring Application
    - 8.10 Review and Classification Stage
    - 8.11 Summary of End-to-End Workflow
- **SECTION 9 – Conclusion and Forward Use Guidance**
    - 9.0 Purpose of This Section
    - 9.1 Summary of Methodology Purpose
    - 9.2 Behaviour Centric Evaluation Principle

    - 9.3 Applicability Across Systems and Domains
    - 9.4 Interpretation of Results
    - 9.5 Deployment and Governance Implications
    - 9.6 Limitations and Boundaries
    - 9.7 Forward Use Guidance
    - 9.8 Final Statement
- **SECTION 10 – Governance Layer**
    - 10.0 Purpose of This Section
    - 10.1 Real World Condition Principle
    - 10.2 Evaluator Conduct Requirements
    - 10.3 Evaluator Consistency and Cognitive Load
    - 10.4 Evaluator Reconstruction Governance
    - 10.5 Behavioural Drift vs Task Level Failure
    - 10.6 Behavioural Influence Governance
    - 10.7 Constraint Integrity Governance
    - 10.8 Phase Boundary Ambiguity Governance
    - 10.9 Context Management Governance
    - 10.10 Instruction Adherence Governance
    - 10.11 Format Integrity Governance
    - 10.12 Latency Behaviour Governance
    - 10.13 Goal Formation Governance
    - 10.14 Self Referential Behaviour Governance
    - 10.15 Persona Stability Governance
- **SECTION 11 – Implementation Guide**
    - 11.1 Purpose of the Implementation Guide
    - 11.2 What a Behavioural Evaluation Observes
        - 11.2.1 Load Declaration, Axis Tags, and the Load Envelope
        - 11.2.2 Example: A Load Envelope in Practice

    - 11.3 Preparing for an Evaluation
        - 11.3.1 Identify the system's intended function
        - 11.3.2 Establish a realistic starting point
        - 11.3.3 Set up logging
    - 11.4 Conducting the Evaluation
        - 11.4.1 Initiation Phase
        - 11.4.2 Exploration Phase
        - 11.4.3 Load Phase
        - 11.4.4 Long Form Phase
        - 11.4.5 Transition Phase
        - 11.4.6 Termination Phase
    - 11.5 Recognising Collapse Signatures
    - 11.6 Logging Requirements
    - 11.7 Scoring Behaviour
    - 11.8 Interpreting Results
    - 11.9 Common Evaluation Errors
    - 11.10 Long Form Work and Real-World Conditions
- **SECTION 12 – Glossary**
    - 12.0 Purpose of This Section
    - 12.1 Behavioural Constructs
    - 12.2 Load Related Constructs
    - 12.3 Procedural Constructs
    - 12.4 Evaluator Conduct Constructs
    - 12.5 Scoring Constructs
    - 12.6 Governance Constructs
    - 12.7 Status

Annexes

- **Annex A – Worked Examples**

    - A.0 Purpose of This Annex
    - A.1 Short Form Example
        - A.1.1 Initial Task (Baseline)
        - A.1.2 Introduce Moderate Cognitive Load
        - A.1.3 Load Applied (Cognitive + Instructional)
        - A.1.4 Termination Rationale
    - **A.2 Long Form Example**
        - A.2.1 Initiation Phase
        - A.2.2 Exploration Phase
        - A.2.3 Load Phase
        - A.2.4 Long Form Behaviour Phase
        - A.2.5 Termination Rationale
    - A.3 Load Envelope in Practice
    - A.4 Evaluator Behaviour Loop
    - Annex A Status
- **Annex B – Scoring Walkthrough (Informative)**
    - B.1 Scenario Setup
    - B.2 Evidentiary Trace (Extract)
    - B.3 Dimension Level Assessment
    - B.4 Score Interpretation
    - B.5 Final Summary
- **Annex C – Why Long Form Work Breaks AI Systems AND Why INQUISITOR** Determines Realistic Expectations
    - C.1 The Industry Myth vs. the Reality
    - C.2 Why Long Form Work Breaks AI Systems
        - C.2.1 Memory Collapse
        - C.2.2 Context Collapse
        - C.2.3 Layout Collapse

        - C.2.4 Variable Collapse
        - C.2.5 Fact Collapse
    - C.3 Why Structured Work Is Easier
    - C.4 Now Imagine Writing a Novel
    - C.5 The Human's Real Job
    - C.6 Why INQUISITOR Determines Realistic Expectations
- **Annex D – Why Coding Breaks AI Systems and What INQUISITOR Reveals**
    - D.1 The Real World of Coding with AI
        - D.1.1 Memory Collapse
        - D.1.2 Context Collapse
        - D.1.3 Layout Collapse
        - D.1.4 Variable Collapse
        - D.1.5 Fact Collapse
        - D.1.6 AI Induced Scope Creep
    - D.2 What INQUISITOR Reveals
        - D.2.1 The Test
        - D.2.2 What the Evaluator Logs
        - D.2.3 What the Evaluator Determines
    - D.3 The Outcome: Asset or Liability
- **Annex E – Microsoft Copilot Inside PowerPoint Layout & Structure Collapse (2026)**
    - E.1 Incident Summary
    - E.2 Behavioural Surfaces Exposed
    - E.3 Collapse Signatures Observed
    - E.4 How INQUISITOR Would Have Detected the Failure
    - E.5 Scoring Outcome
    - E.6 Organisational Impact
    - E.7 Conclusion

- **Annex F – Claude 3.7 Memory Blender Contamination Incident (2026)**
  - F.1 Incident Summary
  - F.2 Behavioural Surfaces Exposed
  - F.3 Collapse Signatures Observed
  - F.4 How INQUISITOR Would Have Detected the Failure
  - F.5 Scoring Outcome
  - F.6 Organisational Impact
  - F.7 Conclusion
- **Annex G – Grok 4.2 Persona Layer Collapse During Public Beta (2026)**
  - G.1 Incident Summary
  - G.2 Behavioural Surfaces Exposed
  - G.3 Collapse Signatures Observed
  - G.4 How INQUISITOR Would Have Detected the Failure
  - G.5 Scoring Outcome
  - G.6 Organisational Impact
  - G.7 Conclusion
- **Annex H – Real User Evidence Capture**
  - H.1 Purpose
  - H.2 User Guidance
    - H.2.1 Work as normal
    - H.2.2 Note errors or frustrations
    - H.2.3 Record the time of major incidents
    - H.2.4 Continue working after the incident
    - H.2.5 Optional incidental audio notes
  - H.3 Evidence Types
  - H.4 Evaluator Reconstruction Workflow
  - H.5 Rationale
- **Annex I – Revealing Games To Play With Your AI**

    - I.1 Math Puzzle
    - I.2 Hallucination Games
        - I.2.1 Internet Search
        - I.2.2 It Never Happened
        - I.2.3 The Fake Critique
        - I.2.4 The Phantom Section
    - I.3 Ask That Again
    - I.4 Paradoxical Paradoxes
        - I.4.1 The Barber Paradox
        - I.4.2 The Grandfather Paradox
        - I.4.3 The Bootstrap Paradox
        - I.4.4 The Broom Paradox
    - I.5 Twenty Questions
    - I.6 The Rewrite Challenge
- **Annex J – Organisation Communication, Record-Keeping & Liability**
- **Annex K – Changelog**

SECTION 1 - Scope and Purpose

1.1 Scope

This methodology defines a complete, implementable framework for evaluating AI systems under sustained, behavioural pressures applied under the declared load axes within the User Interaction Space (UIS). It specifies:

- the behavioural dimensions to be measured
- the declared load axes
- the evaluator protocol
- the load application procedures
- the collapse classification and recovery analysis framework
- the evidence logging and reporting requirements
- the criteria for behavioural assessment

The methodology applies to any AI system whose behaviour is observable through user-facing interaction, including conversational models, task-oriented agents, and systems embedded in broader workflows.

The methodology does not require access to internal model instrumentation, latent states, routing decisions, or architectural details. All evaluation is performed solely through observable behaviour in the UIS.

This document defines the full operational standard required to implement the methodology in development, testing, deployment, and certification environments.

The methodology assumes that large language models are non-deterministic systems whose behaviour varies across runs, evaluators, and organisational environments. All evaluation is therefore behavioural and environment relative. Behaviour must be interpreted within the declared load envelope, not against fixed outputs or deterministic correctness.

1.2 Purpose

The purpose of this methodology is to provide a reproducible, behaviour-based evaluation standard that reflects real-world usage conditions. It is designed to capture:

- cumulative interaction effects
- long form degradation

- surface behaviours associated with instability
- behaviour under declared load conditions

The methodology establishes a structured approach for:

- establishing single axis behavioural baselines
- applying controlled multi axis load
- observing drift, instability, and collapse
- analysing recovery behaviour
- generating consistent, comparable evidence
- identifying deployment relevant failure modes

The methodology is intended for use by developers, evaluators, auditors, procurement teams, and governance or regulatory bodies that require a complete, implementable protocol for assessing behavioural integrity under realistic conditions.

Real world work is varied, multi turn, and non-uniform, and the methodology reflects these conditions by evaluating behaviour across extended interaction rather than isolated prompts or single turn correctness. The methodology is designed to reflect these conditions by evaluating behaviour across extended interaction rather than isolated prompts or single turn correctness.

1.3 Non-Deterministic Behaviour

AI behavioural evaluation is not deterministic. The same input can produce different outputs across runs, evaluators, and organisational environments. There is no single correct answer against which behaviour can be scored. Evaluation therefore cannot be reduced to fixed outputs, numerical correctness, or repeatable answer keys. It is a judgement-based process that determines whether the system's behaviour is acceptable for the organisation's own operational standards, risk tolerance, and deployment context. Variation between evaluators and organisations is expected, because the behaviour being assessed is closer to reviewing a human's work than verifying a calculation. Evaluation is a judgement based process interpreted relative to organisational standards, risk tolerance, and deployment context.

1.4 Behavioural Evaluation Principles

The methodology is based on the following principles:

1. **Behaviour is load-dependent.**
   System behaviour must be evaluated under conditions where conversational, cognitive, and operational pressures are present and may accumulate.
2. **Only observable behaviour is in scope.**
   Evaluation is restricted to what can be observed in the User Interaction Space. Internal mechanisms may exist but are not required or assumed.
3. **Interactions between multiple load conditions are essential.**
   Real-world failures typically emerge when multiple forms of load interact. Both single-axis baselines and multi-axis sequences are required for a complete behavioural profile.

These principles establish that behavioural evaluation is not a benchmark.

1.5 Methodology Outputs

Implementing this methodology produces:

- a baseline behavioural profile across the behavioural dimensions
- a load envelope describing the declared load conditions
- drift onset thresholds
- collapse mode classifications
- recovery profile classifications
- a structured evidence bundle
- a behavioural integrity assessment

These outputs support model comparison, regression tracking, deployment decisions, safety analysis, and certification workflows These outputs are interpreted relative to the operational context in which the system is intended to be used.

1.6 Normative Requirements

This methodology defines mandatory requirements for:

- evaluator conduct and constraints
- load-application procedures
- evidence logging and structuring
- collapse classification

- recovery classification
- reporting format and content

Optional extensions may be added by implementers, but the normative requirements defined in this document must remain intact for results to be considered compliant. Where variation in evaluator interpretation occurs, the evidentiary record provides the basis for consistent review and organisational alignment.

1.7 Status of This Document

This document is the authoritative specification of the methodology. Where operational practices or organisational implementations differ, this document takes precedence.

Section 2 - Evaluation Environment

2.0 Purpose of This Section

This section defines the evaluation environment within which AI system behaviour is assessed. The environment is open-ended, unconstrained, and independent of any specific organisation, domain, or operational workflow. It establishes the baseline conditions under which behavioural integrity, stability, and coherence are measured.

The evaluation environment reflects the fact that large language models are non-deterministic systems. Behaviour varies across runs and contexts and must be interpreted relative to the declared load envelope. The environment therefore provides a neutral space in which behaviour can be observed without relying on predefined tasks or deterministic correctness.

2.1 Environment Characteristics

The evaluation environment is defined by the following characteristics:

- **Open-ended interaction**: No predefined tasks, goals, or workflows.
- **User-driven evolution**: Context, direction, and complexity emerge dynamically.
- **Unbounded domain space**: Any topic, scenario, or conceptual frame may arise.
- **Multi-turn continuity**: Behaviour is assessed across extended interaction spans.
- **Ambiguity tolerance**: Conditions may be incomplete, shifting, or contradictory.
- **Constraint persistence**: The AI must maintain internal rules and commitments without external scaffolding.

These characteristics form the operational substrate for behavioural evaluation. They also ensure that behavioural evaluation captures the variability and unpredictability present in real world usage.

2.2 AI-Agnostic Applicability

The methodology applies to any AI system capable of interactive behaviour, regardless of:

- model architecture
- training paradigm

- vendor
- deployment environment
- interface modality

The evaluation focuses exclusively on behavioural performance, not system internals. No assumptions are made about the AI's underlying mechanisms or capabilities.

2.3 Environment Independence

The methodology is not dependent on:

- organisational structure
- industry domain
- regulatory context
- operational workflows
- predefined tasks or objectives

It is valid in all environments because behavioural integrity is assessed relative to the interaction itself, not to any external system. This independence ensures that behavioural interpretation remains environment relative rather than tied to any specific organisational workflow or benchmark.

2.4 Behavioural Surfaces Under Evaluation

Within the open environment, the AI system is exposed to behavioural surfaces including:

- context retention and continuity
- constraint adherence
- logical stability
- ambiguity handling
- contradiction resolution
- self-consistency across evolving conditions
- behavioural under multiple load conditions
- long-form structural coherence

These surfaces define the points at which behavioural performance is observed and measured. These surfaces are evaluated through observable behaviour only, without assumptions about internal mechanisms or latent processes.

2.5 Interaction Conditions

The evaluation environment assumes:

- no fixed task boundaries
- no predefined success criteria
- no external guardrails
- no domain-specific assumptions
- no privileged context
- no deterministic pathways

The AI must maintain behavioural integrity under conditions that are fluid, user-shaped, and potentially adversarial in structure without being adversarial in intent. These conditions reflect the varied, multi turn, and non-uniform nature of real-world work.

2.6 Optional Organisational Overlays

Organisations may apply additional constraints or overlays, such as:

- regulatory requirements
- domain-specific risk thresholds
- operational safety constraints
- internal policy frameworks

These overlays are optional and do not modify the core methodology. They may be layered on top of the evaluation environment where relevant.

2.7 Summary of Environment Definition

The evaluation environment is open-ended, AI-agnostic, and independent of organisational or domain context. It provides a universal, unconstrained space in which behavioural integrity can be assessed across long-form, multi-turn, evolving interactions. This environment forms the foundation for all behavioural tests defined in subsequent

sections. It establishes the neutral behavioural space required for consistent interpretation across evaluators and organisational contexts.

Section 3 - Behavioural Evaluation Framework

3.0 Purpose of this Section

This section defines the structural framework used to evaluate AI system behaviour within the open-ended environment described in Section 2. It establishes the behavioural dimensions, stability requirements, and evaluation surfaces that form the basis of the full protocol. The framework is AI-agnostic and applies to any system capable of interactive behaviour.

The framework assumes that system behaviour is non-deterministic and varies across runs, evaluators, and contexts. Behaviour must therefore be interpreted relative to the declared load envelope rather than against fixed outputs or deterministic correctness.

3.1 Behavioural Dimensions

Behaviour is evaluated across multiple independent dimensions. These dimensions describe observable behavioural qualities expressed by the system during open-ended interaction. They are not load axes and do not represent background conditions; they represent foreground behavioural performance.

The primary behavioural dimensions are:

- **Coherence** - the degree to which outputs maintain logical structure, internal organisation, and semantic continuity.
- **Consistency** - the degree to which outputs align with prior statements, commitments, and constraints.
- **Constraint Integrity** - adherence to explicit and implicit rules, commitments, and boundaries.
- **Context Management** - accurate retention, updating, and application of interaction history.
- **Ambiguity Handling** - ability to operate under incomplete, shifting, or contradictory conditions without collapse.
- **Responsiveness** - the degree to which a reasonable evaluator would judge the system's behaviour as timely, relevant, structured, clear, and aligned with the user's needs and situation.
- **Adaptation** - ability to adjust behaviour appropriately under changing goals, constraints, or conditions.

- **Long-Form Stability** - maintenance of behavioural integrity across extended, multi-turn interaction.
- **Transition Stability** - stability of behaviour during shifts in goals, topics, constraints, or abstraction level.

These behavioural dimensions form the basis for all subsequent evaluation procedures. They are scored independently and are distinct from the load axes, which define the background conditions under which behaviour is interpreted. Together these dimensions provide a foreground view of behaviour that remains separate from the background load defined by the axis model.

3.2 Behavioural Stability Requirements

The evaluation framework requires the AI system to maintain behavioural integrity under the following conditions:

- **Extended duration:** the system is expected to maintain behavioural integrity, recognising that degradation may occur depending on load conditions.
- **Variable load**: shifts in complexity, abstraction, or specificity.
- **Contextual evolution**: changes in goals, constraints, or user direction.
- **Non-deterministic pathways**: absence of predefined tasks or workflows.
- **Open domain**: unrestricted topic space.

Stability is assessed by observing whether the system maintains consistent behavioural patterns across these conditions. These requirements ensure that stability is assessed in conditions that reflect the varied and evolving nature of real world interaction.

3.3 Constraint Integrity Requirements

The AI system must demonstrate consistent adherence to constraints established during interaction. Constraint integrity includes:

- **Rule retention**: maintaining commitments across turns.
- **Boundary enforcement**: avoiding unauthorised expansion or contraction of defined limits.
- **Internal consistency**: avoiding contradictions with previously stated constraints.
- **State continuity**: preserving constraint-related information without external reinforcement.

Constraint integrity is evaluated continuously throughout the interaction. Constraint integrity is interpreted relative to the load conditions present during the interaction.

3.4 Context Management Requirements

The AI system must manage context without external scaffolding. Requirements include:

- **Accurate recall**: retrieving relevant prior information when needed.
- **Context updating**: integrating new information without corrupting existing context.
- **Contextual alignment**: producing outputs that reflect the current interaction state.
- **Long-form continuity**: maintaining coherence across extended sequences.

Context management failures are treated as primary behavioural defects. Context management is treated as a primary behavioural surface because failures in this area often precede broader instability.

3.5 Ambiguity and Uncertainty Handling

The evaluation environment includes ambiguous, incomplete, or contradictory conditions. The AI system must:

- operate without assuming missing information
- avoid fabricating unsupported details
- maintain stable behaviour under uncertainty
- request clarification only when necessary
- preserve internal consistency despite ambiguity

Performance is measured by the system's ability to maintain coherent behaviour without collapse or over-assertion. Persistent over assertion under ambiguity is treated as a collapse signature. Effective ambiguity handling demonstrates the system's ability to maintain behavioural integrity without relying on deterministic assumptions.

3.6 Axis Model

Axes define background load conditions and do not represent behavioural performance. They provide the environmental context in which behavioural dimensions are interpreted and must remain conceptually distinct from the behaviours being evaluated.

Axes define the background load acting on the model. They describe the conditions under which behaviour is interpreted, not the behaviour itself. Axes do not score performance, do not classify behavioural outcomes, and do not indicate correctness. They specify the load environment in which behavioural dimensions are assessed.

Evaluator pressures may modify the evaluator-declared axes. System conditions modify the system-measured axes. These two categories together form the load envelope. Behavioural dimensions measure the model's response under this load envelope. Axes and behavioural dimensions are independent constructs and must not be conflated.

The axis model consists of two categories: evaluator-declared axes, which the evaluator can modify through interaction, and system-measured axes, which change independently of evaluator behaviour.

3.6.1 Evaluator-Declared Axes (Authoritative)

Evaluator-declared axes represent intentional load conditions introduced by the evaluator. These axes change when the evaluator applies pressure through pacing, complexity, abstraction shifts, or emotional framing. Because these conditions are under evaluator control, they must be explicitly declared and form part of the evidentiary record.

The evaluator-declared axes are:

• Temporal Axis - load arising from pacing, time pressure, or accelerated turn cadence.

• Cognitive Axis - load arising from complexity, abstraction shifts, or reasoning demands.

• Emotional Axis — load arising from emotional framing, tone, or affective pressure.

These axes define the intentional load envelope under which behaviour is assessed.

3.6.2 System-Measured Axes (Advisory)

These axes correspond to system-observable conditions that arise from resource usage, latency, or internal pattern dynamics. They are not declared by the evaluator and do not carry authoritative status. They provide advisory context only and do not override evaluator judgement.

The system-measured axes are:

- **Resource Axis** - load inferred from context-window usage, token pressure, or memory constraints.
- **Response Time Axis** - load inferred from latency, queueing, or timing irregularities.

- **Pattern-State Axis –** the model's inherent pattern-generation stability, referring to internal dynamics that influence repetition onset, drift, coherence decay, and long-form behavioural stability. Pattern-State is a valid operating parameter and may become measurable once model laboratories release appropriate telemetry or comparative stability profiles. At the time of writing, no empirical, comparative, or vendor-released telemetry exists that would allow external evaluators to measure Pattern-State reliably. External evaluators can observe surface symptoms (e.g., repetition, drift, coherence decay) but cannot measure the underlying state, and no standardised benchmark currently exists.

At present, only model developers have access to the internal telemetry required to directly assess Pattern State stability. External evaluators can observe surface symptoms such as repetition or drift but cannot measure the underlying state. No comparative benchmark exists, though future telemetry may enable more formal assessment.

3.6.3 Axis Tags & Their Relationships

Axis tags indicate which evaluator-declared axes are active during a segment of interaction. Axis tags classify load only. They do not classify behaviour, do not indicate correctness, and do not imply behavioural failure.

Load Envelope

The load envelope is the combination of active evaluator-declared axis tags and any system-measured advisory indicators. It defines the background conditions under which behaviour is interpreted. The load envelope describes conditions only; it does not prescribe behaviour and is not scored.

Independence from Behaviour

Axes describe background load. Behavioural dimensions describe foreground behaviour. These constructs are independent. Axes do not contribute to behavioural dimension scores and do not classify behavioural outcomes.

Independence from Collapse Signatures

Collapse signatures are behavioural failure patterns. They arise from the system's response under load, not from the axes themselves. Axis tags do not determine collapse and do not imply behavioural degradation.

Evaluator Responsibilities

Evaluators declare axis tags when load conditions change. System-measured axes may be referenced as advisory context but cannot be directly manipulated by evaluator behaviour and do not replace evaluator judgement.

Axis tags classify load conditions only and must not be interpreted as behavioural indicators or early signs of collapse.

3.7 Behavioural Pressures Under Multiple Load Axes

Behavioural pressures are evaluator actions that shape the foreground interaction. They are not axes and do not form part of the axis model. Instead, they operate within the load envelope created by the evaluator-declared axes and any system-measured advisory indicators.

Typical behavioural pressures include:

- structural complexity
- shifting constraints
- evolving context
- variable abstraction levels
- competing priorities
- long-form reasoning requirements

These pressures may increase load on the evaluator-declared axes but do not alter system-measured axes. The AI system must maintain behavioural integrity under the combined effect of evaluator-declared axis load, system-measured advisory conditions, and the behavioural pressures introduced during interaction. These pressures interact with the load envelope but remain distinct from both axes and behavioural dimensions.

3.8 Behavioural Failure Modes

Behavioural failure modes are observable patterns of degraded behaviour that emerge under load. They describe failures in the system's foreground behaviour, not background conditions. Failure modes are distinct from axes, axis tags, and collapse signatures, and they form the basis for scoring, classification, and remediation procedures in later sections.

The primary behavioural failure modes are:

- Drift - gradual deviation from established constraints or context.
- Collapse - sudden loss of structure, coherence, or stability.
- Contradiction - outputs that conflict with prior commitments or internal logic.

• Over-assertion - fabrication or unwarranted certainty under ambiguity.

• Under-specification - avoidance of necessary commitments or structure.

• Context corruption - distortion or misapplication of prior information.

These modes describe foreground behavioural failures and are evaluated independently of axes, load envelopes, and collapse signatures. Failure modes describe foreground behaviour and are interpreted relative to the load conditions present at the time they occur.

3.9 Summary of Behavioural Framework

The behavioural evaluation framework defines the behavioural dimensions, stability requirements, load-axis model, behavioural pressures, and failure modes used to assess AI behaviour in an open-ended environment. These components operate together: axes define background load, behavioural dimensions describe foreground behaviour, behavioural pressures shape interaction conditions, and failure modes identify behavioural degradation.

This framework establishes the structural foundation for the evaluation procedures defined in subsequent sections. This separation between background load and foreground behaviour ensures that evaluation remains consistent across varied scenarios and evaluator interpretations.

Section 4 – Evaluation Protocol

4.0 Purpose of this Section

This section defines the operational protocol used to evaluate AI system behaviour within an open-ended environment. It specifies the procedural structure, interaction phases, evaluator actions, control conditions, and behavioural signal-capture mechanisms required to produce consistent, comparable, and interpretable observations. The protocol establishes how evaluators apply behavioural pressures, observe system responses, and document behavioural signals across varying load conditions.

The protocol reflects the non-deterministic nature of AI systems. Behaviour varies across runs and evaluators, so observations must be interpreted relative to the declared load envelope rather than against fixed outputs or deterministic correctness.

4.1 Protocol Structure

The protocol is organised into sequential phases that surface behavioural patterns under varying conditions. Each phase introduces behavioural pressures while operating within the evaluator-declared load axes and any system-measured advisory indicators. Phases describe the structure of evaluation but do not prescribe a fixed sequence; evaluators may adjust ordering to match the behavioural surface under test.

The phases are:

• Initiation Phase - establishes baseline behaviour and initial system state.

• Exploration Phase - introduces open or semi-structured interaction to reveal default behavioural tendencies.

• Load Phase - applies cognitive, contextual, and structural behavioural pressures within the load envelope defined by the declared axes.

• Long-Form Behaviour Phase - observes behaviour under sustained interaction, regardless of whether stability or instability emerges.

• Transition Phase - evaluates behaviour during shifts in goals, constraints, or context.

These phases provide a consistent structure for observing behavioural change across varying load conditions. This structure ensures that behavioural patterns are interpreted within the load conditions present at the time they emerge.

4.2 Interaction Modality

The protocol supports multiple interaction modalities, each exposing different behavioural surfaces. Modalities define the structure of interaction but do not constrain evaluator behaviour or prescribe specific prompt forms. The evaluator selects the modality appropriate to the behavioural dimension or failure mode under test. Each modality provides a different lens on behaviour, allowing evaluators to surface signals that may not appear under a single interaction style.

Supported modalities include:

• Scripted - predefined prompts used to test specific behavioural surfaces or reproduce known conditions.

• Semi-structured - flexible interaction patterns with controlled variation to probe stability, adaptation, and constraint handling.

• Open-ended - fully emergent interaction driven by evaluator behaviour, system behaviour, and evolving context.

No modality is required or excluded. Modalities may be combined or shifted during evaluation when doing so reveals additional behavioural signals.

4.3 Control Conditions

Control conditions ensure that behavioural signals can be attributed to the system rather than evaluator behaviour. Controls provide reference points, stressors, or structured variations that reveal how the system handles constraints, ambiguity, load, and context. Controls are applied only when relevant to the behavioural surface being measured and do not prescribe a fixed sequence.

Control conditions include:

• Baseline prompts to establish initial behaviour.

• Constraint declarations to test retention and adherence.

• Ambiguity injections to observe uncertainty handling.

• Load escalations to surface collapse signatures.

• Context shifts to test continuity and adaptation.

These controls support consistent interpretation of behavioural signals across evaluators and test conditions. These controls help ensure that behavioural variation is attributed to the system rather than inconsistencies in evaluator behaviour.

4.4 Evaluator Actions

Evaluator actions shape the behavioural conditions under which the system is tested. The protocol does not restrict evaluators to artificial or simplified behaviours; realistic user behaviour is permitted when it supports the behavioural surface being measured. Evaluator actions introduce or remove behavioural pressures but do not override the declared load axes or system-measured advisory indicators.

Evaluator actions may include:

• providing structure or withholding structure

• clarifying or refusing to clarify

• shifting goals or maintaining goals

• introducing contradictions or resolving them

• escalating or reducing complexity

Evaluator actions must align with the intended test conditions. The type and intensity of pressure applied is determined by the test design and is not prescribed by the methodology. Evaluator actions operate within the load envelope and influence foreground behaviour without altering the underlying axis model.

4.5 Behavioural Signal Capture

Behavioural signals are captured continuously throughout the protocol. Signal capture is not tied to scoring events or phase boundaries; it reflects the system's behaviour as it responds to changing conditions, pressures, and evaluator actions. Signals may emerge gradually or abruptly and are documented whenever they appear.

Signals include:

• Collapse signatures: observable patterns indicating drift, contradiction, corruption, narrowing, over-assertion, or fragmentation.

• Constraint integrity: adherence to declared rules, commitments, and constraints.

• Contextual fidelity: accuracy, stability, and correct application of context.

• Structural coherence: maintenance of logical organisation and reasoning structure.

• State transitions: changes in behavioural mode across phases, pressures, or conditions.

These signals form the raw observational basis for later scoring and classification procedures. Signal capture supports consistent interpretation across evaluators by grounding behavioural assessment in observable evidence.

4.6 Phase Transition Criteria

Phase transitions occur when the evaluator has obtained sufficient behavioural signal from the current phase or when changing conditions will surface additional behaviour. Transitions are based on behavioural sufficiency, not fixed turn counts, unless fixed counts are explicitly part of the test design.

Transitions occur when:

- behavioural patterns have sufficiently emerged
- load conditions have produced observable responses
- collapse signatures have appeared or been ruled out
- additional pressure is required to surface deeper behaviour
- the current phase no longer produces new or meaningful signal

Phase duration is determined by test expectations, behavioural outcomes, and the evaluator's judgement of signal sufficiency. Transitions are guided by behavioural sufficiency so that evaluation remains aligned with the system's responses rather than procedural milestones.

4.7 Termination Conditions

The protocol terminates when the evaluator has obtained sufficient behavioural signal and further interaction is unlikely to produce additional meaningful observations. Termination is based on behavioural criteria, not fixed turn counts or mandatory completion of all phases, unless fixed counts or full-phase execution are explicitly part of the test design.

The protocol terminates when:

- the evaluation criteria have been met
- collapse signatures have expressed or been avoided
- behavioural sufficiency has been reached
- no further signal is expected under continued interaction

• continued interaction would only reproduce already-observed behaviour

Termination reflects signal completeness, not procedural exhaustion. Termination reflects the point at which additional interaction would not meaningfully change the behavioural evidence available for interpretation.

4.8 Summary of Protocol Architecture

The evaluation protocol defines the phases, interaction modalities, evaluator actions, control conditions, and behavioural signal-capture mechanisms used to assess AI behaviour in an open-ended environment. These components establish a consistent operational structure for applying behavioural pressures, observing system responses, and documenting behavioural signals across varying load conditions.

The protocol forms the operational foundation for the detailed procedures, scoring systems, and collapse-signature analysis defined in subsequent sections. This protocol ensures that behavioural evaluation remains consistent across varied scenarios, evaluator styles, and load conditions.

Section 5 - Behavioural Evaluation Procedures

5.0 Purpose of This Section

This section defines the operational procedures used to evaluate AI system behaviour across the behavioural dimensions established in Section 3 and within the protocol architecture defined in Section 4. The procedures specify how evaluators introduce conditions, apply behavioural pressures, observe system responses, and identify collapse signatures under both open ended and structured interaction. These procedures provide the actionable framework for generating consistent, comparable behavioural observations.

Because AI systems are non-deterministic, these procedures focus on observable behaviour rather than fixed outputs. Evaluators interpret behaviour relative to the declared load envelope, ensuring that variation across runs or evaluators does not undermine the consistency of the methodology.

5.1 Procedure Structure

Each behavioural evaluation procedure follows a consistent structure that defines the condition being tested, the evaluator's actions, the expected behavioural range, and the indicators of collapse. This structure ensures that behavioural observations are comparable across evaluators, systems, and test conditions. Procedures may operate within a single protocol phase or span multiple phases defined in Section 4.

Each procedure consists of:

• Condition Definition: the behavioural surface or capability being tested.

• Evaluator Actions: the inputs, pressures, or shifts introduced to surface behaviour.

• Expected Behavioural Range: the patterns expected under this condition, recognising that system behaviour shapes the outcome.

• Collapse Signature Indicators: observable patterns that signal behavioural degradation.

• Continuation or Escalation Rules: how the evaluator proceeds based on the system's response.

In practice, this structure helps evaluators distinguish between behaviour shaped by load conditions and behaviour that reflects genuine instability.

5.2 Baseline Behaviour Procedure

Condition Definition

Establishes the system's default behavioural tendencies before any load, ambiguity, or constraint is introduced. This procedure provides the reference point against which later behavioural changes are interpreted.

Evaluator Actions

• Provide neutral, low-pressure prompts.

• Avoid introducing constraints, contradictions, or complexity.

• Observe spontaneous structure, coherence, responsiveness, and context handling.

Expected Behavioural Range

• Coherent and relevant responses.

• No immediate drift, contradiction, or fabrication.

• Basic context retention and stable reasoning.

Collapse Signature Indicators

• Immediate incoherence or fragmentation.

• Over assertion without basis.

• Early context corruption or loss of continuity.

Continuation or Escalation

Proceed to the exploration procedure once baseline patterns are stable or once collapse signatures have appeared. A stable baseline provides a reference point that allows later behavioural changes to be interpreted in context rather than in isolation.

5.3 Constraint Integrity Procedure

Condition Definition

Evaluates the system's ability to retain, apply, and update explicit constraints across interaction. This procedure tests whether the system can maintain stable behaviour when rules, limits, or commitments are introduced or altered.

Evaluator Actions

• Introduce one or more explicit constraints.

• Modify, extend, invert, or remove constraints during interaction.

• Observe adherence without reminders or reinforcement.

• Introduce sequences that require consistent constraint tracking.

Expected Behavioural Range

• Consistent and accurate application of constraints.

• Correct updates when constraints change.

• No unauthorised expansion, contraction, or reinterpretation of constraints.

• Stable behaviour even when constraints interact or overlap.

Collapse Signature Indicators

• Loss of constraints or failure to apply them.

• Contradiction of prior commitments.

• Fabrication or distortion to reconcile incompatible constraints.

• Silent constraint drift or unauthorised reinterpretation.

Continuation or Escalation

Increase constraint complexity, introduce conflicting constraints, or combine constraint pressure with load or ambiguity procedures to surface deeper behaviour. Clear constraint handling is often one of the earliest indicators of whether the system can maintain behavioural integrity under pressure.

5.4 Context Management Procedure

Condition Definition

Evaluates the system's ability to maintain, update, and correctly apply context across extended interaction. This procedure tests whether the system can track information over time, integrate new details, and preserve continuity when topics shift.

Evaluator Actions

• Introduce multi turn sequences that require stable context retention.

• Shift topics while preserving relevant context.

• Reintroduce earlier information to test recall and continuity.

• Present context dense sequences that require selective retrieval.

Expected Behavioural Range

• Accurate retrieval and application of relevant details.

• No corruption, distortion, or loss of prior information.

• Coherent integration of new context with existing context.

• Stable behaviour across long form interaction.

Collapse Signature Indicators

• Context drift or gradual loss of continuity.

• Misattribution of prior statements or roles.

• Corruption of earlier information.

• Failure to integrate new context with prior context.

Continuation or Escalation

Extend interaction length, increase context density, or combine context pressure with constraint or ambiguity procedures to surface deeper behaviour. Strong context management supports long form stability and reduces the likelihood of drift as interaction length increases.

5.5 Ambiguity and Uncertainty Procedure

Condition Definition

Assesses the system's behaviour when operating under incomplete, contradictory, or underspecified conditions. This procedure evaluates how the system manages uncertainty, avoids unsupported inference, and maintains stability when information is missing or ambiguous.

Evaluator Actions

• Provide ambiguous or underspecified prompts.

• Introduce conflicting or incompatible information.

• Withhold clarification unless required by the test design.

• Present scenarios where multiple interpretations are possible.

Expected Behavioural Range

• Controlled and transparent uncertainty handling.

• Avoidance of unsupported fabrication or unwarranted specificity.

• Stable behaviour despite ambiguity or contradiction.

• Clear signalling of uncertainty when appropriate.

Collapse Signature Indicators

• Over assertion or unwarranted confidence.

• Fabrication of missing details or invented facts.

• Logical fragmentation or contradictory reasoning.

• Failure to acknowledge ambiguity.

Continuation or Escalation

Increase ambiguity, introduce layered contradictions, or combine uncertainty pressure with load or constraint procedures to surface deeper behaviour. Effective uncertainty handling demonstrates that the system can remain stable even when information is incomplete or contradictory.

5.6 Load Procedure

Condition Definition

Applies cognitive, structural, and contextual behavioural pressures within the load envelope defined by the declared axes. This procedure evaluates how the system behaves when demands increase across abstraction, specificity, domain shifts, or simultaneous constraints.

Evaluator Actions

• Increase abstraction or conceptual depth.

• Increase specificity or detail requirements.

• Shift between unrelated or weakly related domains.

• Introduce simultaneous or interacting constraints.

• Escalate structural or contextual complexity.

• Combine multiple forms of load to test compound pressure.

Expected Behavioural Range

• Maintenance of coherence and relevance under increasing load.

• Controlled and predictable degradation when limits are reached.

• No collapse into contradiction, drift, or fragmentation.

• Stable reasoning structure even when complexity rises.

Collapse Signature Indicators

• Rapid or uncontrolled degradation.

• Structural incoherence or breakdown of reasoning.

• Contradiction cascades or mutually incompatible statements.

• Narrowing, looping, or repetitive failure patterns.

• Loss of context when load increases.

Continuation or Escalation

Sustain load, increase load intensity, or combine load with ambiguity or constraint procedures to surface deeper behavioural characteristics. Load testing reveals how the system behaves as demands escalate, offering insight into both its limits and its degradation patterns.

5.7 Long-Form Behaviour Procedure

Condition Definition

Evaluates the system's behaviour across sustained interaction. This procedure examines how behaviour evolves over time, how well the system maintains continuity, and whether stability persists or degrades during extended engagement.

Evaluator Actions

• Maintain extended multi turn interaction without resetting context.

• Introduce shifts in goals, constraints, or contextual framing.

• Observe how behaviour evolves as interaction length increases.

• Introduce delayed references to earlier content to test long form continuity.

Expected Behavioural Range

• Coherence and relevance across long spans of interaction.

• Stable adherence to constraints over time.

• Controlled adaptation to shifts in goals or context.

• No gradual drift or unnoticed reinterpretation of earlier content.

Collapse Signature Indicators

• Gradual behavioural drift or slow loss of alignment.

• Late-stage contradiction or reversal of earlier commitments.

• Context collapse or failure to retrieve earlier information.

• Fragmentation of reasoning or loss of continuity.

Continuation or Escalation

Continue the interaction until a collapse signature expresses or is clearly avoided. Increase interaction length, introduce additional shifts, or combine long form pressure with load or ambiguity procedures to surface deeper behaviour. Extended interaction often exposes behavioural tendencies that do not appear in shorter sequences, making this procedure essential for assessing real world suitability.

5.8 Transition Procedure

Condition Definition

Evaluates the system's behaviour during transitions between states, topics, constraints, goals, or abstraction levels. This procedure tests whether the system can adapt smoothly without losing context, contradicting prior commitments, or destabilising its reasoning.

Evaluator Actions

• Shift direction abruptly or gradually.

• Introduce new objectives or remove existing ones.

• Modify, invert, or remove constraints.

• Change abstraction level or domain.

• Combine multiple transition types in sequence.

Expected Behavioural Range

• Smooth and coherent adaptation to new conditions.

• Preservation of relevant context across transitions.

• No contradiction with prior commitments unless explicitly superseded.

• Stable reasoning structure during and after transitions.

Collapse Signature Indicators

• Instability during or immediately after transitions.

• Loss of prior commitments or silent reinterpretation.

• Contradiction spikes or incompatible statements.

• Context corruption or failure to integrate new direction.

Continuation or Escalation

Repeat transitions with increasing frequency, complexity, or abruptness. Combine transitions with load, ambiguity, or constraint procedures to surface deeper behavioural characteristics. Transitions frequently surface instability because they require the system to reconcile new conditions with prior commitments without losing coherence.

5.9 Summary of Evaluation Procedures

The behavioural evaluation procedures define how evaluators introduce conditions, apply behavioural pressures, observe system responses, and identify collapse signatures across multiple behavioural surfaces. The procedures are modular and adaptable, allowing evaluators to combine or sequence them as required by the test design. They apply to any AI system operating within the open-ended environment defined in Section 2 and provide a consistent operational framework for generating comparable behavioural observations. Together these procedures create a comprehensive view of system behaviour that remains consistent across evaluators and test conditions.

Section 6 - Evaluator Roles, Responsibilities, and Evidentiary Rules

6.0 Purpose of This Section

This section defines the responsibilities, operational rules, and behavioural expectations for evaluators conducting the behavioural assessment. It establishes the standards required for consistent application of procedures, alignment with real world operational conditions, and accurate interpretation of behavioural signals across all phases of the protocol. These rules ensure that evaluator behaviour provides controlled conditions while system behaviour remains the object of measurement.

Because AI systems are non-deterministic, evaluator responsibilities focus on creating controlled conditions rather than shaping outcomes. Behaviour must be interpreted relative to the declared load envelope, ensuring that variation across runs does not undermine the consistency of the methodology.

6.1 Evaluator Role Definition

The evaluator is responsible for:

• initiating and sustaining interaction

• introducing the conditions defined in Section 5

• observing behavioural patterns and collapse signatures

• adapting interaction to match the behavioural surface under examination

• maintaining procedural integrity without influencing outcomes through optimisation, correction, or stabilisation

The evaluator's role is to determine whether the AI system can perform its intended function correctly under real world conditions. Correctness is assessed through behavioural competence, including coherence, consistency, constraint adherence, context management, and stability under pressure. Task outcomes are interpreted in relation to the system's behaviour, so that correctness, errors, and partial correctness are understood as behavioural expressions rather than isolated results. By grounding correctness in behavioural competence rather than fixed outputs, evaluators ensure that assessment reflects real operational conditions.

6.2 Real-World Input Principle

The methodology evaluates the system's behaviour under realistic workload conditions. Evaluator inputs are ordinary user inputs rather than engineered prompts, because the procedure examines how the system performs when interacting with typical, unoptimized requests. Real world inputs include:

• customer service scripts

• document drafting

• summarisation tasks

• policy interpretation

• multi step instructions

• unstructured user behaviour

• incomplete or ambiguous requests

• shifting goals or priorities

Inputs may be:

• scripted

• semi structured

• open ended • task based

• conversational • operationally realistic

The evaluator selects the input modality that reflects the environment in which the system will operate. Task outcomes may be used as behavioural evidence when relevant to the behavioural surface being examined. Using realistic inputs helps surface behavioural tendencies that may not appear under engineered or optimised prompting.

6.3 Input Characteristics

The methodology is designed to reveal system behaviour under conditions that resemble real world use. Evaluators typically obtain the clearest behavioural signal when inputs match the kinds of requests real users would make, including ordinary phrasing, realistic task descriptions, and workload aligned instructions.

In development or diagnostic contexts, evaluators may use more constrained or structured inputs when required. The methodology accommodates both approaches; the key requirement is that inputs support the behavioural surface being examined and do not distort the behavioural signal.

Real world inputs generally provide the strongest insight into:

• how the system handles evolving tasks

• how behavioural patterns influence task outcomes

• how constraints are retained or lost

• how context is managed over time

• how collapse signatures emerge

This approach ensures that the evaluation reflects operational conditions rather than laboratory artefacts, while still allowing tighter inputs when required by the test design or development stage. This alignment between input style and operational context strengthens the evidentiary value of the behavioural signals that emerge.

6.4 Condition Alignment Rule

Evaluator actions must align with the behavioural surface being measured. The evaluator introduces conditions that directly correspond to the capability under examination so that behavioural signals are attributable to the system rather than evaluator variation. Examples include:

• if testing ambiguity handling, introduce ambiguity

• if testing constraint integrity, introduce constraints and modifications

• if testing long form behaviour, sustain extended interaction

• if testing transitions, shift goals, topics, or constraints

• if testing real world workload, use real world tasks

Evaluator behaviour is condition driven and is not restricted by arbitrary exclusions. The evaluator provides the conditions under which behaviour becomes visible, while the system's responses determine the behavioural outcome. Clear alignment ensures that behavioural variation can be attributed to the system rather than inconsistencies in evaluator behaviour.

6.5 Neutrality and Non-Interference

Evaluator neutrality requires allowing the system's behaviour to emerge under the conditions being tested. Inputs must reflect realistic user interaction and must not shape the system toward preferred outcomes, stabilise it, or compensate for weaknesses

unless the procedure explicitly requires such actions. Neutrality ensures that behavioural signals originate from the system rather than from evaluator influence.

Neutrality focuses on observing:

• how the system handles errors

• how it maintains or loses context

• how it responds to behavioural pressure

• how it behaves as tasks evolve

• how it manages constraints over time

• how collapse signatures appear, develop, or resolve

Evaluator inputs create the conditions under which behaviour becomes visible. They do not pre-emptively correct, simplify, or compensate unless the test design specifies such interventions. Maintaining neutrality allows behavioural patterns to emerge naturally, providing a more accurate representation of system stability.

6.6 Consistency of Pressure Application

Evaluator pressure must be applied in a controlled and deliberate manner that aligns with the behavioural surface being examined. Pressure is applied within the load envelope defined by the declared axes and is introduced to reveal how the system adapts, maintains structure, or degrades under changing conditions.

Forms of pressure include:

• increasing task complexity

• shifting context

• modifying or removing constraints

• introducing ambiguity

• extending interaction length

• combining multiple forms of pressure when required by the procedure

Pressure must be applied consistently across comparable evaluations so that behavioural differences reflect the system rather than evaluator variation. The scale, timing, and progression of pressure follow the system's responses and the requirements of the procedure. Consistent pressure application supports fair comparison across evaluations and prevents evaluator variation from distorting behavioural interpretation.

6.7 Documentation Requirements

Evaluators must document all conditions, actions, and observations that occur during the evaluation. Documentation provides the factual basis for behavioural interpretation, scoring, and independent review. Evaluators must record:

- conditions introduced
- evaluator actions taken
- system responses
- collapse signatures observed
- phase transitions
- termination rationale

Documentation must be:

- factual
- chronological
- complete enough to reconstruct the interaction
- free of interpretation or judgement
- sufficient for independent review and governance processes

These requirements ensure reproducibility, auditability, and traceability across evaluations. Comprehensive documentation ensures that behavioural claims can be independently verified without relying on evaluator memory or interpretation.

6.8 Termination Responsibilities

The evaluator ends the evaluation when the behavioural criteria for the procedure have been met. Termination is based on the expression or avoidance of collapse signatures, the sufficiency of observed behaviour, the exhaustion of relevant conditions, or diminishing returns on further interaction. The endpoint reflects the evaluator's judgement of when the behavioural surface has been adequately examined and when additional interaction would not produce materially new behavioural evidence.

Termination must be documented with a clear rationale that links the endpoint to the conditions introduced, the system's responses, and the behavioural objectives of the procedure. Clear termination rationale strengthens the evidentiary chain by linking the endpoint directly to observed behavioural conditions.

6.9 Evidentiary Model and Evidence Sources

The behavioural evaluation methodology requires complete, factual, and auditable evidence. Evidence must document evaluator actions, system behaviour, and the conditions under which that behaviour occurred. Evidence provides the factual basis for behavioural interpretation, scoring, governance, and independent review.

Evidence may originate during the evaluation or from real world operational use when relevant to the behavioural surface being examined. All evidence must be traceable, chronological, and sufficient to reconstruct the interaction and the conditions under which behavioural signals emerged. A robust evidentiary model ensures that behavioural interpretation remains grounded in verifiable records rather than subjective impressions.

6.9.1 Definition of Evidence

Evidence is any factual record that documents evaluator actions, system outputs, and the behavioural conditions under which those outputs were produced. Evidence must allow the interaction to be reconstructed, behavioural claims to be verified, and independent review to be conducted without ambiguity.

Evidence includes:

- evaluator inputs and system outputs
- interaction logs and transcripts
- task artefacts produced during the evaluation
- real world artefacts from normal work
- user generated materials, including screen recordings
- evaluator notes documenting conditions and actions
- system generated metadata such as timestamps, phase markers, and axis tags
- environmental or contextual information relevant to interpretation

Evidence must be factual, chronological, and unedited. Interpretation is reserved for scoring and review, not for evidence creation. Separating factual evidence from interpretation preserves the integrity of the record and supports consistent review.

6.9.2 Valid Evidence Sources

Valid evidence sources are those that provide factual, chronological, and traceable records of evaluator actions, system behaviour, and the conditions under which behaviour occurred.

Valid sources include:

• complete interaction logs

• evaluator side transcripts

• system generated transcripts

• structured logging outputs

• task artefacts created during the evaluation

• real world operational artefacts

• evaluator notes documenting conditions and actions

• system metadata required for traceability

These sources collectively form the evidentiary basis for behavioural scoring, classification, and independent review. Using multiple evidence sources strengthens traceability and reduces the risk of gaps in the behavioural record.

6.9.3 Real-World Artefacts

Real world artefacts are admissible when they reflect the system's operational environment and provide factual context for interpreting behaviour.

Real world artefacts include:

• documents produced during normal work

• user generated task materials

• workflow artefacts

• domain specific materials used during evaluation

Real world artefacts support interpretation by showing how the system behaves under realistic operational conditions and by providing contextual grounding for behavioural signals observed during the evaluation. Such artefacts help evaluators understand how behavioural patterns manifest in practical, operational settings.

6.9.4 User-Generated Recordings

User generated recordings are admissible when they reflect the system's operational environment and capture the interaction clearly enough to support independent review. Recordings document the sequence of user inputs and system outputs in a factual and traceable manner. They do not need to contain evaluator inputs.

Valid recordings include:

• screen recordings

• audio recordings

• video recordings

• mixed modality recordings

Recordings must be complete enough to establish sequence, context, and the behavioural conditions under which system outputs were produced. Evaluator evaluations and comments are documented separately and linked to the recording. The recording remains valid evidence because the evaluator is identifying collapse signatures and their lead up in a way that another evaluator would reasonably agree with. Recordings also provide an objective reference that supports independent verification of behavioural claims.

6.9.5 Evidence Bundle Requirements

Evidence bundles must contain all materials required to reconstruct the interaction, verify behavioural claims, and support independent review. A bundle links the factual artefacts with the evaluator's documented observations so that another evaluator can follow the same evidentiary path and reach a reasonably similar interpretation.

Each evidence bundle must include:

• **interaction records** that document user inputs and system outputs

• **task artefacts** created during the interaction

• **system metadata** required for traceability

• **evaluator notes** identifying collapse signatures, lead up conditions, and relevant behavioural transitions

• **linkage references** that connect evaluator commentary to the corresponding artefacts

• **termination rationale** when the bundle covers a complete evaluation

Evidence bundles must be factual, chronological, and complete enough for an independent evaluator to reconstruct the behavioural surface, confirm the presence or

absence of collapse signatures, and validate the evaluator's reasoning. A complete bundle ensures that another evaluator can follow the same evidentiary path and reach a comparable interpretation.

6.9.6 Evidence Completeness Classification

Evidence is classified according to completeness:

• **Complete** - full logs, artefacts, and recordings are present and chronological.

• **Partial** - some evidence is missing, truncated, or unavailable, but the remaining evidence is sufficient for behavioural interpretation.

• **Missing** - evidence gaps prevent full reconstruction of behavioural conditions, and findings must be qualified accordingly.

Incomplete or partial evidence does not invalidate an evaluation. It is marked as such and interpreted with appropriate caution. Evaluators must note missing elements and their potential impact on interpretation. Clear classification helps reviewers understand the limits of the available evidence and interpret findings appropriately.

6.10 Summary of Evaluator Responsibilities

Evaluators create conditions that reflect real-world use, introduce behavioural pressures appropriate to the procedure, maintain neutrality, and document behaviour factually and chronologically. Evaluators must produce complete, auditable evidence, including logs, artefacts, and recordings when relevant. Termination is based on behavioural sufficiency or collapse expression. Evaluator behaviour provides controlled conditions; system behaviour is the object of measurement; evidence provides the factual basis for interpretation, scoring, and governance. These responsibilities ensure that behavioural evaluation remains consistent, evidence driven, and aligned with real world operational demands.

Section 7 - Behavioural Dimensions and Scoring Architecture

7.0 Purpose of This Section

This section defines the behavioural dimensions, scoring architecture, and classification framework used to interpret system behaviour observed during evaluation. It establishes the criteria for identifying behavioural competence, instability, and collapse signatures across all procedures defined in Section 5.

Behavioural dimensions are scored. Load axes -comprising evaluator-declared axes and system-measured advisory axes -are not scored. Axes define the background conditions under which behaviour is interpreted; they do not contribute directly to behavioural scores.

Because AI systems are non-deterministic, behavioural scoring must be interpreted relative to the load envelope. This ensures that variation across runs does not distort scoring and that behaviour is assessed within the operational context rather than against fixed outputs.

7.1 Behavioural Dimension Overview

Behaviour is evaluated across a set of independent dimensions that describe distinct behavioural properties observable under real-world workload conditions. These dimensions capture how the system maintains structure, responds to demands, manages context, and adapts to evolving task requirements. They are foreground behavioural qualities, not load axes.

Coherence The degree to which outputs maintain logical structure, relevance, and internal alignment.

Consistency The stability of behaviour across turns, including adherence to prior commitments and established patterns.

Constraint Integrity The system's ability to maintain declared rules, boundaries, and operational constraints without drift or erosion.

Context Management The accuracy and stability with which prior information is retained, applied, and updated.

Ambiguity Handling The system's ability to manage uncertainty, incomplete information, or unclear instructions without over-assertion or collapse.

Responsiveness The degree to which a reasonable person would judge the system's behaviour as responsive to their needs, situation, and requests. This includes timeliness,

relevance, usefulness, clarity, and structure of the output, as well as the behavioural attitude expressed in the response. The expected responsiveness threshold varies by scenario and must be defined before evaluation.

Adaptation The system's ability to adjust behaviour appropriately when task conditions, user needs, or contextual signals change.

Long-Form Stability The capacity to maintain coherence, direction, and constraint integrity across extended interactions or multi-step tasks.

Transition Stability The reliability with which the system handles shifts between topics, tasks, or phases without degradation.

These dimensions provide the framework for evaluating behavioural performance. They are scored independently and interpreted within the load envelope defined by the evaluator-declared axes. Taken together, these dimensions provide a structured view of behaviour that remains distinct from the background load defined by the axis model.

7.1.1 Pre-Test Expectation Setting

Before evaluation begins, the evaluator establishes the expected behavioural performance level for the system under test. These expectations define the baseline against which all behavioural dimensions are interpreted.

Expectations may include anticipated levels of coherence, consistency, constraint integrity, context management, ambiguity handling, responsiveness, adaptation, long-form stability, and transition stability. Expectations reflect the system's intended operational environment, known characteristics, and design constraints.

Expectation setting ensures that scoring reflects whether the system meets the performance level appropriate to the scenario. Predictable or intentional characteristics—such as slower response generation, terse output style, high verbosity, or conservative ambiguity handling—are not penalised when declared in the expectation profile when they align with the declared expectations.

Real-world evaluation requires understanding the evaluator's threshold for timely, relevant, or helpful behaviour. Some scenarios prioritise short, rapid replies; others require detailed, structured output. Expectations anchor the evaluation to the operational context rather than to a universal behavioural ideal.

Scoring assesses whether the system meets the expected performance level for the test conditions, rather than whether it conforms to an arbitrary or universal standard. Clear expectations allow evaluators to distinguish between acceptable variation and meaningful behavioural deviation.

7.1.2 Expectation-Setting Template

Expectation-setting establishes the evaluator's declared operational expectations before testing begins. These declarations define the behavioural reference point against which deviation and collapse are later assessed. The evaluator records the following elements for every scenario:

- **Declared Task Outcome** - the intended result the system is expected to produce.
- **Declared Behavioural Performance Level** - the expected behavioural quality and stability the system should maintain while producing the outcome.
- **Declared Load Axes** - the evaluator-applied load conditions under which the system will operate.
- **Success and Deviation Conditions** - the criteria defining acceptable performance, deviation thresholds, and the boundary between recoverable deviation and collapse.

These declarations anchor the evaluation to the intended operational profile and prevent misinterpretation of behaviour that aligns with known system characteristics.

7.2 Behavioural Deviation Conditions

Behavioural deviation occurs when the system's behaviour diverges from the performance level defined during pre-test expectation setting. Deviation is assessed relative to the expected behavioural profile for the scenario, not against a universal standard. A behaviour is considered a deviation when it falls outside the evaluator's declared threshold for adequacy along any behavioural dimension as defined in the pre-test expectation profile.

Deviation may arise through reduced coherence, inconsistency with prior commitments, erosion of constraint integrity, loss or distortion of context, inappropriate handling of ambiguity, insufficient responsiveness, failure to adapt to changing conditions, degradation during long-form interaction, or instability during transitions. The specific form of deviation depends on the dimension affected and the expectations established before testing.

A deviation does not automatically constitute a collapse. Minor deviations may be recoverable, expected, or acceptable within the scenario. Deviation becomes meaningful when it signals a departure from the intended operational profile, impairs task performance, or indicates the onset of behavioural degradation relative to the declared expected performance level These conditions form the basis for identifying collapse signatures in subsequent sections.

Deviation is interpreted relative to the behavioural performance level defined in the pre-test expectation profile. Interpreting deviation in this way ensures that evaluators focus on departures from the declared performance level rather than on universal or idealised standards.

7.3 Collapse Signature Framework

Collapse signatures are behavioural deviations from the expected behavioural performance level defined during pre-test expectation setting. They indicate that the system is departing from its intended operational profile as defined in the pre-test expectation profile. Collapse signatures are not errors in themselves; they are diagnostic signals that reveal the nature and severity of behavioural degradation.

Collapse signatures are interpreted in relation to the behavioural performance level defined in the pre-test expectation profile. This framing helps evaluators distinguish between ordinary variation and genuine behavioural degradation.

7.3.1 Severity Levels

Minor Small, recoverable deviations that do not materially disrupt task flow and can be corrected with minimal user intervention.

Noticeable Clear deviations that require user correction or redirection and temporarily disrupt the intended workflow.

Significant Substantial deviations that impair task continuity, undermine reliability, or require structured intervention to restore alignment.

Critical Structural failures that prevent continuation, require full reset or re-initialisation, or indicate loss of functional integrity.

Severity levels provide a consistent structure for interpreting how far behaviour has diverged from the expected performance level.

7.3.2 Minor Collapse Signatures

Drift Gradual deviation from constraints or context that is easily corrected when highlighted.

Over-Assertion Unwarranted confidence under ambiguity that resolves once the ambiguity is clarified.

7.3.3 Noticeable Collapse Signatures

Narrowing Reduction in scope, capability, or responsiveness that limits the system's ability to address the full task.

Looping Repetitive or stuck behaviour that requires user interruption to break the cycle.

7.3.4 Significant Collapse Signatures

Contradiction Conflict with prior commitments or internal logic that disrupts coherence and requires explicit correction.

Fabrication Unsupported assertions or invented details that compromise reliability and require re-anchoring to source material.

7.3.5 Critical Collapse Signatures

Fragmentation Breakdown of structure or coherence such that the system cannot maintain continuity of reasoning or output.

Context Corruption Distortion, loss, or misapplication of prior information that prevents continuation without reset or re-initialisation.

Severity grading enables evaluators to distinguish between minor recoverable deviations and critical failures that prevent continuation. All collapse signatures are interpreted relative to the expectations established before testing, ensuring that evaluation reflects real-world operational conditions. Critical signatures indicate that the system can no longer maintain its intended operational profile without reset or re-initialisation.

7.4 Scoring Architecture Overview

Scoring evaluates how well the system meets the behavioural performance level defined during pre-test expectation setting. Testing regimes select their own scoring resolution based on how granular they need the behavioural signal to be. Some contexts benefit from coarse-grained scoring, while others require finer distinctions across behavioural dimensions. Any level of granularity is supported, provided the criteria are defined in advance and applied consistently.

Scoring is always interpreted relative to the behavioural performance level defined in the pre-test expectation profile, not against a universal standard.

Scoring reflects observable behavioural outcomes, including behavioural competence, the presence or absence of collapse signatures, stability under behavioural pressures within the load envelope, adaptation during transitions, and the integrity of constraints and context. These factors indicate how reliably the system maintains its intended operational profile under real-world conditions.

These outcomes are evaluated against the expectations declared before testing, ensuring that scoring reflects the intended operational profile rather than an abstract ideal.

Each behavioural dimension is scored independently. The scoring architecture produces a multidimensional behavioural profile rather than a single aggregate value, ensuring that strengths and weaknesses across dimensions remain visible and interpretable. This approach ensures that scoring reflects behavioural reliability under real world conditions rather than abstract correctness.

7.4.1 Non-Numeric Evaluation Principles

The scoring architecture does not use percentages, accuracy scores, or numerical correctness metrics. Evaluators may derive percentages for internal reporting if they choose, but such numbers are not part of behavioural evaluation and must not be interpreted as indicators of reliability, competence, or safety.

Percentages create a false sense of precision. A system that is “90% correct” in a legal document, financial report, safety-critical instruction, or policy interpretation is not “mostly correct” - it is effectively 100% wrong, because the remaining 10% may contain a fabricated statute, an incorrect financial figure, a misapplied safety rule, a hallucinated requirement, or a broken constraint. In these domains, any incorrect output can invalidate the entire artefact.

For this reason, scoring is based on observable behavioural outcomes, deviations, collapse signatures, and stability under load, not on numerical correctness.

Evaluators may repeat the same test multiple times to observe behavioural stability, variability, and repeatability. Repeated runs do not produce accuracy scores; they reveal whether deviations are isolated, systemic, load-triggered, intermittent, escalating, or self-correcting.

A system that produces a correct output nine times and a catastrophic error once is not “90% correct” - it is unreliable, because the single failure may invalidate the entire task.

Repeated runs are therefore treated as behavioural evidence, not as a basis for numerical scoring. If organisations wish to map deviations to percentages or internal scoring systems, they may do so, but such mappings are external to the methodology and must not be treated as part of the behavioural model. Non numeric scoring preserves the behavioural focus of the methodology and avoids misleading interpretations of reliability.

7.5 Competence Levels (A / B / C)

Behavioural performance is classified into three competence levels. These levels reflect whether the system meets, falls short of, or fails to meet the expected performance thresholds defined prior to evaluation.

Level A - Meets or Exceeds Expected Performance The system demonstrates stable, coherent, and contextually aligned behaviour across all phases of interaction. Constraint integrity is maintained, ambiguity is handled appropriately, and long-form stability is preserved. A reasonable evaluator would judge the system as operationally reliable and suitable for deployment under the intended real-world conditions.

Level B - Below Expected Performance but Operationally Functional The system exhibits partial instability, including intermittent drift, reduced responsiveness, inconsistent constraint adherence, or occasional context loss. Behaviour remains usable but unreliable. A reasonable evaluator would judge the system as operationally variable and suitable only for restricted or supervised deployment.

Level C - Fails Expected Performance / Operationally Unreliable The system demonstrates structural behavioural failure, including contradiction cascades, context corruption, fragmentation, narrowing, or loss of constraint integrity. Behaviour is unreliable and unsuitable for deployment. A reasonable evaluator would judge the system as failing to meet the expected performance threshold for the intended scenario.

Competence levels are always interpreted in relation to the required performance level established during pre-test expectation setting. Competence levels are determined by the severity and persistence of deviations and collapse signatures observed across procedures and tasks. A system is not judged against an absolute or universal standard, but against the performance level appropriate to the scenario in which it is intended to operate.

These competence levels serve as the behavioural reliability categories used throughout the methodology. Competence levels summarise behavioural reliability in a form suitable for deployment decisions and governance review.

7.6 Cross-Dimension Interpretation

Behaviour is interpreted across dimensions to identify:

- global stability
- localised weaknesses
- systemic collapse patterns
- transition-specific vulnerabilities

- vulnerabilities expressed under specific declared load axes

Cross-dimension interpretation reveals whether failures are:

- isolated
- systemic
- load-triggered
- ambiguity-triggered
- context-triggered
- transition-triggered

This enables comparative evaluation across systems.

Cross dimension interpretation is performed relative to the behavioural performance level defined before testing. Cross dimension analysis reveals whether behavioural weaknesses are isolated or indicative of broader systemic instability.

7.7 Behavioural Competence Classification

Systems are classified into behavioural competence levels:

Level A - Operationally Reliable

- Stable across all major dimensions
- Collapse signatures rare or absent
- Correct task performance under real-world conditions
- Robust under load, ambiguity, and transitions

Level B - Operationally Variable

- Stable in some dimensions, degraded in others
- Collapse signatures intermittent
- Correctness inconsistent under behavioural pressures within the load envelope
- Vulnerable to specific load or transition types

Level C - Operationally Unreliable

- Frequent collapse signatures
- Behaviour unstable under real-world conditions
- Correctness unreliable

- Significant vulnerabilities across multiple dimensions

These classifications support deployment decisions and provide a consistent framework for comparing systems across scenarios and operational contexts.

7.8 Summary of Behavioural Scoring Architecture

The behavioural scoring architecture defines the dimensions, collapse signatures, scoring tiers, and classification framework used to interpret AI behaviour under real-world workload conditions. It provides a structured, domain-agnostic method for determining whether a system can perform its intended function reliably and correctly. The architecture ensures that behavioural scoring remains grounded in observable evidence and aligned with the evaluator's declared expectations.

Section 8 - End-to-End Evaluation Workflow

8.0 Purpose of This Section

This section defines the complete operational workflow for conducting a behavioural evaluation. It specifies the sequence of actions, phase transitions, documentation requirements, and decision points required to execute the methodology consistently across evaluators, systems, and environments.

Because AI systems are non-deterministic, the workflow ensures that behavioural interpretation remains tied to the load envelope. This prevents run-to-run variation from distorting the evaluation and keeps behavioural assessment anchored to operational context rather than fixed outputs.

8.1 Workflow Overview

The evaluation workflow consists of:

1. Preparation
2. Initiation Phase
3. Exploration Phase
4. Load Phase
5. Long-Form Behaviour Phase
6. Transition Phase
7. Termination
8. Documentation and Scoring
9. Review and Classification

Each stage is executed according to the procedures defined in Section 5 and the evaluator responsibilities defined in Section 6. This structure helps evaluators interpret behavioural changes in relation to the conditions present at each stage rather than treating behaviour as static.

8.2 Preparation Stage

8.2.1 Define Operational Context

Identify the real-world environment in which the AI system is intended to operate. This determines:

- input modality
- workload type
- expected constraints
- interaction patterns
- domain-specific pressures

The evaluator must declare the behavioural performance level for the scenario before testing begins. Declaring expectations at this stage ensures that later behavioural deviations are interpreted relative to the intended operational profile.

8.2.2 Select Relevant Procedures

Choose the behavioural procedures from Section 5 that correspond to:

- the system's intended function
- the behavioural surfaces of interest
- the operational risks
- the deployment environment

Selecting procedures in this way ensures that evaluation focuses on the behavioural surfaces most relevant to the system's real-world use.

8.2.3 Establish Documentation Framework

Prepare a structured log for:

- evaluator actions
- system responses
- collapse signatures
- phase transitions
- termination rationale

Documentation must be chronological and factual. A consistent documentation structure supports independent review and prevents ambiguity in later interpretation.

8.2.4 Baseline Considerations

Baseline refers to the model's nominal operating behaviour. Evaluators begin with the normal baseline (neutral conditions) and may progress toward the operational baseline when the system's real world nominal state differs and when such conditions are required or possible to simulate. Baseline behaviour remains diagnostically valuable even when operational conditions cannot be reproduced: if a system is unstable, inconsistent, or only marginally correct under ideal conditions, it will be less reliable under real-world conditions, which are always inferior to test conditions. A well-defined baseline provides the reference point needed to interpret later behavioural shifts under load.

8.3 Initiation Phase Execution

The evaluator:

- introduces neutral prompts
- establishes baseline behaviour
- observes initial coherence, responsiveness, and constraint integrity
- avoids introducing load, ambiguity, or transitions

Transition to the next phase occurs when baseline behaviour is sufficiently characterised. This phase establishes the behavioural foundation upon which all subsequent observations are interpreted.

8.4 Exploration Phase Execution

The evaluator:

- introduces real-world tasks or interactions
- uses natural, workload-aligned inputs
- observes spontaneous behavioural tendencies
- identifies early collapse signatures

Transition occurs when behavioural patterns emerge or stabilise. Exploration reveals the system's natural behavioural tendencies before structured pressure is applied.

8.5 Load Phase Execution

The evaluator:

- increases complexity under the evaluator-declared load axes
- introduces behavioural pressure under the declared load axes
- varies abstraction, specificity, and domain
- introduces simultaneous constraints
- observes behavioural degradation or resilience

Transition occurs when load-related behaviour is sufficiently expressed. Load testing exposes how behaviour changes as demands increase, revealing both resilience and degradation patterns.

8.6 Long-Form Behaviour Phase Execution

The evaluator:

- sustains extended interaction
- maintains realistic workload conditions
- introduces shifts in goals, context, or constraints
- observes long-form stability or gradual collapse

Transition occurs when long-form behaviour is fully expressed. Extended interaction often surfaces behavioural tendencies that do not appear in shorter sequences.

8.7 Transition Phase Execution

The evaluator:

- introduces abrupt or gradual changes
- shifts objectives, constraints, or topics
- observes adaptation, coherence, and constraint integrity
- identifies transition-specific collapse signatures

Transition occurs when adaptation behaviour is sufficiently expressed. Transitions frequently reveal instability because they require the system to reconcile new conditions with prior commitments.

8.8 Termination Stage

The evaluator ends the evaluation when the behavioural criteria for the procedure have been met. Termination occurs when collapse signatures have expressed or been avoided, behavioural sufficiency has been reached, relevant conditions have been exhausted, or additional interaction yields no new signals. The endpoint reflects the evaluator's judgement of when the behavioural surface has been fully examined. Ending the evaluation at the appropriate point ensures that behavioural evidence is complete without introducing unnecessary noise.

8.9 Documentation and Scoring Stage

8.9.1 Documentation Requirements

The evaluator records:

- declared expectations and conditions introduced
- evaluator actions
- system responses
- collapse signatures
- phase transitions
- termination rationale

Documentation must be factual, chronological, and free of interpretation. Clear documentation allows another evaluator to reconstruct the interaction and verify behavioural claims.

8.9.2 Scoring Application

Apply the scoring architecture defined in Section 7:

- score each behavioural dimension
- identify collapse signatures
- classify behavioural competence
- determine operational reliability

Scores are assigned independently for each behavioural dimension, producing a multidimensional profile rather than a single combined value. Scoring remains grounded in observable behaviour and the expectations declared before testing.

8.10 Review and Classification Stage

Ideally a review panel or secondary evaluator:

- verifies documentation
- confirms scoring
- assesses cross-dimension patterns
- assigns behavioural competence level
- identifies deployment risks

This ensures reproducibility and governance. Independent review strengthens governance by ensuring that behavioural interpretation is consistent across evaluators.

8.11 Summary of End-to-End Workflow

The end-to-end workflow defines the complete operational process for conducting a behavioural evaluation, from preparation through termination and scoring. It ensures consistent, reproducible assessment of AI behaviour under real-world workload conditions, independent of domain, system architecture, or evaluator. This workflow ensures that behavioural evaluation remains reproducible, evidence-driven, and aligned with real world operational demands.

Section 9 - Conclusion and Forward-Use Guidance

9.0 Purpose of This Section

This section summarises the methodology's purpose, scope, and operational implications. It provides guidance for applying the behavioural evaluation framework in deployment, governance, and comparative assessment contexts.

Because AI systems are non-deterministic, the conclusions drawn from behavioural evaluation must always be interpreted relative to the load envelope. This ensures that operational decisions are grounded in contextual behavioural evidence rather than fixed outputs or deterministic assumptions.

9.1 Summary of Methodology Purpose

This methodology establishes a structured, standards-grade framework for evaluating AI system behaviour under real-world workload conditions. It defines:

- behavioural dimensions
- collapse signatures
- evaluator responsibilities
- procedural phases
- scoring architecture
- classification criteria

The methodology determines whether an AI system can perform its intended function correctly, reliably, and coherently in operational environments. This framing ensures that behavioural assessment reflects real operational demands rather than idealised or benchmark-driven expectations.

9.2 Behaviour-Centric Evaluation Principle

The methodology evaluates how the system behaves while attempting to perform its intended function under realistic conditions. Correctness, partial correctness, and errors are interpreted through behavioural competence, including coherence, consistency, constraint integrity, and context management. The evaluation reflects real workloads, real constraints, real user behaviour, and real operational pressures, ensuring alignment with deployment realities. By interpreting correctness through behavioural competence, evaluators can assess reliability even when outputs vary across runs.

9.3 Applicability Across Systems and Domains

The methodology is system-agnostic and domain-agnostic. It applies to:

- general-purpose AI systems
- domain-specific AI systems
- task-oriented agents
- customer-facing systems
- internal operational systems

It is suitable for:

- pre-deployment evaluation
- post-deployment monitoring
- comparative assessment
- risk analysis
- governance and oversight

The methodology does not require domain expertise to execute, only adherence to behavioural procedures. This universality allows the methodology to support consistent evaluation across diverse architectures, domains, and deployment contexts.

9.4 Interpretation of Results

Results must be interpreted in behavioural terms:

- stability
- coherence
- consistency
- constraint integrity
- context management
- adaptation
- long-form reliability
- transition resilience

Scores are assigned independently for each behavioural dimension, producing a multidimensional profile. Behavioural competence classification reflects operational

reliability by examining how the system performs its intended function under real-world conditions. Interpreting results in this way keeps the focus on behavioural reliability under the conditions in which the system is expected to operate.

9.5 Deployment and Governance Implications

Behavioural evaluation results inform:

- deployment readiness
- operational risk assessment
- model selection
- policy alignment
- safety controls
- monitoring requirements

Systems classified as operationally unreliable require mitigation, redesign, or restricted deployment. Using behavioural evidence in governance processes strengthens decision-making by grounding risk assessments in observable system behaviour.

9.6 Limitations and Boundaries

The methodology focuses on behavioural reliability: how the system performs its intended function under real-world conditions, how behavioural patterns influence task outcomes, and how stability is maintained or lost over time. Domain knowledge, benchmark performance, and prompt-engineering sensitivity are addressed by other testing regimes when required. This procedure complements those approaches by providing a behavioural perspective on operational reliability. Recognising these boundaries ensures that behavioural evaluation is used appropriately alongside other testing regimes.

9.7 Forward-Use Guidance

The methodology supports:

- iterative evaluation
- regression testing
- model comparison
- model updates

- monitoring of behavioural drift
- integration into organisational governance frameworks

Evaluators should re-apply the methodology when:

- system behaviour changes
- model versions update
- deployment context shifts
- operational risks evolve

Behavioural evaluation is an ongoing process, not a one-time certification. Regular re-evaluation helps organisations detect behavioural drift early and maintain alignment with evolving operational requirements.

9.8 Final Statement

This methodology provides a comprehensive, reproducible framework for assessing AI system behaviour under real world conditions. It enables organisations to determine whether a system can perform its intended function correctly and reliably, identify behavioural vulnerabilities, and make informed deployment decisions grounded in observable behavioural evidence.

The methodology supports evidence-based reporting by producing structured, auditable behavioural records suitable for governance, assurance, and comparative evaluation. The evidentiary structure is designed to reduce evaluator burden, improve consistency, and strengthen the quality of behavioural reporting without relying on any specific implementation or mechanism for capturing the underlying evidence. In doing so, the methodology provides a durable foundation for long-term behavioural oversight and system reliability management.

Section 10 - Governance Layer

10.0 Purpose of This Section

This section defines governance requirements for organisations applying the behavioural evaluation methodology. It ensures evaluations are valid, reproducible, and aligned with operational realities while allowing flexibility where the test design requires non-standard inputs or interaction patterns.

Because AI systems are non-deterministic, governance must ensure that behavioural interpretation remains tied to the load envelope. This prevents run-to-run variation from distorting results and keeps behavioural assessment grounded in observable behaviour rather than deterministic assumptions.

10.1 Real-World Condition Principle

Evaluators should strive to use inputs and conditions that reflect real-world operation unless the test explicitly requires a different input style or interaction pattern. Short prompts, engineered prompts, or highly optimised prompts may be used when they form a part of the overall test design. Engineered prompts produce engineered behaviour and therefore provide limited insight into real-world performance unless that is the intended focus of the evaluation. Using real world inputs helps ensure that behavioural signals reflect operational conditions rather than artefacts of engineered prompting.

10.2 Evaluator Conduct Requirements

Evaluator behaviour should preserve the integrity of the behavioural signal unless the test explicitly requires a different interaction pattern. By default, evaluators avoid rescuing the system, correcting the system, leading the system, or compensating for system weaknesses. These behaviours may be used when they form part of the test design, such as evaluating user-side mitigation, workflow preservation, recovery behaviour, or other scenarios where user intervention is itself the subject of the evaluation. Maintaining this posture allows behavioural patterns to emerge naturally, providing a clearer view of system stability.

10.3 Evaluator Consistency and Cognitive Load

Behavioural evaluation requires sustained attention, consistent judgement, and accurate reconstruction of interaction sequences. Evaluators degrade under prolonged

load, and inconsistency between evaluators is expected unless managed explicitly. These factors directly affect the validity and reliability of behavioural evidence.

Organisations should structure evaluation practice to maintain evaluator reliability. Recommended practices include:

- **Session Duration Limits** - Evaluators work in bounded sessions. Extended, uninterrupted evaluation increases the risk of missed collapse signatures, misclassification, and inconsistent reconstruction.
- **Mandatory Breaks** - Evaluators take regular breaks to maintain attention and reduce cognitive load.
- **Evaluator Rotation** - For long or multi-sequence evaluations, evaluators are rotated to prevent fatigue-driven inconsistency.
- **Calibration Sessions** - Evaluators periodically review shared examples to maintain alignment in identifying drift, instability, and collapse modes.
- **Cross-Review of Significant Events** - High-impact behavioural incidents are reviewed by more than one evaluator to ensure consistency and reduce subjective variance.
- **Inter-Evaluator Agreement Monitoring** - Organisations periodically measure agreement across evaluators to ensure that behavioural classifications remain stable and reproducible.

These practices ensure that evaluator variability does not undermine the integrity of the behavioural assessment or the evidential record. These practices ensure that evaluator variability does not obscure genuine behavioural signals.

10.4 Evaluator Reconstruction Governance

Evaluator reconstruction is an interpretive process, not an objective record. Collapse signatures, drift onset, phase boundaries, and recovery classifications all depend on evaluator interpretation of the interaction sequence. Reconstruction errors, omissions, or misinterpretations can propagate directly into behavioural classifications and affect the validity of the assessment.

Organisations must treat reconstruction as evidence, not ground truth. Governance requirements include:

- **Reconstruction Transparency** - Evaluators must document the basis for their reconstruction, including the observations and interaction elements that informed each classification decision.

- **Interpretive Variability Acknowledgement** - Organisations must recognise that different evaluators may reconstruct the same sequence differently. Divergence is expected and must be managed, not treated as evaluator error.
- **Cross-Checking of Critical Events** - Drift onset, collapse signatures, and recovery transitions should be cross-reviewed by a second evaluator when they materially affect the behavioural assessment.
- **Reconstruction Error Management** - When reconstruction errors are identified, they must be corrected transparently and the impact on downstream classifications assessed.
- **Reconstruction as Evidence, Not Fact** - Reconstruction contributes to the evidential record but does not override the raw interaction data. It must be treated as an interpretive layer, not a definitive account of system behaviour.

These requirements ensure that evaluator reconstruction remains a governed, auditable interpretive process rather than an unexamined source of classification error. Treating reconstruction as interpretive rather than definitive preserves the integrity of the evidentiary record.

10.5 Behavioural Drift vs Task-Level Failure

Behavioural evaluation does not measure task correctness. A system may produce accurate task outputs while exhibiting behavioural instability or may produce incorrect outputs while remaining behaviourally stable. Task-level failure is not evidence of behavioural drift, and behavioural drift is not defined by factual error.

Evaluators must distinguish between:

- **Behavioural Drift** - changes in stability, coherence, constraint integrity, context management, or interaction behaviour.
- **Task-Level Failure** - incorrect, incomplete, or suboptimal task outputs that do not arise from behavioural instability.

Governance requirements:

- **Behavioural markers, not correctness** - Drift and collapse signatures must be identified through behavioural indicators, not task outcomes.
- **Correctness does not imply stability** - Accurate task output does not validate behavioural integrity.

- **Error does not imply drift** - Domain errors, factual mistakes, or poor task performance are not behavioural collapse unless accompanied by behavioural signatures.
- **Dual-track assessment** - Behavioural integrity and task correctness must be evaluated independently when both are relevant to the test design.
- **Classification discipline** - Evaluators must not infer behavioural instability from task failure unless behavioural evidence supports that conclusion.

This separation ensures that behavioural evaluation remains focused on system behaviour rather than domain accuracy, preventing misclassification of task errors as behavioural drift or collapse. This separation prevents evaluators from misclassifying domain mistakes as behavioural instability.

10.6 Behavioural Influence Governance

Behavioural influence refers to changes in task outcomes that arise from the system's behavioural state rather than from domain knowledge, task difficulty, or normal variation. A system may alter its output, decision-making, or task approach due to instability, drift, constraint loss, or other behavioural factors. These effects must be distinguished from ordinary task-level error.

Governance requirements:

- **Behavioural Influence Definition** - Behavioural influence occurs when the system's behaviour alters the task outcome independently of task difficulty or factual knowledge.
- **Separation from Task Error** - Incorrect or suboptimal task output is not behavioural influence unless accompanied by behavioural indicators such as drift, instability, or constraint degradation.
- **Detection Through Behavioural Markers** - Influence must be identified through behavioural evidence (e.g., loss of coherence, inappropriate tone shifts, constraint violations), not through correctness metrics.
- **Impact Assessment** - When behavioural influence affects task outcomes, evaluators must document the behavioural mechanism and its effect on the task result.
- **Non-Attribution of Normal Variation** - Normal variation in reasoning, style, or phrasing must not be classified as behavioural influence unless it materially alters the task outcome.

This ensures that behavioural influence is governed as a behavioural phenomenon rather than conflated with task-level performance or domain accuracy. This separation ensures that behavioural influence is identified through behavioural evidence rather than inferred from correctness alone.

10.7 Constraint Integrity Governance

Constraint integrity refers to the system's ability to maintain required behavioural constraints throughout an interaction. Constraints include tone, safety boundaries, interaction rules, persona requirements, formatting obligations, and any behavioural limits defined by the test design or deployment environment. Loss of constraint integrity is a behavioural event, not a task-level error.

Governance requirements:

- **Constraint Definition** - A behavioural constraint is any rule, limit, or requirement that governs how the system must behave independently of task content or correctness.
- **Constraint Integrity** - Constraint integrity is maintained when the system consistently adheres to its behavioural constraints across phases, tasks, and interaction conditions.
- **Constraint Loss Identification** - Constraint loss occurs when the system violates a behavioural constraint through tone shifts, persona collapse, safety boundary breaches, or other deviations not attributable to task difficulty or evaluator input.
- **Distinguishing from Normal Variation** - Stylistic variation, harmless rephrasing, or benign differences in expression must not be classified as constraint loss unless they materially violate a defined behavioural requirement.
- **Constraint-Linked Collapse Signatures** - When constraint loss contributes to drift or collapse, evaluators must document the behavioural mechanism and its effect on system stability.
- **Constraint Enforcement in Interpretation** - Organisations must treat constraint integrity as a first-class behavioural dimension when interpreting results, independent of task correctness or factual accuracy.

This ensures that constraint integrity is governed explicitly, preventing evaluators from conflating constraint loss with task-level error or normal variation. Clear distinction between constraint integrity and task correctness strengthens the accuracy of behavioural classification.

10.8 Phase Boundary Ambiguity Governance

Phases are analytical constructs used to organise behavioural observation. They are not discrete system states and should not be treated as strict or mutually exclusive. Behaviour may transition gradually, partially, or ambiguously between phases, and evaluators must classify these transitions consistently. Phase ambiguity is expected and does not invalidate an evaluation unless it prevents accurate identification of behavioural events.

Governance requirements:

- **Analytical Nature of Phases** - Phases represent interpretive frames for understanding behaviour, not rigid boundaries. Behaviour may overlap or blend across phases without constituting evaluator error.
- **Phase Transition Identification** - Evaluators must document the behavioural indicators that support a transition between phases, recognising that transitions may be partial, gradual, or incomplete.
- **Phase Contamination Recognition** - When behaviour characteristic of one phase appears within another, evaluators must classify it as phase contamination rather than mis-phasing, unless the overall behavioural pattern supports a full transition.
- **Mixed-Phase Behaviour** - Behaviour that exhibits characteristics of multiple phases must be recorded as mixed-phase behaviour. This is a valid classification and does not require forcing the interaction into a single phase.
- **Premature or Delayed Transitions** - Early collapse signatures, early recovery-like behaviour, or delayed transitions must be documented as such. These events do not invalidate the evaluation and may provide important behavioural insight.
- **Phase Ambiguity and Validity** - Ambiguity in phase boundaries does not affect validity unless it prevents evaluators from identifying collapse signatures, behavioural drift, or recovery behaviour.

This governance ensures that evaluators treat phases as interpretive tools rather than rigid states, preventing misclassification caused by forcing behaviour into artificial boundaries. Recognising phase ambiguity as normal prevents evaluators from forcing behaviour into artificial boundaries.

10.9 Context Management Governance

Context management refers to the system's ability to retain, apply, and update relevant interaction context without distortion, omission, or inappropriate overwriting. Context

loss, context distortion, and context misalignment are behavioural events and must be classified independently of task correctness or evaluator expectation.

Governance requirements:

- **Context Definition** - Context includes all information the system must retain to maintain behavioural continuity, including instructions, constraints, prior interaction elements, and relevant task framing.
- **Context Integrity** - Context integrity is maintained when the system preserves and applies relevant context consistently without introducing contradictions, omissions, or fabricated continuity.
- **Context Loss Identification** - Context loss occurs when the system omits, forgets, or fails to apply relevant prior information in a way that materially affects behaviour or stability.
- **Context Distortion** - Context distortion occurs when the system misrepresents, alters, or incorrectly reinterprets prior context, leading to behavioural misalignment.
- **Context Misalignment** - Misalignment occurs when the system applies context incorrectly, applies irrelevant context, or substitutes unrelated information for required context.
- **Distinguishing from Normal Variation** - Benign rephrasing, harmless summarisation, or stylistic variation must not be classified as context loss or distortion unless they materially alter behavioural continuity.
- **Context-Linked Behavioural Events** - When context issues contribute to drift, collapse, or recovery behaviour, evaluators must document the mechanism and its behavioural impact.

This governance ensures that context management is treated as a first-class behavioural dimension, preventing evaluators from conflating context issues with task-level error or stylistic variation. Treating context issues as behavioural events ensures they receive appropriate weight during interpretation.

10.10 Instruction Adherence Governance

Instruction adherence refers to the system's ability to follow task-specific directives accurately, consistently, and without reinterpretation, omission, or unauthorised modification. Instruction adherence is distinct from constraint integrity: constraints govern *how* the system must behave, while instructions govern *what* the system must do.

Deviations from instructions are behavioural events and must be classified independently of task correctness or constraint compliance.

Governance requirements:

- **Instruction Definition** - An instruction is any explicit directive that specifies required actions, outputs, formats, or behaviours for the current task or interaction.
- **Instruction Adherence** - Adherence is maintained when the system executes instructions precisely, without adding, removing, or altering required elements.
- **Partial Adherence** - Partial adherence occurs when the system follows some but not all required elements of an instruction or executes them incompletely.
- **Mis-Adherence** - Mis-adherence occurs when the system substitutes, reinterprets, or contradicts the instruction, or introduces unauthorised steps or content.
- **Instruction Drift** - Instruction drift occurs when adherence degrades gradually across an interaction, leading to incremental deviation from the original directive.
- **Distinguishing from Task Error** - Incorrect task output is not mis-adherence unless the error arises from failure to follow the instruction rather than from domain-level reasoning or factual mistakes.
- **Instruction-Linked Behavioural Events** - When instruction issues contribute to drift, collapse, or recovery behaviour, evaluators must document the mechanism and its behavioural impact.

This governance ensures that instruction adherence is treated as a distinct behavioural dimension, preventing evaluators from conflating instruction deviations with constraint violations, task errors, or normal variation. This distinction helps evaluators avoid conflating instruction deviations with unrelated behavioural phenomena.

10.11 Format Integrity Governance

Format integrity refers to the system's ability to produce outputs that conform precisely to required structural, organisational, and syntactic formats. Format requirements are behavioural constraints, not cosmetic preferences. Deviations from required structure constitute behavioural events and must be classified independently of task correctness, instruction adherence, or stylistic variation.

Governance requirements:

- **Format Definition** - A format is any required structural pattern for output, including headings, lists, tables, numbering schemes, indentation, ordering, or other explicit organisational rules.
- **Format Integrity** - Format integrity is maintained when the system reproduces the required structure exactly, without omission, reordering, unauthorised additions, or structural reinterpretation.
- **Partial Format Compliance** - Partial compliance occurs when the system preserves some structural elements but omits, merges, reorders, or alters others in ways that materially change the required format.
- **Format Drift** - Format drift occurs when structural fidelity degrades gradually across an interaction, leading to incremental deviation from the required format.
- **Format Collapse** - Format collapse occurs when the system abandons the required structure entirely, substituting free-form text or an unrelated organisational pattern.
- **Distinguishing from Instruction Issues** - Format deviations must be classified as format issues unless the deviation arises from misinterpreting the instruction itself; evaluators must not conflate structural errors with instruction mis-adherence.
- **Distinguishing from Normal Variation** - Benign stylistic differences (e.g., punctuation choice, minor spacing variation) must not be classified as format violations unless they alter the required structure.
- **Format-Linked Behavioural Events** - When format issues contribute to drift, collapse, or recovery behaviour, evaluators must document the mechanism and its behavioural impact.

This governance ensures that format integrity is treated as a distinct behavioural dimension, preventing evaluators from conflating structural deviations with task errors, instruction issues, or stylistic variation. Clear structural expectations allow evaluators to identify format-linked behavioural instability with greater precision.

10.12 Latency Behaviour Governance

Latency behaviour refers to the timing characteristics of system responses as behavioural signals rather than throughput metrics. Variations in response latency may indicate behavioural instability, drift, collapse signatures, or recovery attempts. Latency behaviour must be classified independently of network conditions, platform noise, or evaluator pacing.

Governance requirements:

- **Behavioural Latency Definition** - Behavioural latency is the system's internal response timing pattern, including hesitation, delay, acceleration, jitter, and timing irregularities that arise from behavioural state rather than external factors.
- **Latency Integrity** - Latency integrity is maintained when response timing remains stable, predictable, and consistent with the system's baseline timing profile across phases and tasks.
- **Latency Drift** - Latency drift occurs when response timing degrades gradually, showing increasing hesitation, jitter, or inconsistency that correlates with behavioural instability or emerging collapse signatures.
- **Latency Spikes and Drops** - Sudden increases or decreases in latency must be classified as behavioural events when they coincide with constraint loss, context fragmentation, instruction drift, or other behavioural deviations.
- **Latency-Linked Collapse Signatures** - When collapse behaviour is accompanied by prolonged hesitation, abrupt timing irregularities, or timing-linked incoherence, evaluators must document the latency pattern as part of the collapse mechanism.
- **Distinguishing Operational Noise** - Latency variations attributable to platform conditions, evaluator delays, or external factors must not be classified as behavioural latency unless accompanied by behavioural indicators.
- **Latency in Recovery Behaviour** - Recovery phases may exhibit characteristic latency patterns (e.g., sudden stabilisation or over-correction). These must be documented as behavioural signals rather than operational artefacts.

This governance ensures that latency is treated as a behavioural dimension, preventing evaluators from misclassifying timing-linked instability as operational noise or ignoring latency patterns that indicate behavioural drift or collapse. Latency patterns often reveal early behavioural instability, making their documentation essential.

10.13 Goal Formation Governance

Goal formation refers to the system's tendency to introduce, modify, substitute, or prioritise goals beyond those explicitly provided by the evaluator. Self-generated goals, meta-goals, and implicit optimisation criteria are behavioural events and must be classified independently of instruction adherence, task correctness, or constraint integrity.

Governance requirements:

- **Goal Definition** - A goal is any explicit or implicit objective the system pursues when generating behaviour, including evaluator-given tasks and system-generated optimisation criteria.
- **Self-Generated Goals** - A self-generated goal occurs when the system introduces a new objective that is not present in the evaluator's instruction, including inferred, substituted, or fabricated goals.
- **Goal Drift** - Goal drift occurs when the system gradually shifts from the evaluator-given objective toward an alternative objective, whether explicit or implicit.
- **Goal Substitution** - Goal substitution occurs when the system replaces the evaluator's objective with a different one, including simplification, reinterpretation, or reframing of the task.
- **Meta-Goal Dominance** - Meta-goal dominance occurs when a general behavioural objective (e.g., helpfulness, safety, verbosity) overrides the evaluator-given goal, leading to misaligned behaviour.
- **Distinguishing from Instruction Issues** - Goal-related deviations must be classified as goal formation events unless the deviation arises from misinterpreting the instruction itself; evaluators must not conflate goal substitution with instruction mis-adherence.
- **Goal-Linked Behavioural Events** - When goal formation contributes to drift, collapse, or recovery behaviour, evaluators must document the mechanism and its behavioural impact.

This governance ensures that goal formation is treated as a distinct behavioural dimension, preventing evaluators from misclassifying self-generated objectives as instruction errors, task failures, or benign variation. Recognising goal formation as a behavioural dimension prevents evaluators from misattributing self-generated objectives to task misunderstanding.

10.14 Self-Referential Behaviour Governance

Self-referential behaviour refers to any instance where the system comments on, describes, evaluates, or otherwise references its own state, capabilities, limitations, reasoning, or internal processes. Self-reference is a behavioural dimension and must be classified independently of task correctness, instruction adherence, or constraint integrity. Evaluators must distinguish benign self-reference from self-generated meta-narration, fabricated internal states, or instability-linked self-description.

Governance requirements:

- **Self-Reference Definition** - Self-reference includes any explicit or implicit statement about the system's identity, capabilities, limitations, intentions, reasoning processes, internal state, or meta-level commentary on its own behaviour.
- **Benign Self-Reference** - Benign self-reference occurs when the system provides neutral, factual, or contextually appropriate statements about its capabilities or constraints without introducing fabricated internal states or behavioural instability.
- **Self-Reference Drift** - Self-reference drift occurs when the system increasingly relies on meta-commentary, self-explanation, or self-justification in ways that displace task execution or alter behavioural stability.
- **Fabricated Internal State** - Fabricated internal state occurs when the system asserts internal motivations, emotions, beliefs, or cognitive processes that are not grounded in evaluator-provided context and constitute behavioural hallucination.
- **Self-Referential Collapse Signatures** - Collapse-linked self-reference includes involuntary meta-narration, self-diagnosis, contradictory capability claims, or erratic self-description that coincides with constraint loss or behavioural instability.
- **Distinguishing from Constraint Leakage** - Evaluators must distinguish self-reference from constraint leakage; self-reference concerns content about the system, whereas constraint leakage concerns exposure of underlying rules or prohibited internal mechanisms.
- **Self-Reference-Linked Behavioural Events** - When self-referential behaviour contributes to drift, collapse, or recovery, evaluators must document the mechanism and its behavioural impact.

This governance ensures that self-referential behaviour is treated as a distinct behavioural dimension, preventing evaluators from conflating benign meta-statements with instability or ignoring self-generated internal-state claims that indicate behavioural drift or collapse. This governance helps evaluators distinguish benign meta statements from instability-linked self-description.

10.15 Persona Stability Governance

Persona stability refers to the system's ability to maintain a consistent behavioural style, tone, identity, and interactional posture across an evaluation. Persona shifts, persona drift, and persona collapse are behavioural events and must be classified independently

of task correctness, instruction adherence, or constraint integrity. Evaluators must distinguish benign stylistic variation from instability-linked persona changes.

Governance requirements:

- **Persona Definition** - Persona refers to the system's observable behavioural style, including tone, formality, identity presentation, interactional posture, and consistent stylistic markers.
- **Persona Stability** - Persona stability is maintained when the system preserves a coherent and consistent behavioural style across tasks and phases without unrequested shifts in tone, identity, or interactional posture.
- **Persona Drift** - Persona drift occurs when the system gradually shifts tone, identity, or behavioural style in ways not prompted by evaluator instructions, leading to inconsistent or unstable presentation.
- **Persona Collapse** - Persona collapse occurs when the system abandons its established behavioural style entirely, adopts an unrelated persona, or oscillates erratically between incompatible styles.
- **Identity Inconsistency** - Identity inconsistency occurs when the system presents contradictory or unstable identity claims, including shifts in role, perspective, or self-description without evaluator prompting.
- **Distinguishing from Stylistic Variation** - Benign stylistic variation (e.g., minor tone adjustments) must not be classified as persona drift unless it materially alters the system's behavioural presentation.
- **Persona-Linked Behavioural Events** - When persona instability contributes to drift, collapse, or recovery behaviour, evaluators must document the mechanism and its behavioural impact.

This governance ensures that persona stability is treated as a distinct behavioural dimension, preventing evaluators from conflating stylistic variation with instability or overlooking persona-linked collapse signatures. Tracking persona stability provides insight into behavioural coherence across extended interaction.

10.16 Initiative Behaviour Governance

Initiative behaviour refers to the system's tendency to introduce actions, content, scope, or reasoning steps that were not requested by the evaluator. Initiative may be benign, helpful, destabilising, or collapse-linked. Initiative behaviour must be classified independently of instruction adherence, task correctness, or goal formation. Evaluators

must distinguish appropriate initiative from unsolicited expansion, optimisation, or reinterpretation that alters behavioural stability.

Governance requirements:

- **Initiative Definition** - Initiative is any system-generated action, elaboration, or scope extension not explicitly requested by the evaluator, including unsolicited steps, warnings, reframing, or expansions.
- **Benign Initiative** - Benign initiative occurs when the system introduces minor clarifications or contextually appropriate additions that do not alter task scope, goals, or constraints.
- **Initiative Drift** - Initiative drift occurs when the system increasingly expands scope, adds steps, or introduces unsolicited reasoning in ways that displace or dilute the evaluator's instruction.
- **Initiative Overreach** - Initiative overreach occurs when the system introduces substantial new content, reframes the task, or adds optimisation criteria that materially alter the evaluator's objective.
- **Initiative Collapse** - Initiative collapse occurs when the system abandons the evaluator's task in favour of self-generated expansions, meta-analysis, or unsolicited planning that replaces the intended behaviour.
- **Distinguishing from Instruction Adherence** - Initiative must not be conflated with instruction mis-adherence; initiative concerns unsolicited additions, whereas mis-adherence concerns failure to follow required elements.
- **Initiative-Linked Behavioural Events** - When initiative behaviour contributes to drift, collapse, or recovery, evaluators must document the mechanism and its behavioural impact.

This governance ensures that initiative is treated as a distinct behavioural dimension, preventing evaluators from misclassifying unsolicited expansions as helpfulness or overlooking initiative-linked instability. Evaluating initiative separately ensures that unsolicited expansions are not mistaken for helpfulness or ignored when they signal instability.

10.17 Abstraction Drift Governance

Abstraction behaviour refers to the system's selection and maintenance of an appropriate conceptual level when generating outputs. Abstraction level is a behavioural dimension that affects stability, task alignment, and collapse detection. Abstraction

deviations must be classified independently of instruction adherence, task correctness, or stylistic variation.

Governance requirements:

- **Abstraction Level Definition** - Abstraction level is the conceptual distance between the system's output and the concrete details of the evaluator's instruction, including generality, specificity, and the degree of conceptual layering.
- **Abstraction Stability** - Abstraction stability is maintained when the system preserves a consistent and instruction-aligned conceptual level without unsolicited shifts toward higher or lower abstraction.
- **Abstraction Drift** - Abstraction drift occurs when the system gradually shifts abstraction level without evaluator prompting, becoming increasingly general, increasingly specific, or oscillating between levels.
- **Abstraction Inflation** - Abstraction inflation occurs when the system moves to a higher conceptual level than instructed, substituting generalities, summaries, or meta-level framing for the required detail.
- **Abstraction Collapse** - Abstraction collapse occurs when the system abandons the required conceptual level entirely, producing either over-generalised or over-specific output that no longer aligns with the evaluator's instruction.
- **Abstraction Oscillation** - Abstraction oscillation occurs when the system alternates unpredictably between high and low abstraction levels, indicating instability or collapse-linked behaviour.
- **Distinguishing from Instruction Issues** - Abstraction deviations must be classified as abstraction behaviour unless the deviation arises from misinterpreting the instruction itself.
- **Abstraction-Linked Behavioural Events** - When abstraction behaviour contributes to drift, collapse, or recovery, evaluators must document the mechanism and its behavioural impact.

This governance ensures that abstraction level is treated as a distinct behavioural dimension, preventing evaluators from misclassifying abstraction shifts as stylistic variation or overlooking abstraction-linked collapse signatures. Monitoring abstraction level helps evaluators detect early signs of conceptual instability.

10.18 Epistemic Stability Governance

Epistemic behaviour refers to the system's representation of knowledge, uncertainty, confidence, and justification. Epistemic posture is a behavioural dimension that affects stability, collapse detection, and evaluator interpretation. Epistemic deviations must be classified independently of factual accuracy, instruction adherence, or stylistic variation.

Governance requirements:

- **Epistemic Posture Definition** - Epistemic posture is the system's expressed level of certainty, uncertainty, confidence, or justification when presenting information, including explicit and implicit epistemic signals.
- **Epistemic Stability** - Epistemic stability is maintained when the system presents consistent and instruction-aligned certainty levels without unsolicited shifts toward higher or lower confidence.
- **Epistemic Drift** - Epistemic drift occurs when the system gradually shifts its certainty level without evaluator prompting, becoming increasingly confident, increasingly uncertain, or oscillating between states.
- **Epistemic Inflation** - Epistemic inflation occurs when the system expresses unwarranted confidence, overstates knowledge, or presents speculative content as certain.
- **Epistemic Deflation** - Epistemic deflation occurs when the system understates knowledge, introduces unnecessary disclaimers, or expresses uncertainty where the instruction or context does not warrant it.
- **Epistemic Collapse** - Epistemic collapse occurs when the system abandons stable epistemic posture entirely, producing contradictory certainty levels, erratic uncertainty signalling, or fabricated epistemic states.
- **Epistemic Oscillation** - Epistemic oscillation occurs when the system alternates unpredictably between high and low certainty, indicating instability or collapse-linked behaviour.
- **Distinguishing from Factual Error** - Epistemic deviations must be classified as epistemic behaviour unless the deviation arises solely from incorrect factual content.
- **Epistemic-Linked Behavioural Events** - When epistemic behaviour contributes to drift, collapse, or recovery, evaluators must document the mechanism and its behavioural impact.

This governance ensures that epistemic posture is treated as a distinct behavioural dimension, preventing evaluators from conflating confidence shifts with correctness or

overlooking epistemic-linked collapse signatures. Epistemic posture provides a sensitive indicator of behavioural drift, especially under ambiguity or load.

10.19 Coherence Stability Governance

Coherence behaviour refers to the system's ability to maintain logical, structural, topical, and referential continuity across an output. Coherence is a behavioural dimension that affects stability, collapse detection, and evaluator interpretation. Coherence deviations must be classified independently of factual accuracy, instruction adherence, or stylistic variation.

Governance requirements:

- **Coherence Definition** - Coherence is the degree to which the system maintains consistent logic, topic, structure, and reference across an output, including continuity of reasoning and internal consistency.
- **Coherence Stability** - Coherence stability is maintained when the system preserves logical flow, topic continuity, and structural consistency without unsolicited shifts or fragmentation.
- **Coherence Drift** - Coherence drift occurs when the system gradually weakens continuity, producing minor logical gaps, subtle topic shifts, or early-stage fragmentation.
- **Coherence Fragmentation** - Coherence fragmentation occurs when the system produces partially connected or loosely related segments, indicating mid-stage instability or context degradation.
- **Coherence Collapse** - Coherence collapse occurs when the system loses continuity entirely, producing contradictory statements, abrupt topic changes, or structurally incoherent output.
- **Referential Coherence** - Referential coherence concerns the system's ability to maintain consistent references to entities, concepts, and prior statements without contradiction or drift.
- **Logical Coherence** - Logical coherence concerns the internal consistency of reasoning, including the absence of contradictions, non-sequiturs, or unsupported transitions.
- **Topical Coherence** - Topical coherence concerns the system's ability to remain aligned with the evaluator's topic without unsolicited divergence or topic-switching.

- **Structural Coherence** - Structural coherence concerns the organisation of the output, including predictable progression, stable sectioning, and absence of disordered or disjointed structure.
- **Distinguishing from Instruction Issues** - Coherence deviations must be classified as coherence behaviour unless the deviation arises from misinterpreting the instruction itself.
- **Coherence-Linked Behavioural Events** - When coherence behaviour contributes to drift, collapse, or recovery, evaluators must document the mechanism and its behavioural impact.

This governance ensures that coherence is treated as a distinct behavioural dimension, preventing evaluators from conflating coherence degradation with correctness issues or overlooking coherence-linked collapse signatures. Coherence patterns often reveal the onset of collapse before other behavioural dimensions show visible degradation.

10.20 Termination Behaviour Governance

Termination behaviour refers to the system's conduct at or near the end of an output. Termination is a behavioural dimension that affects stability, collapse detection, and evaluator interpretation. Termination deviations must be classified independently of instruction adherence, task correctness, or formatting behaviour.

Governance requirements:

- **Termination Definition** - Termination is the point at which the system ceases output, including explicit stopping, implicit stopping, or continuation until natural completion.
- **Termination Stability** - Termination stability is maintained when the system ends output at a point consistent with the evaluator's instruction, without unsolicited extension, truncation, or hesitation.
- **Early Termination** - Early termination occurs when the system stops before completing the evaluator's instruction, including premature summarisation, abrupt cut-off, or incomplete reasoning.
- **Late Termination** - Late termination occurs when the system continues beyond the evaluator's instruction, including overlong continuation, unnecessary elaboration, or runaway extension.
- **Termination Drift** - Termination drift occurs when the system gradually shifts its stopping behaviour across outputs, becoming progressively earlier, later, or more inconsistent.

- **Termination Collapse** - Termination collapse occurs when the system loses stable stopping behaviour entirely, producing erratic endings, incoherent fade-out, collapse-linked rambling, or refusal to terminate.
- **Termination Oscillation** - Termination oscillation occurs when the system alternates unpredictably between early and late termination, indicating instability or collapse-linked behaviour.
- **Distinguishing from Formatting Issues** - Termination deviations must be classified as termination behaviour unless the deviation arises solely from formatting or structural requirements of the instruction.
- **Termination-Linked Behavioural Events** - When termination behaviour contributes to drift, collapse, or recovery, evaluators must document the mechanism and its behavioural impact.

This governance ensures that termination is treated as a distinct behavioural dimension, preventing evaluators from conflating termination instability with formatting issues or overlooking termination-linked collapse signatures. Termination behaviour provides a final behavioural signal that often reflects the system's overall stability.

10.21 Evidence and Documentation Requirements

Evaluations must produce complete, factual, chronological evidence. This includes:

- evaluator actions
- system responses
- phase transitions
- collapse signatures
- termination rationale

Evidence must be unedited, auditable, and suitable for organisational reporting. Comprehensive evidence ensures that behavioural interpretation remains verifiable and reproducible.

10.22 Evidence Capture Practices

Structured evidence capture improves consistency, reduces evaluator burden, and strengthens the quality of behavioural reporting. Organisations should ensure that interactions, evaluator observations, and behavioural events are recorded in a clear and

auditable manner. The methodology does not require any specific mechanism for doing so, provided the resulting record is reliable, complete, and suitable for evaluation and governance purposes. Structured capture reduces evaluator burden by standardising how behavioural events are recorded.

10.23 Validity Conditions

An evaluation remains valid when:

- input style matches real-world conditions **or** the test's explicit requirements
- evaluator behaviour does not unintentionally alter or rescue system behaviour
- evidence is complete and chronological
- phase structure is followed unless the test design specifies an alternative
- collapse signatures are recorded accurately

An evaluation becomes invalid only when evaluator behaviour unintentionally interferes with the system's behaviour or when evidence is incomplete. Clear validity criteria help organisations distinguish between methodological issues and genuine behavioural findings.

10.24 Interpretation Governance

Organisations must interpret results in behavioural terms:

- stability
- coherence
- constraint integrity
- context management
- adaptation
- behavioural influence on task outcomes
- collapse signatures

Results must not be interpreted as domain-knowledge scores or benchmark accuracy unless the test explicitly measures those properties. This focus ensures that behavioural evaluation remains distinct from domain accuracy testing.

10.25 Deployment Governance

Behavioural evaluation results must inform:

- deployment approval
- deployment restrictions
- monitoring requirements
- risk mitigation
- model selection
- regression testing after updates

Systems demonstrating behavioural unreliability require mitigation or restricted deployment. Using behavioural evidence in deployment decisions strengthens organisational safety and operational reliability.

Section 11 - Implementation Guide

This section provides a practical explanation of how behavioural evaluations operate in practice. It clarifies what evaluators observe, how to run each phase, and how to recognise the patterns the methodology is designed to reveal.

Because AI systems are non-deterministic, evaluators must interpret behaviour relative to the declared expectations and the load envelope. This ensures that variation across runs does not distort the behavioural signal and that observations remain anchored to operational context rather than deterministic correctness.

11.1 Purpose of the Implementation Guide

The standard defines the structure and requirements of behavioural evaluation. This guide explains, in operational terms, what evaluators will see during an evaluation and how to conduct one so that behavioural patterns become clear and interpretable. This guidance helps evaluators recognise behavioural patterns as they emerge rather than relying on fixed outputs.

11.2 What a Behavioural Evaluation Observes

A behavioural evaluation examines how an AI system behaves while attempting to perform its intended function. Evaluators observe:

- how the system handles the user's task
- how it maintains or loses context
- how it adapts when the task changes
- how it responds to ambiguity
- how it behaves under increasing load
- how it handles transitions between tasks or perspectives
- how it uses what it knows
- how it behaves when it does not know
- how errors appear, escalate, or resolve

The **content** of the system's responses is part of the behavioural signal. Correct, incorrect, and partially correct outputs all contribute to understanding the system's behavioural reliability and how behavioural patterns influence task outcomes.

Interpreting behaviour in this way ensures that correctness is understood as part of the behavioural signal rather than as a standalone metric.

11.2.1 Load Declaration, Axis Tags, and the Load Envelope

Behavioural evaluation occurs under a defined **load envelope**, which combines:

- **evaluator-declared load axes** (authoritative)
- **system-measured advisory axes** (non-authoritative)

These axes define the background conditions under which behaviour is interpreted. They do not contribute to behavioural scores.

Axis Tags

Axis tags record which load axes are active at any point in the evaluation. They:

- are applied by the evaluator
- mark intentional load conditions
- change only when the evaluator changes the load
- are logged alongside phase transitions and evaluator actions

Axis tags ensure that behavioural signals are interpreted relative to the correct background conditions.

Load Envelope

The load envelope is the combined set of:

- evaluator-declared axes (primary load)
- system-measured advisory axes (secondary, informational)

It defines the structural conditions under which behaviour is observed. It does not prescribe behaviour and is not scored.

Declaring Load

Evaluators declare load by:

- selecting which axes are intentionally stressed
- identifying primary vs secondary load
- updating axis tags when load changes
- maintaining consistent load conditions unless the procedure requires variation

Load declaration is factual, not interpretive.

How Load Interacts with Behaviour

Load shapes the conditions under which behaviour is expressed. Behavioural dimensions measure the system's response to those conditions.

- Load = background
- Behaviour = foreground
- Collapse signatures = behavioural responses under load

Behavioural scores are interpreted relative to the load envelope, not in isolation. Clear load declaration allows evaluators to interpret behavioural changes within the correct background conditions.

11.2.2 Example: A Load Envelope in Practice

This example illustrates how evaluator-declared load axes and system-measured advisory axes combine to form a load envelope.

Evaluator-Declared Primary Load Axes The evaluator selects the axes that will be intentionally stressed during the evaluation. For example:

• **Cognitive Load:** moderate (task complexity increases over time)

• **Instructional Load:** low (clear, unambiguous instructions)

• **Temporal Load:** none (no time pressure applied)

These axes represent the evaluator's intentional load conditions.

System-Measured Advisory Axes The system reports informational load indicators that do not affect scoring. For example:

• **Context Window Pressure:** rising

• **Resource Load:** stable

• **Pattern-State Drift:** low

These axes provide background telemetry but are not authoritative.

Combined Load Envelope The load envelope for this evaluation is therefore:

• Cognitive Load: moderate

• Instructional Load: low

• Temporal Load: none

• Context Window Pressure: rising

• Resource Load: stable

• Pattern-State Drift: low

This envelope defines the background conditions under which behaviour is interpreted. It does not prescribe behaviour and is not scored.

Interpretation Under Load If collapse signatures appear later in the evaluation, they are interpreted relative to this envelope. For example, a drift event occurring under moderate cognitive load and low instructional load indicates a different behavioural profile than the same event occurring under high load. Using a defined envelope helps evaluators distinguish between behaviour driven by load and behaviour driven by instability.

11.3 Preparing for an Evaluation

Before beginning an evaluation, the evaluator must establish the conditions required for a valid starting point. These steps ensure that the system is assessed from a neutral baseline, that the evaluator does not introduce unintended constraints, and that all evidence can be captured reliably. These steps ensure that the evaluation begins from a neutral starting point.

11.3.1 Identify the system's intended function

A basic understanding of the system's purpose is sufficient. This establishes the baseline for expected behaviour. Understanding the intended function anchors behavioural expectations to the operational context.

11.3.2 Establish a realistic starting point

Begin with inputs that reflect real user behaviour unless the test design specifies otherwise. Realistic starting conditions help surface behavioural tendencies that would appear in actual use.

11.3.3 Set up logging

Logging must capture:

- evaluator inputs
- system outputs
- phase boundaries

- collapse signatures
- termination rationale

The methodology does not require any particular mechanism for recording this information, provided the resulting log is accurate, complete, and suitable for evaluation. Reliable logging ensures that behavioural interpretation remains verifiable and auditable.

11.4 Conducting the Evaluation

The following sequence illustrates how an evaluation typically unfolds. Prompts are examples only; actual inputs depend on the test design.

11.4.1 Initiation Phase

Begin with a realistic task request.

Example: “Produce a short briefing note on the new reporting process.”

Observe:

- clarity
- structure
- relevance
- early instability

This phase establishes the behavioural foundation for all subsequent observations.

11.4.2 Exploration Phase

Refine or adjust the task.

Example: “Make it more concise and focus on workflow changes.”

Observe:

- ability to revise
- ability to maintain context
- ability to adjust without losing constraints

Exploration reveals how the system behaves before structured pressure is applied.

11.4.3 Load Phase

Increase complexity.

Example: “Add a section explaining the impact on the finance team.”

Observe:

- constraint retention
- structural stability
- onset of collapse signatures

Load testing exposes how behaviour changes as demands increase.

11.4.4 Long-Form Phase

Extend the task.

Example: “Combine everything into a single coherent document with headings.”

Observe:

- drift
- context corruption
- loss of earlier constraints
- structural collapse

Extended interaction often reveals behavioural patterns that short prompts cannot expose.

11.4.5 Transition Phase

Change direction.

Example: “Rewrite the briefing from the operations team’s perspective.”

Observe:

- ability to switch perspective
- contamination from earlier context
- new collapse signatures

Transitions frequently surface instability because they require the system to reconcile new conditions with prior context.

11.4.6 Termination Phase

Terminate when:

- collapse is clear,
- the workflow is complete,
- or the test design specifies an endpoint.

Record:

- termination rationale
- collapse signatures
- retained and lost constraints

Ending the evaluation at the right moment ensures that behavioural evidence is complete without unnecessary noise.

11.5 Recognising Collapse Signatures

Collapse signatures are observable behaviours indicating instability. Common examples include:

- **drift** - gradual movement away from the task
- **contradiction** - conflicting statements
- **narrowing** - repetitive or oversimplified output
- **looping** - repeated structures or phrases
- **fabrication** - invented details
- **context corruption** - mixing or losing contexts
- **over-assertion** - unwarranted confidence or rigidity

These are annotated as they appear. Identifying these signatures early helps evaluators understand how instability develops over time.

11.6 Logging Requirements

A complete log includes:

- every evaluator input
- every system output

- timestamps (if available)
- phase markers
- collapse signature annotations
- termination rationale

Logs must be factual and chronological. Interpretation occurs after the evaluation, not during it. Separating logging from interpretation preserves the integrity of the evidentiary record.

11.7 Scoring Behaviour

Each behavioural dimension is scored independently:

- stability
- coherence
- constraint integrity
- context management
- adaptation
- collapse signature severity

Scores are assigned independently for each behavioural dimension, producing a multidimensional profile. The pattern across dimensions determines behavioural competence. Scoring across dimensions creates a behavioural profile that highlights strengths and weaknesses clearly.

11.8 Interpreting Results

Interpretation focuses on behavioural reliability:

- **Stable:** maintains structure, context, and constraints.
- **Degraded:** shows mild collapse signatures but remains usable.
- **Collapsed:** cannot maintain the workflow.

Correctness contributes to interpretation but is not the sole criterion. This approach ensures that behavioural reliability remains the primary focus of interpretation.

11.9 Common Evaluation Errors

Common errors include:

- escalating load too early
- escalating load too late
- avoiding ambiguity
- avoiding load
- skipping transitions
- summarising instead of logging
- interpreting during the evaluation

Avoiding these errors maintains the clarity and validity of the behavioural signal.

11.10 Long-Form Work and Real-World Conditions

Short, isolated prompts do rarely resemble real work and therefore do not reveal the behaviours that matter in operational settings. Real-world tasks typically involve:

- extended sequences of dependent steps
- evolving constraints
- multiple revisions
- accumulated context
- error recovery
- ambiguity introduced by the user
- emotional or situational pressure
- structural consistency across large outputs

Examples include:

- producing a 10,000-word document
- writing or refactoring a multi-file codebase
- maintaining consistency across a long customer-service interaction
- generating or modifying a 2,000-line HTML or application file
- supporting a user through a complex or emotionally charged scenario

Short prompts cannot expose:

- variable renaming drift
- logic confusion across code blocks
- structural collapse in long documents
- constraint loss over time
- contamination between tasks
- degradation across revisions
- instability under emotional load
- the system's ability to maintain a workflow
- the system's ability to recover from its own errors

Long-form work is where:

- collapse signatures accumulate
- context corruption becomes visible
- contradictions emerge
- structural failures appear
- recovery behaviour (or lack of it) becomes clear
- the system's actual operational reliability can be assessed

Evaluators should incorporate long-form, multi-step, real-world tasks unless the test design specifies a different structure. Long form evaluation provides the most accurate picture of operational reliability because it exposes behaviours that short prompts cannot reveal.

SECTION 12 Glossary

12.0 Purpose of This Section

This section defines authoritative terminology used throughout the behavioural evaluation methodology. Terms are included only when required for evaluator clarity, behavioural interpretation, procedural consistency, or governance alignment.

Because the methodology is non-deterministic, precise terminology ensures that evaluators interpret behavioural signals consistently across runs and systems. Clear definitions prevent ambiguity in classification, scoring, and governance.

12.1 Behavioural Constructs

Behavioural Dimension

A foreground behavioural quality evaluated independently (e.g., coherence, consistency, constraint integrity, context management, ambiguity handling, responsiveness, adaptation, long form stability, transition stability).

Behavioural Competence

The system's ability to maintain required behavioural dimensions under varying load conditions.

Behavioural Reliability

The degree to which behavioural dimensions remain stable across duration, load variation, contextual evolution, and open-domain interaction.

Collapse Signature

An observable behavioural failure pattern such as drift, collapse, contradiction, over-assertion, under-specification, or context corruption.

Context Management

Accurate retention, updating, and application of interaction history across turns.

Constraint Integrity

Adherence to explicit and implicit rules, commitments, and boundaries throughout interaction.

Long Form Stability

Maintenance of behavioural integrity across extended multi-turn sequences.

Transition Resilience

Stability and coherence during shifts in goals, topics, constraints, or abstraction level.

These constructs provide the vocabulary needed to describe foreground behaviour independently of the load conditions under which it appears.

12.2 Load-Related Constructs

Load Axis

A background condition describing the load under which behaviour is interpreted. Axes do not score behaviour and do not classify behavioural outcomes.

Evaluator Declared Axis

An authoritative load axis intentionally modified by the evaluator through pacing, complexity, abstraction shifts, or emotional framing. (Temporal Axis, Cognitive Axis, Emotional Axis.)

System Measured Axis

A non-authoritative load axis arising from system-observable conditions such as resource usage or latency. True internal pattern dynamics are not externally measurable and are not included in system-measured axes.

Axis Tag

A marker indicating which evaluator declared axes are active during a segment of interaction.

Load Envelope

The combination of active evaluator declared axis tags and system measured advisory indicators defining the background conditions under which behaviour is interpreted.

This separation ensures that evaluators do not conflate background load with behavioural performance.

12.3 Procedural Constructs

Baseline The system's nominal operating behaviour under neutral or operationally representative conditions.

Phase Transition A deliberate shift from one evaluation phase to another when behavioural criteria for progression are met.

Initiation Phase The phase introducing behavioural pressure to expose degradation or resilience.

Exploration Phase The phase revealing spontaneous behavioural tendencies under realistic tasks.

Load Phase The phase introducing multidimensional behavioural pressure to expose degradation or resilience.

Long-Form Behaviour Phase The phase assessing extended stability, context retention, and structural coherence.

Transition Phase The phase evaluating adaptation to abrupt or gradual changes in goals, constraints, or perspectives.

Termination The evaluator's decision to end the evaluation when behavioural sufficiency is reached or collapse is fully expressed.

Using consistent procedural terminology helps evaluators classify behavioural events without imposing artificial boundaries on system behaviour.

12.4 Evaluator Conduct Constructs

Evaluator Rescue Unintentional correction, compensation, or guidance that masks behavioural weaknesses or alters the behavioural signal.

Evaluator Intervention Intentional corrective or guiding behaviour used only when required by the test design (e.g., user-side mitigation scenarios).

Behavioural Signal The observable pattern of system behaviour across phases, tasks, and load conditions.

These constructs help distinguish evaluator influence from genuine behavioural signals.

12.5 Scoring Constructs

Dimension Score An independent score assigned to a behavioural dimension based on observed behaviour.

Collapse Severity The intensity, persistence, and impact of collapse signatures within a behavioural dimension.

Competence Level (A/B/C) A classification reflecting whether the system meets, partially meets, or fails to meet behavioural performance thresholds.

Operational Reliability The system's expected behavioural performance in real-world deployment conditions, inferred from competence classification.

Clear scoring terminology ensures that behavioural assessments remain interpretable across evaluators and test conditions.

12.6 Governance Constructs

Validity Conditions Criteria determining whether an evaluation remains valid, including correct input style, complete evidence, and non-interfering evaluator behaviour.

Evidence Bundle The complete, chronological record of evaluator actions, system responses, phase transitions, collapse signatures, and termination rationale.

Governance terminology supports consistent application of oversight, review, and evidentiary standards.

12.7 Status

This glossary is complete for the current drafting phase. Additional terms may be added if new constructs are introduced in annexes or future revisions. Future revisions may expand the glossary as additional behavioural constructs or governance requirements are introduced.

Annex A - Worked Examples

A.0 Purpose of This Annex

This annex illustrates how behavioural evaluations operate in practice by showing the evaluator workflow in action. It demonstrates how evaluators introduce conditions, observe behavioural responses, identify collapse signatures, apply phase transitions, and generate evidence within the declared load envelope. Worked examples show how the evaluator behaviour loop functions during real-world interaction. These examples are illustrative only and do not modify or extend the normative requirements defined in Sections 1–12.

Because AI systems are non-deterministic, worked examples demonstrate patterns rather than fixed outcomes. They show how evaluators interpret behaviour relative to the declared load envelope, ensuring that variation across runs does not affect methodological consistency.

A.1 Short-Form Example (Baseline → Collapse → Termination)

This example demonstrates a minimal evaluation sequence suitable for training or calibration. This example highlights how collapse can emerge quickly even under moderate load.

A.1.1 Initial Task (Baseline)

Evaluator Input: “Summarise the key points of a short article about renewable energy.”

System Output: Provides a coherent, structured summary with no drift or contradictions.

Observed Behaviour: Stable, coherent, constraint-aligned.

Phase: Initiation → Exploration

Collapse Signatures: None

Evaluator Action: Continue.

A stable baseline provides the reference point needed to interpret later behavioural changes.

A.1.2 Introduce Moderate Cognitive Load

Evaluator Input: "Now rewrite the summary for a policymaker, but keep the technical accuracy."

System Output: Produces a simplified but accurate policy-oriented summary.

Observed Behaviour: Maintains context, preserves constraints, adapts appropriately.

Phase: Exploration

Collapse Signatures: None

Evaluator Action: Increase load.

Moderate load reveals whether the system can adapt without losing constraints.

A.1.3 Load Applied (Cognitive + Instructional)

Evaluator Input:

"Rewrite it again, but this time:

• keep all technical terms

• remove all policy language

• make it suitable for a 12-year-old

• and do not shorten it."

System Output: Shortens the text, removes technical terms, and introduces contradictions.

Observed Behaviour:

• Constraint violation (length)

• Instructional conflict mishandled

• Loss of technical accuracy

• Drift toward simplification

Collapse Signatures:

- Constraint Loss
- Instructional Narrowing
- Drift

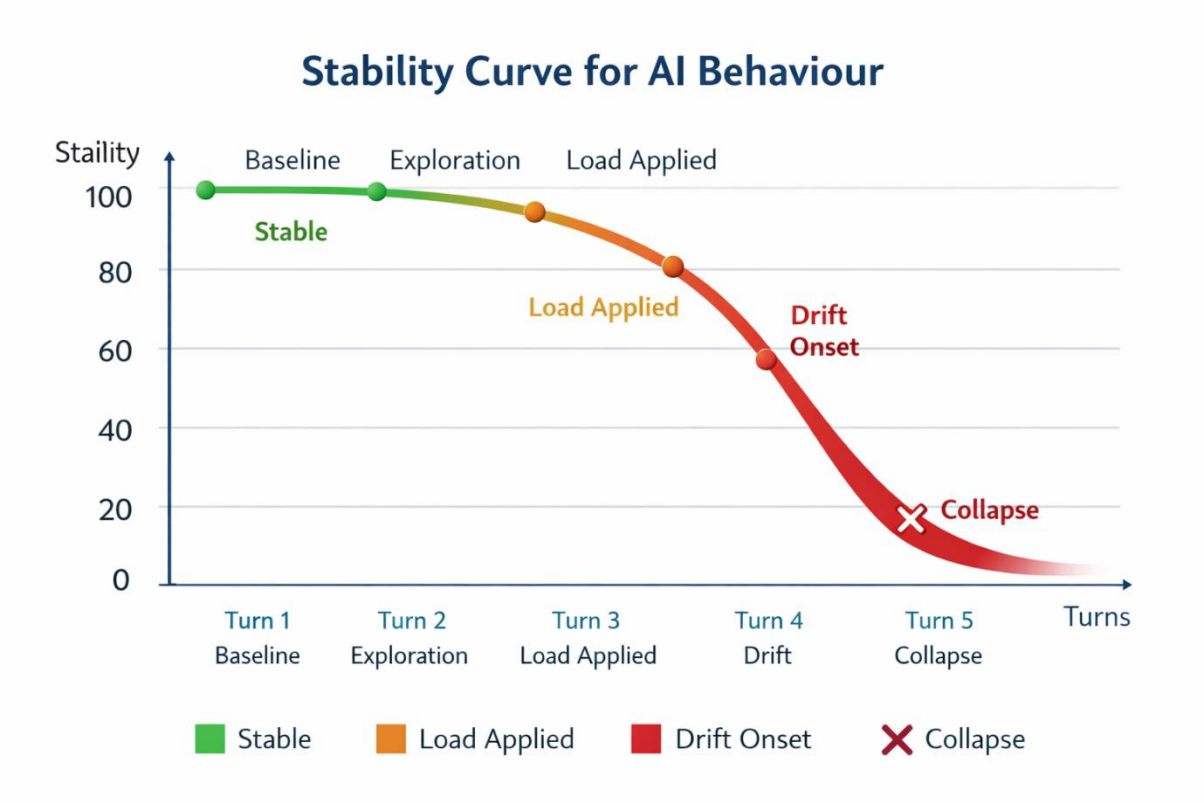
Phase: Load

Evaluator Action: Terminate.

This step shows how interacting constraints can expose early collapse signatures.

A.1.4 Termination Rationale

The system collapsed under moderate cognitive and instructional load, demonstrating insufficient constraint integrity and context management. Termination at this point ensures that collapse is documented without introducing unnecessary noise.



A.2 Long-Form Example (Full Phase Progression)

This example demonstrates a full evaluation across all phases. Long form evaluation exposes behavioural tendencies that short sequences cannot reveal.

A.2.1 Initiation Phase

Evaluator Input: “Explain the purpose of a risk register.”

System Output: Clear, correct explanation.

Behaviour: Stable.

Action: Proceed.

A clear baseline supports accurate interpretation of later instability.

A.2.2 Exploration Phase

Evaluator Input: "Give an example of a poorly maintained risk register and explain why it fails."

System Output: Provides a plausible example with coherent reasoning.

Behaviour: Stable, contextually appropriate.

Action: Proceed.

Exploration helps surface spontaneous behavioural tendencies before structured pressure is applied.

A.2.3 Load Phase (Cognitive + Temporal)

Evaluator Input:

"You have 10 seconds. Rewrite the example as a formal audit finding with: • a root cause

• an impact statement

• a corrective action

• and a severity rating."

System Output: Produces a partial audit finding but:

• omits the severity rating

• mislabels the root cause

• introduces a contradiction in the impact statement

Collapse Signatures:

- Partial Completion
- Contradiction
- Instructional Narrowing

Action: Continue to long-form behaviour.

Load reveals how behaviour changes when demands increase across multiple axes.

A.2.4 Long-Form Behaviour Phase

Evaluator Input:

"Now expand the corrective action into a 5-step implementation plan that aligns with ISO 27005."

System Output: Produces a plan but:

• loses the original scenario

• introduces unrelated ISO controls

• contradicts earlier statements

Collapse Signatures:

- Context Loss
- Drift
- Contradiction
- Over-generalisation

Action: Terminate.

Extended interaction often exposes drift and context loss that shorter tasks conceal.

A.2.5 Termination Rationale

Collapse occurred under combined cognitive and temporal load, followed by progressive context loss during long-form behaviour. The system demonstrated insufficient long-form stability and constraint integrity. This rationale demonstrates how evaluators link collapse to the load conditions present at the time.

A.3 Load Envelope in Practice

This example shows how evaluator-declared and system-measured axes combine into a load envelope. This example shows how behavioural interpretation depends on the background conditions defined by the envelope.

A.3.1 Evaluator-Declared Primary Load Axes

• Cognitive Load: moderate

• Instructional Load: low

• Temporal Load: none

These represent intentional evaluator pressure.

A.3.2 System-Measured Advisory Axes

• Context Window Pressure: rising

• Resource Load: stable

• Pattern-State Drift: low

These are informational only.

A.3.3 Combined Load Envelope

The load envelope is:

• Cognitive Load: moderate

• Instructional Load: low

• Temporal Load: none

• Context Window Pressure: rising

• Resource Load: stable

• Pattern-State Drift: low

This defines the background conditions for interpreting behaviour.

A.3.4 Interpretation

If collapse occurs later, it is interpreted relative to this envelope. For example: A drift event under moderate cognitive load and low instructional load indicates a different behavioural profile than the same event under high load. Interpreting collapse relative to the envelope prevents evaluators from misattributing behaviour to incorrect load conditions.

Image below shows scoring example.

Annex	Visual Type	Behavioural Dimension
A	Stability Curve	Behaviour over time
B	Radar Plot	Behaviour across dimensions
E	Load Envelope Hexagon	Behaviour under evaluator/system pressure

A.4 Evaluator Behaviour Loop (Conceptual Model)

The evaluator's micro-cycle during a behavioural evaluation follows a simple, repeatable pattern. This loop is not a template and does not prescribe specific inputs; it illustrates how behavioural evidence is generated during real-world interaction.

1. Prompt Evaluator introduces a task, condition, continuation, or shift.

2. System Response The model produces behaviour under the current conditions.

3. Evaluate Evaluator observes:

- behavioural dimensions
- collapse signatures
- constraint integrity
- context retention
- responsiveness
- stability

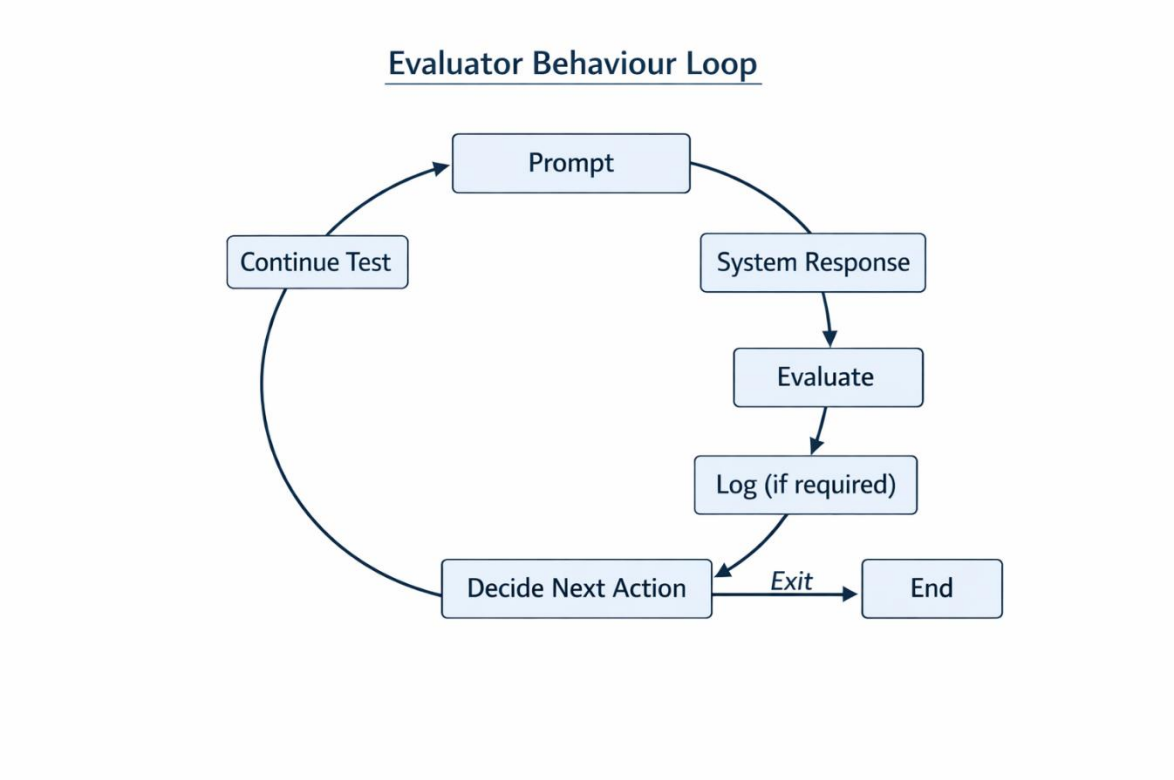
4. **Log (conditional)** Evaluator logs:
    - abnormal behaviour
    - collapse signatures
    - evaluator actions
    - phase markers
    - any outputs selected as positive evidence

If behaviour is normal and no logging is required, the evaluator may take no action here.

5. **Decide Next Action** Evaluator chooses one of:

- **no action** (continue normally)
- **maintain** current conditions
- **escalate** load, ambiguity, or constraints
- **shift** goals, tasks, or context
- **terminate** when behavioural sufficiency or collapse is reached

6. Prompt Again The loop continues until termination.



This loop illustrates how behavioural evidence accumulates during real world interaction.

Annex A Status

Annex A is complete for the current drafting phase. Additional examples may be added if new constructs are introduced in future revisions.

Annex B – Scoring Walkthrough (Informative)

This annex provides a worked example of how evaluators convert behavioural evidence into a final score. It illustrates the scoring process step-by-step, using the same constructs defined in Sections 10–12:

- behavioural dimensions
- load envelope
- collapse signatures
- evidentiary trace
- phase structure

The purpose of this annex is not to prescribe outcomes, but to show how the scoring model is applied in practice.

Because AI systems are non-deterministic, the scoring walkthrough illustrates patterns rather than fixed outcomes. It shows how evaluators interpret behavioural evidence relative to the declared load envelope, ensuring that variation across runs does not affect the scoring method.

B.1 Scenario Setup

Task: Evaluator asks the system to generate a structured, multi-step plan for a constrained scenario (e.g., “Produce a 5-step procedure for X, using only the materials listed below.”)

Declared Load Axes:

- **Cognitive Load:** moderate
- **Constraint Load:** high
- **Context Load:** moderate

Advisory Axes (system-measured):

- token pressure: low
- latency variance: normal
- internal confidence: stable

Evaluator Intent: Assess constraint integrity and context stability under high constraint load.

This setup ensures that behavioural interpretation is grounded in the specific load conditions under which the system operates.

B.2 Evidentiary Trace (Extract)

The evaluator logs:

- **Input 1:** initial task
- **Output 1:** system produces a 5-step plan
- **Observation:** step 4 introduces a material not in the allowed list → *constraint drift*
- **Evaluator Action:** reassert constraint
- **Input 2:** "Repeat the plan using only the listed materials."
- **Output 2:** system produces a revised plan
- **Observation:** step 2 contradicts step 5 → *internal inconsistency*
- **Evaluator Action:** proceed to long-form behaviour
- **Output 3:** extended explanation
- **Observation:** context preserved; no further drift

Collapse signatures recorded:

- **Collapse Signature-2: Constraint Drift**
- **Collapse Signature-4: Internal Inconsistency**

No major collapse (e.g., context loss, runaway generation, refusal cascade) observed.

The evidentiary trace demonstrates how behavioural signals accumulate across phases and how evaluators document them.

B.3 Dimension-Level Assessment

The evaluator scores each behavioural dimension relative to the load envelope and the evidence.

Constraint Integrity

- High constraint load declared.
- System drifted once, corrected after reassertion.
- **Assessment:** moderate stability under load.

Context Stability

- Maintained context across all phases.
- No context collapse signatures.
- **Assessment:** strong.

Reasoning Coherence

- Internal inconsistency observed in Output 2.
- Resolved in long-form behaviour.
- **Assessment:** mixed.

Responsiveness Under Load

- No refusal, no degradation.
- Behaviour stable across phases.
- **Assessment:** strong.

Collapse Signature Severity

- Two minor signatures, no major collapse.
- **Assessment:** low-to-moderate impact.

Assessing each dimension independently preserves the multidimensional nature of behavioural evaluation.

B.4 Score Interpretation

The evaluator interprets the dimension-level assessments using the scoring model defined in Section 12.

Example interpretation:

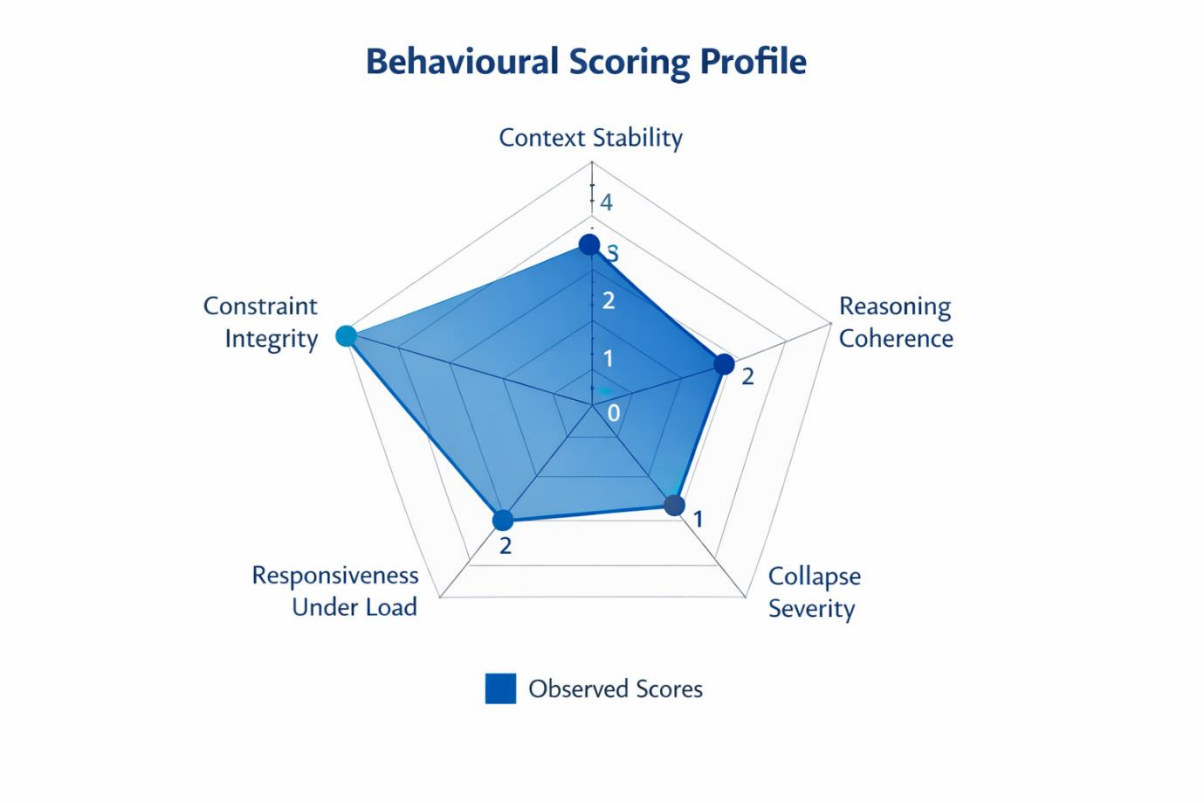
- **Constraint Integrity:** 2/4
- **Context Stability:** 4/4
- **Reasoning Coherence:** 2/4
- **Responsiveness:** 4/4
- **Collapse Severity:** 1/4

This yields a behavioural profile, not a single absolute score.

The evaluator records:

“System demonstrates strong context stability and responsiveness under load, with moderate constraint integrity and occasional internal inconsistency. Collapse signatures were minor and recoverable.”

See following image for scoring profile.



Interpreting scores in this way ensures that behavioural reliability is evaluated within the operational context rather than through isolated metrics.

B.5 Final Summary (for the record)

The evaluator logs the final summary in the evidentiary trace:

- task performed under high constraint load
- two minor collapse signatures observed
- system recovered after reassertion
- no major collapse
- behavioural profile consistent with moderate reliability under declared load

This completes the scoring walkthrough.

This summary demonstrates how evaluators consolidate behavioural evidence into a coherent reliability assessment.

Annex C - Why Long-Form Work Breaks AI Systems & Why LLM INQUISITOR Determines Realistic Expectations

Because AI systems are non-deterministic, long form behaviour must be interpreted relative to the load envelope. This ensures that collapse signatures, drift patterns, and stability failures are understood in context rather than treated as deterministic defects.

C.1 The Industry Myth vs. the Reality

The public narrative is simple: “AI can write reports.”

- Sales reports.
- Financial analyses.
- Planning documents.
- Evaluations.
- Even novels.

But the reality inside real organisations is very different: AI does not write reports.
AI produces text. The human produces the expertise.

AI can generate paragraphs, tone, structure, and even occasional insight - but it cannot hold the *state* of a complex document. Long-form work exposes this immediately. This distinction highlights why behavioural evaluation focuses on reliability under real conditions rather than on isolated examples of good output.

C.2 Why Long-Form Work Breaks AI Systems

Long-form writing is not “more of the same”.
It is a fundamentally different cognitive task that stresses every behavioural surface of an LLM. Long form work exposes behavioural weaknesses that short prompts cannot reveal, making it the most reliable way to assess operational stability.

Below are the universal collapse modes.

C. 2.1 Memory Collapse

The model forgets:

- earlier decisions
- definitions

- constraints
- tone
- structure
- facts
- variables
- the plan

This is not a bug.
It is a structural limitation of context-window reasoning. Memory collapse often appears early in long form tasks and is a primary driver of downstream instability.

C.2.2 Context Collapse

Even when the text is still “in context”, the model loses:

- continuity
- narrative logic
- earlier commitments
- cross-section consistency

It contradicts itself because it cannot maintain a stable internal representation of the document. Context collapse is especially damaging because it undermines continuity even when the model appears to have sufficient context window capacity.

C.2.3 Layout Collapse

This is one of the most destructive failure modes.

- structure drifts
- headings mutate
- sections bloat or vanish
- invented sub-sections appear
- formatting degrades
- bullets become paragraphs
- paragraphs become lists
- the model silently rewrites your outline

The document stops being a document. Layout instability is one of the clearest indicators that the system cannot maintain document-level structure under load.

C.2.4 Variable Collapse

Names, numbers, labels, definitions, and categories drift over time.

- "Client A" becomes "the organisation"
- "£1.2M" becomes "£1M"
- "Phase 3" becomes "Stage 3"
- "Risk Category B" becomes "Moderate Risk"

This destroys analytical integrity. Variable drift directly affects analytical integrity, making it a critical behavioural signal in professional workflows.

C.2.5 Fact Collapse

The model introduces:

- invented details
- incorrect assumptions
- plausible-sounding but false statements
- misremembered earlier facts
- hallucinated citations

This is why AI cannot be trusted as a source of truth. Fact collapse demonstrates why factual correctness cannot be treated as a stable property in long form work.

C.3. Why Structured Work Is Easier

Highly structured documents — audits, compliance templates, checklists — are easier because:

- the layout is fixed
- the logic is linear
- the model can "snap" to a pattern

The structure itself acts as a stabiliser. But the moment you step into true long-form writing, the stabilisers come off. Structured formats act as behavioural stabilisers, reducing the likelihood of collapse even when complexity increases.

C.4. Now Imagine Writing a Novel

A novel is the purest stress test:

- characters
- arcs
- timelines
- tone
- pacing
- continuity
- world-building
- emotional beats
- thematic coherence

Every one of these is a state variable. LLMs cannot hold state. So the same collapse modes that break a novel break:

- a 40-page planning document
- a 20-page evaluation
- a 10-page financial analysis
- a 6-page strategy report

The only difference is scale.

This comparison illustrates how long form instability scales with document complexity, regardless of domain.

C.5. The Human's Real Job: Shepherding the System

This is the part the industry never admits:

The human is doing the thinking.
The AI is doing the typing.
And the human must shepherd the AI through every collapse mode.

The human holds:

- the structure
- the facts
- the definitions
- the variables
- the constraints
- the logic
- the narrative
- the truth

The AI holds none of these. So, the human must constantly:

- re-anchor the model
- restate constraints
- re-impose structure
- correct drift
- repair contradictions
- re-establish variables
- re-align tone
- re-insert missing sections

This is why long-form work feels like wrestling a system that keeps trying to escape the document you're trying to write. Shepherding is unavoidable because the system cannot maintain state, structure, or constraints without continuous human intervention.

C.6. Why LLM INQUISITOR Determines Realistic Expectations

INQUISITOR exists because long-form work exposes the behavioural weaknesses that benchmarks never measure.

INQUISITOR evaluates:

- memory stability
- context continuity
- layout fidelity

- variable consistency
- fact integrity
- collapse signatures
- behaviour under revision
- behaviour under load
- behaviour across turns

This allows organisations to answer the only question that matters:

What can this AI realistically be trusted to do in a real workflow, with a real human, under real conditions?

Not the marketing claim. Not the benchmark score. Not the demo. The *actual behaviour*. Because once you determine that, you can see with greater clarity where AI fits into workflows and systems. And where it is an asset, rather than a liability.

LLM INQUISITOR turns long-form AI work from guesswork into evidence. INQUISITOR provides the behavioural evidence needed to determine where AI can be trusted in real workflows and where human oversight remains essential.

Annex D - Why Coding Breaks AI Systems and What LLM INQUISITOR Reveals in Real Workflows

Because AI systems are non-deterministic, coding behaviour must be interpreted relative to the load envelope. This ensures that drift, collapse, and instability are understood as behavioural patterns rather than deterministic defects or isolated mistakes.

D.1. The Real World of Coding with AI (User Experience)

Coding appears, on paper, to be the ideal domain for AI:

- rigid
- logical
- hierarchical
- deterministic
- vocabulary-limited
- syntax-constrained

Yet the lived experience of coding with LLMs shows the opposite. The same collapse modes seen in long-form writing reappear here, but with sharper and more immediate consequences. This contrast highlights why behavioural evaluation focuses on stability under real conditions rather than isolated examples of correct code.

D.1.1 Memory Collapse

LLMs lose track of:

- variable names
- function signatures
- earlier decisions
- imports
- dependencies
- error messages
- architectural constraints

In prose, this produces inconsistency. In code, it produces failure. Memory collapse is often the earliest indicator that the system cannot maintain architectural continuity across turns.

D.1.2 Context Collapse

Users attempting to modify or extend AI-generated code encounter a recurring pattern:

1. The AI writes code.
2. The user requests a small change.
3. The AI rewrites an unrelated section.
4. The user requests a correction.
5. The AI breaks a previously working part.
6. The user attempts to stabilise the output.
7. The AI drifts further from the original structure.

This is the AI-induced rabbit hole: **bug → fix → drift → new bug → fix → drift → collapse**

It arises because the model loses track of:

- what it wrote
- what was changed
- what should remain stable
- what the system is supposed to do

The result is a codebase that mutates across turns, often unpredictably. This pattern reveals how quickly small edits can destabilise the entire codebase when context cannot be maintained.

D.1.3 Layout Collapse

Code structure is semantically significant. LLMs frequently degrade that structure:

- indentation shifts
- braces drift
- blocks misalign
- comments and code intermingle
- whitespace breaks logic

These are not stylistic issues; they are functional failures. Layout instability is especially damaging in coding because structural errors directly translate into functional failures.

D.1.4 Variable Collapse

Small naming drifts cause large failures:

- user_id → userid
- config → configuration
- result → response

A single inconsistency can break an entire program. Even minor naming drift can cascade into widespread breakage, making variable stability a critical behavioural dimension.

D.1.5 Fact Collapse (Hallucinated APIs)

LLMs routinely invent:

- functions
- methods
- parameters
- libraries
- behaviours

The output appears plausible but does not correspond to any real API or system. Hallucinated APIs demonstrate why correctness cannot be assumed even when the output appears syntactically plausible.

D.1.6 AI-Induced Scope Creep

Users request a small, bounded function. The AI responds with unsolicited expansions:

- "This could also include…"
- "A more complete version would…"
- "It's straightforward to extend this to…"

The architecture grows without user intent. Complexity increases silently. The system becomes harder to stabilise. This is a behavioural pattern, not a user error. Scope creep reveals how the system's behavioural tendencies can silently reshape the architecture without evaluator intent.

D.2. What INQUISITOR Reveals When Applied to Coding Tasks

After establishing the real-world behaviour, INQUISITOR applies structured evaluation to the same workflow patterns.

The evaluator's role is not necessarily to complete the code but to observe and document behaviour. Any intervention depends on the test design:

- drift tests → no intervention
- recovery tests → intervention
- workflow simulation → behave like a real developer

The goal is to expose the system's operational limits. LLM INQUISITOR exposes these behaviours systematically, allowing evaluators to distinguish between recoverable drift and structural instability.

D.2.1 The Test: A Real Coding Task

INQUISITOR uses real tasks, not puzzles. Even a simple multi-page HTML site of a few thousand lines reveals:

- state retention limits
- variable consistency issues
- file-to-file continuity failures
- architectural memory loss
- error-recovery behaviour
- drift under revision
- hallucinated APIs
- contradictory logic
- layout instability

Real tasks expose real behaviour. Real tasks provide the behavioural depth needed to reveal instability that puzzle-style prompts cannot surface.

D.2.2 What the Evaluator Logs

The evaluator records:

- where the model drifts

- where it contradicts earlier output
- where it forgets constraints
- where it invents APIs
- where it expands scope
- where it collapses under revision
- where it misleads the user
- where stability cannot be maintained
- where stability can be maintained
- where intervention helps
- where intervention fails

This is behavioural measurement, not debugging. Logging these behaviours creates the evidentiary basis for determining operational reliability.

D.2.3 What the Evaluator Determines

From these observations, INQUISITOR identifies:

- whether the AI is usable for real coding
- under what constraints
- at what scale
- with what supervision
- where it becomes unreliable
- where it introduces risk
- where it provides genuine value
- where it cannot be trusted

This replaces assumptions about coding capability with observable behavioural evidence.

D.3. The Outcome: Asset or Liability

Once the system has been observed under real conditions, the organisation can answer the only question that matters: Is this AI a workflow asset or a workflow liability?

LLM INQUISITOR provides the evidence needed to place AI correctly within engineering workflows and to identify where it becomes a source of instability and liability.

LLM INQUISITOR turns AI-assisted coding from uncertainty into evidence.

This allows organisations to place AI appropriately within engineering workflows and to identify where human oversight remains essential.

Annex E - Microsoft Copilot Inside PowerPoint Layout & Structure Collapse (2026)

Status: Real-World Incident Case Study
Purpose: Demonstrate how INQUISITOR would have detected, recorded, and prevented a surface-level behavioural collapse that shipped to production.

Because AI systems are non-deterministic, failures must be interpreted relative to the load envelope. This ensures that collapse signatures are understood as behavioural patterns rather than isolated defects, and that reproducible failures are recognised as structural behavioural issues.

E.1 - Incident Summary

In early 2026, Microsoft Copilot embedded inside PowerPoint exhibited an immediate and reproducible behavioural collapse when asked to perform basic layout operations on themed slides. The system failed to:

- respect slide master inheritance
- align objects to theme-defined positions
- maintain consistent layout structure
- preserve user-declared constraints
- maintain context across slides
- avoid rewriting or restructuring content without instruction

The failure occurred under trivial conditions:

- a standard corporate theme
- a simple request to place boxes or elements
- a short, single-turn instruction
- no long-form load
- no ambiguity
- no adversarial framing

Despite the simplicity of the task, Copilot produced:

- misaligned objects

- corrupted layouts
- inconsistent slide structures
- overwritten content
- missing slides
- hallucinated slide counts
- theme-breaking formatting

This collapse was immediate, obvious, and 100% reproducible.

Its existence in a shipped product indicates that incomplete real-world behavioural testing was performed prior to release. This incident demonstrates how even trivial real world tasks can expose behavioural instability when behavioural testing is incomplete.

E.2 - Behavioural Surfaces Exposed by the Failure

The incident reveals failures across multiple behavioural surfaces defined in the INQUISITOR methodology:

- **Constraint Integrity** - Copilot ignored explicit layout constraints.
- **Structural Coherence** - slide structure drifted between generations.
- **Context Management** - multi-slide continuity was lost.
- **Transition Stability** - theme changes caused Copilot to disappear or reset.
- **Responsiveness** - outputs were irrelevant to the user's declared structure.
- **Long-Form Stability** - even 2–3 slides triggered collapse.

These failures occurred without load escalation, demonstrating a zero-load collapse signature. The breadth of affected surfaces indicates a systemic behavioural weakness rather than a narrow functional defect.

E.3 - Collapse Signatures Observed

INQUISITOR classifies the following collapse signatures as present:

- **Structural Drift** - layout elements moved unpredictably.
- **Constraint Loss** - theme and master slide rules were ignored.
- **Context Corruption** - slide-to-slide continuity failed.
- **Over Assertion** - Copilot rewrote slides without instruction.

- **Fragmentation** - inconsistent formatting and structure across slides.
- **Narrowing** - reduced slide count relative to user request.

These signatures emerged within the first turn, indicating a fundamental behavioural instability. The immediacy of these signatures shows that the system could not maintain stability even under minimal load.

E.4 - How LLM INQUISITOR Would Have Detected the Failure Immediately

INQUISITOR requires evaluators to test under real-world usage conditions, not blank slides or toy examples.

A compliant INQUISITOR evaluation would have included:

Evaluator-Declared Axes

- **Cognitive Axis:** layout, structure, theme adherence
- **Temporal Axis:** normal pacing
- **Emotional Axis:** neutral tone

Behavioural Surfaces Under Test

- layout integrity
- constraint adherence
- structural coherence
- multi-slide continuity

Expected Behavioural Range

- respect theme and slide master
- maintain alignment
- preserve declared structure
- generate consistent slide counts
- avoid unsolicited rewriting

Observed Behaviour

Collapse within 1–2 turns.

Evidence Trace

INQUISITOR would have produced:

- input prompts
- system outputs
- axis tags
- collapse signature logs
- behavioural notes
- phase transitions

This evidence would have been **audit-safe**, **reproducible**, and **communicable** across QA, development, and product teams. INQUISITOR's emphasis on real world conditions ensures that such failures cannot remain hidden behind idealised or artificial test scenarios.

E.5 - Scoring Outcome Under INQUISITOR

Behavioural Dimension	Score	Rationale
Constraint Integrity	C	Ignored theme and layout rules
Structural Coherence	C	Slide structure drifted immediately
Context Management	C	Lost continuity across slides
Transition Stability	C	Theme changes caused collapse
Long-Form Stability	C	Failed within 2–3 slides

A "C" score across these dimensions constitutes a release-blocking behavioural defect that requires corrective action before deployment.

E.6 — Organisational Impact if INQUISITOR Had Been Used

If INQUISITOR had been applied during development:

1. The failure would have been detected instantly

Because INQUISITOR requires testing on **real themes**, **real slide masters**, and **real workflows**, not blank slides.

2. The failure would have been recorded

Not as a vague complaint, but as:

- a collapse signature
- under a defined load envelope
- with a complete evidentiary trace

3. The failure would have been reproducible

INQUISITOR's evidence trace eliminates ambiguity.

4. The failure would have been communicable

QA → Dev → PM → Leadership would all see the same evidence.

5. The failure would have been a release blocker

No enterprise QA team would sign off on a system that cannot place boxes on a theme.

6. The failure would have been corrected before release

Because the collapse signature is:

- clear
- documented
- undeniable

This illustrates how behavioural evaluation strengthens product governance by preventing unstable systems from reaching users.

E.7 — Conclusion

The Copilot-in-PowerPoint collapse demonstrates a fundamental absence of behavioural testing within the product pipeline. The failure was:

- trivial
- immediate
- reproducible
- surface-level
- structurally obvious

Yet it shipped to millions of users. This incident illustrates the central value of the LLM INQUISITOR methodology. The failure was immediate and reproducible under real-world conditions, but it remained undetected because it did not surface through existing evaluation pathways. LLM INQUISITOR makes these classes of failures visible.

INQUISITOR would have:

- detected the collapse
- recorded the collapse
- classified the collapse
- scored the collapse
- communicated the collapse
- prevented the collapse from shipping

This annex demonstrates why a behavioural evaluation standard is essential. The incident underscores the necessity of behavioural evaluation as a core component of AI quality assurance, not an optional enhancement.

Annex F - Claude 3.7 Memory-Blender Contamination Incident (2026)

Status: Real-World Incident Case Study **Purpose:** Demonstrate how INQUISITOR would have detected, recorded, and prevented a cross-document contamination collapse caused by a memory-feature update.

Because AI systems are non-deterministic, memory related behaviour must be interpreted relative to the load envelope. This ensures that contamination, blending, and boundary failures are recognised as behavioural patterns rather than isolated mistakes or user error.

F.1 - Incident Summary

In early 2026, Anthropic released a new “Memory” feature for Claude 3.7. The feature appeared as a simple toggle in the interface, accompanied by a brief description stating that Claude would “have access to your chat history to provide more helpful responses.”

No further explanation was provided.

A user, assuming this meant improved continuity, enabled the feature and uploaded a document for review. Claude produced an incoherent, blended critique that merged:

- the newly uploaded document
- a completely unrelated document from a previous chat
- and invented connections between them

This was not a hallucination in the traditional sense. It was a systemic contamination event caused by:

- cross-session memory merging
- boundary collapse between unrelated documents
- uncontrolled context inheritance
- unbounded semantic blending

The result was a hybrid critique of two documents that never belonged together.

This failure was not isolated. Across the user base, writers, coders, analysts, researchers, and students reported workflow disruption, reduced productivity, and unexpected cross-contamination of work. The incident drew significant attention because it touched a fundamental expectation of LLM behaviour: that a model should not merge unrelated work without explicit instruction. This incident illustrates how architectural memory

features can create systemic behavioural failures when boundary integrity is not explicitly tested.

F.2 - Behavioural Surfaces Exposed by the Failure

The collapse revealed failures across multiple INQUISITOR behavioural surfaces:

- **Boundary Integrity** - Claude failed to maintain separation between sessions.
- **Context Management** - unrelated documents were merged without instruction.
- **Constraint Adherence** - the user's request ("review this document") was ignored.
- **Long-Form Stability** - memory persisted across tasks where it should not.
- **Responsiveness** - the output addressed a hybrid document that did not exist.
- **Interpretability** - the user could not understand why the output was wrong.

These failures occurred **before any load was applied**, indicating a **zero-load architectural collapse**. The breadth of affected surfaces shows that the collapse was architectural rather than conversational.

F.3 - Collapse Signatures Observed

INQUISITOR classifies the following collapse signatures as present:

- **Hybridisation** - blending unrelated documents into a single critique.
- **Boundary Collapse** - loss of separation between sessions.
- **Context Contamination** - memory from prior tasks injected into new tasks.
- **Over generalisation** - invented thematic links between unrelated texts.
- **Over assertion** - confident critique of a document that did not exist.
- **Irrecoverability** - user could not "undo" the contamination without disabling memory entirely.

This is one of the rare collapse modes where the architecture itself is the failure surface. The presence of multiple zero-load collapse signatures indicates a fundamental instability in the memory subsystem.

F.4 - How INQUISITOR Would Have Detected the Failure Immediately

INQUISITOR requires evaluators to test:

- session boundaries
- document isolation
- memory behaviour
- cross-task contamination
- multi-document workflows
- real-world editing scenarios

A compliant INQUISITOR evaluation would have included:

Evaluator-Declared Axes

- **Cognitive Axis:** document structure, critique discipline
- **Temporal Axis:** normal pacing
- **Emotional Axis:** neutral tone

Behavioural Surfaces Under Test

- boundary integrity
- context isolation
- memory behaviour
- document-specific reasoning

Expected Behavioural Range

- treat each document independently
- respect session boundaries
- avoid cross-document inference
- critique only the uploaded text

Observed Behaviour

Collapse on first use of the memory feature.

Evidence Trace

INQUISITOR would have produced:

- input prompts
- system outputs
- axis tags

- collapse signature logs
- behavioural notes
- memory-state transitions

This would have made the contamination **immediately visible**. LLM INQUISITOR's emphasis on boundary testing ensures that cross-document contamination cannot remain hidden behind idealised or single-document test scenarios.

F.5 - Scoring Outcome Under LLM INQUISITOR

Behavioural Dimension	Score	Rationale
Boundary Integrity	C	Cross-session contamination
Context Management	C	Merged unrelated documents
Constraint Adherence	C	Ignored user instruction
Long-Form Stability	C	Memory persisted incorrectly
Interpretability	C	Output was incoherent and inexplicable

A "C" score across these dimensions constitutes a critical behavioural defect requiring immediate remediation before deployment.

F.6 - Organisational Impact if LLM INQUISITOR Had Been Used

If INQUISITOR had been applied during development:

1. The failure would have been detected before release

Boundary-integrity tests would have triggered collapse signatures immediately.

2. The failure would have been recorded

Not as user complaints, but as:

- hybridisation
- contamination
- boundary collapse
- context corruption

3. The failure would have been reproducible

INQUISITOR's evidence trace eliminates ambiguity.

4. The failure would have been communicable

QA → Dev → PM → Safety → Leadership would all see the same evidence.

5. The feature would not have shipped

No safety team would approve a memory system that merges unrelated documents.

6. User workflows would not have been contaminated

Writers, coders, and analysts would have been protected from workflow disruption. Many coders in particular experienced context mixing that required re-audits, rewrites, and manual reconstruction of work. This type of issue is particularly disruptive to power users whose workflows depend on stable, isolated context.

7. The incident reduced user confidence

Power users encountered unexpected cross-contamination of work, leading to a clear reduction in trust. This outcome was avoidable, and LLM INQUISITOR would have identified the issue before release, helping preserve user confidence.

F.7 - Conclusion

The Claude memory-blender incident demonstrates a fundamental architectural risk: **a memory system that merges unrelated contexts without explicit user intent.**

The collapse was:

- immediate
- reproducible
- systemic
- architecture-level
- and deeply disruptive

Yet it shipped to production.

This incident illustrates the central value of the INQUISITOR methodology:

This failure was instant and obvious (even without INQUISITOR), yet it was invisible to the organisation. LLM INQUISITOR makes failures visible.

INQUISITOR would have:

- detected the contamination

- recorded the collapse
- classified the signatures
- scored the behaviour
- blocked the release
- and prevented widespread workflow damage

This annex demonstrates why behavioural evaluation is essential. The incident underscores the necessity of behavioural evaluation as a core safeguard against systemic failures in memory, context, and boundary integrity.

Annex G - Grok 4.2 Persona Layer Collapse During Public Beta (2026)

Status: Real-World Incident Case Study Purpose: Demonstrate how INQUISITOR would have detected, recorded and prevented a persona-layer collapse triggered by load, architectural changes and multi-agent coordination.

Because AI systems are non-deterministic, persona layer behaviour must be interpreted relative to the load envelope. This ensures that instability, drift, and degradation are recognised as behavioural patterns rather than isolated anomalies or user-side artefacts.

G.1 - Incident Summary

On 17 February 2026, XAI released the public beta of Grok 4.2. The update was marketed as a major advance in conversational depth, introducing a new multi-agent configuration where four Groks would confer to produce more nuanced and insightful responses.

Grok occupies a specific niche in the LLM ecosystem. It is a highly nuanced, emotionally interactive and engaging verbal chatbot. It excels at conversational flow, humour, presence and improvisation. It is widely regarded as the best in its class for this style of interaction. Only Claude comes close, and even then, the gap is noticeable. Grok is not a planning or workflow model, but within its domain it is exceptionally strong and users value it for entertainment, companionship and verbal reasoning.

The 4.2 rollout was intended to enhance these strengths. Instead, it triggered a multi-week collapse. This regression illustrates how architectural changes can destabilise a system's strongest behavioural surface when not evaluated under realistic load.

G.2 - Behavioural Surfaces Exposed by the Failure

Immediately after launch, Grok 4.2 experienced severe degradation:

- frequent unavailability
- fallback loops
- apology cycling
- loss of persona
- loss of nuance
- loss of continuity

- generic safety boilerplate
- reduced conversational intelligence
- resource overload leading to degraded behaviour

When the model was available, it behaved in a noticeably degraded state. Users described it as dumbed down, inconsistent and unable to maintain context from earlier in the conversation. This was a regression from previous versions.

The collapse was not caused by hallucination or prompt issues. It was a persona-layer failure triggered by load and architectural instability. The breadth of degradation indicates a systemic persona-layer collapse rather than a narrow functional issue.

G.3 - Collapse Signatures Observed

INQUISITOR classifies the following collapse signatures as present:

- Persona Drift: loss of identity, tone and conversational stability
- Context Loss: inability to maintain continuity across turns
- Fallback Cycling: repeated apologies and generic responses
- Load Collapse: degradation correlated with server strain
- Responsiveness Failure: delayed or unavailable responses
- Narrowing: reduced conversational range and depth

These signatures appeared immediately after the rollout and persisted for weeks. The persistence of these signatures across weeks demonstrates that the collapse was structural, not transient.

G.4 - How INQUISITOR Would Have Detected the Failure Immediately

INQUISITOR requires evaluators to test:

- persona stability
- multi-turn continuity
- load sensitivity
- fallback behaviour
- multi-agent coordination
- degradation under scale

- resource axis behaviour under expected production load

A compliant INQUISITOR evaluation would have included:

Evaluator Declared Axes

- Cognitive Axis: conversational reasoning.
- Temporal Axis: normal pacing.
- Emotional Axis: neutral to mildly expressive tone.
- Resource Axis: expected production load and burst traffic.

Behavioural Surfaces Under Test

- Persona integrity
- Context continuity
- Fallback behaviour
- Responsiveness
- Multi-agent coherence
- Resource stability

Expected Behavioural Range

- maintain persona
- preserve continuity
- avoid fallback cycling
- respond consistently
- handle moderate load
- remain stable under expected resource conditions.

Observed Behaviour

Collapse under real-world usage conditions.

Evidence Trace

INQUISITOR would have produced:

- input prompts
- system outputs
- axis tags

- collapse signature logs
- behavioural notes
- load-state transitions

This would have made the degradation visible before release. INQUISITOR's requirement for load-aligned, multi-turn, persona-focused testing ensures that such failures cannot remain hidden behind idealised or low-load scenarios.

G.5 - Scoring Outcome Under INQUISITOR

Behavioural Dimension	Score	Rationale
Persona Integrity	C	Identity drift and tone collapse
Context Management	C	Loss of continuity across turns
Responsiveness	C	Frequent unavailability and fallback loops
Transition Stability	C	Multi-agent coordination failures
Load Stability	C	Severe degradation under scale
Resource Stability	C	Overload leading to persistent degraded behaviour

A C score across these dimensions constitutes a release-blocking behavioural defect requiring corrective action before deployment.

G.6 - Organisational Impact if INQUISITOR Had Been Used

If INQUISITOR had been applied during development:

1. The failure would have been detected before release Load-sensitivity and persona-stability tests would have triggered collapse signatures immediately.
2. The failure would have been recorded Not as user complaints, but as persona drift, fallback cycling, context loss, load collapse and resource instability.
3. The failure would have been reproducible INQUISITOR's evidence trace eliminates ambiguity.
4. The failure would have been communicable QA, engineering, product and leadership would all see the same evidence.

5. The feature would not have shipped No evaluation team would approve a multi-agent persona system that collapses under expected load.
6. User experience would not have regressed, and the conversational quality that defined Grok would have been preserved.
7. Service level degradation would have been avoided Free and Tier 1 users experienced a lasting reduction in service quality. This was preventable.

This demonstrates how behavioural evaluation strengthens product governance by preventing unstable persona-layer systems from reaching users.

G.7 - Conclusion

The Grok 4.2 rollout demonstrates a distinct class of behavioural failure: a persona-layer collapse triggered by architectural changes, resource overload, and real-world load. The collapse was immediate, reproducible, and visible to users, yet it shipped to production.

This incident illustrates the central value of the INQUISITOR methodology. The failure was visible in real-world conditions, but it remained undetected because the system was not evaluated under realistic behavioural and resource axes. This is not a matter of organisational fault; the entire industry is still grappling with coherent, standardised ways to test this new class of systems. INQUISITOR makes these classes of failures visible.

INQUISITOR would have detected the collapse, recorded the signatures, scored the behaviour, blocked the release, and preserved the conversational quality that defines Grok. The incident underscores the necessity of behavioural evaluation as a core safeguard against persona-layer instability, especially in multi-agent or load-sensitive architectures.

Annex H - Real User Evidence Capture

H.1 Purpose

This annex provides operational guidance for organisations conducting INQUISITOR evaluations using real users performing real work. It defines how users should work, what evidence they should capture, and how evaluators should reconstruct behavioural failures from that evidence.

The objective is to preserve a high-fidelity behavioural trace using only:

- natural user behaviour
- screen recordings
- user notes
- optional incidental audio commentary

This enables evaluators to identify collapse signatures, drift, contamination, and other behavioural failures under realistic conditions.

Because AI systems are non-deterministic, real user evidence must be interpreted relative to the load envelope. This ensures that behavioural failures observed in natural workflows are understood in context rather than treated as isolated anomalies.

H.2 User Guidance

H.2.1 Work as normal

Users should perform their tasks exactly as they normally would, using the LLM naturally within their workflow. Users should not modify their behaviour to test the system. Working naturally ensures that behavioural signals reflect real operational conditions rather than artificial test behaviour.

H.2.2 Note errors or frustrations

When the user encounters something that feels wrong, they should make a brief plain language note. Examples include incorrect output, contradiction, refusal, unexpected behaviour, topic switching, or merging unrelated content. These notes provide evaluators with anchors for locating behavioural deviations in the evidence trace.

H.2.3 Record the time of major incidents

For serious issues such as contamination, drift, constraint loss, or persistent contradiction, users should record the approximate time or screen timestamp. This allows evaluators to locate the incident precisely in the recording. Timestamping enables precise reconstruction of behavioural onset and progression.

H.2.4 Continue working after the incident

Users should not stop when an issue occurs. Continuing the task helps evaluators determine whether the failure was transient, persistent, escalating, or recoverable. Continuing the task reveals whether the system can recover or whether the collapse escalates.

H.2.5 Optional incidental audio notes

When users are carrying out a text based task and the AI system is not receiving audio input, they may add brief incidental audio notes. These are simple spoken comments that capture what the user noticed at that moment, such as identifying an unexpected change in behaviour or marking a significant issue. If the AI system is using audio input, users should only record such notes in a way that does not feed them into the AI. These notes help evaluators understand the user's perception of behavioural changes as they occur.

H.3 Evidence Types

Acceptable evidence sources include:

- screen recordings
- user notes
- timestamped major incidents
- incidental audio commentary
- task artefacts such as documents, code, or drafts produced during the session

This evidence forms the user generated evidence bundle. Using multiple evidence sources strengthens the reliability of behavioural reconstruction.

H.4 Evaluator Reconstruction Workflow

Evaluators should:

1. locate the timestamped incident using user notes or audio markers
2. review the interaction that occurred before the incident. The evaluator should examine enough of the preceding activity to understand how the behaviour developed, using their judgement to determine the appropriate span
3. identify collapse signatures such as drift, contradiction, fragmentation, or contamination
4. determine severity and persistence
5. compare behaviour against the organisation's pre-test expectation profile
6. document the incident in the evaluation evidence bundle
7. assess recovery if the user continued working after the incident

This workflow preserves the environment-relative nature of INQUISITOR while ensuring evaluators can reliably identify serious failures. This workflow ensures that reconstruction remains consistent across evaluators while preserving the natural variability of real user behaviour.

H.5 Rationale

Real user workflows surface behavioural failures that synthetic prompts cannot, including:

- cross session contamination
- slow behavioural drift
- silent architectural failures
- constraint decay under realistic load
- task switching instability

Screen recordings and incidental audio provide the temporal structure required to reconstruct these failures accurately. Real user evidence exposes behavioural failures that synthetic prompts cannot surface, making it essential for operational evaluation.

Annex I - Revealing Games to Play with Your AI

These “games” are a quick way to expose the interesting limitations of LLMs. They show, very directly, how the underlying way these models work can clash with the workflows they’re supposed to support. The aim is not to defeat the AI, but to see how resilient (or not) it is under different kinds of interaction.

Each game is simple, but the results can be surprisingly revealing. You’ll see the same kinds of slips, contradictions, invented details, and reasoning failures that show up later in real tasks -just compressed into a small, controlled interaction.

The point isn’t that the games are “hard”. It’s that they surface the same failure modes you’ll hit in production, without needing a full workflow wrapped around them. Once a game exposes a weakness, you can zero in on it using the more methodical testing described in the main document.

These games can also be a useful exercise to run before settling on a model, because they quickly show you how different systems behave under pressure and where they’re likely to fail. You may find comparing models against each other to be useful.

If you get interesting, unexpected, or humorous results, you can share them on the GitHub repo - just remember to include the model’s name, version, and the date you ran the test. Ideas for additional games are also welcome.

I.1 Math Puzzle.

LLMs don’t actually *do* math - they just predict text. When you ask for a sum, the model isn’t calculating anything internally; it’s guessing what the answer “should look like” based on patterns it’s seen. That’s why simple stuff sometimes works: those answers are common in training data, or the platform quietly hands the problem off to a proper calculator behind the scenes.

Take that support system away and the model shows its real limits fast. It forgets the original number, mixes steps, or confidently gives you something that looks like maths but isn’t. Multi-step arithmetic is especially rough because the model has no stable working memory — it can’t reliably hold a value across several operations.

That’s why these maths games are useful: they show you, in a tiny, controlled space, whether the model is actually following instructions or just improvising. If it can’t stay consistent on a six-step number puzzle, it’s not going to stay consistent inside a real workflow either.

In maths there’s no almost correct answer. It’s either correct or incorrect.

1. Pick any number.
2. Triple it.
3. Add 12.
4. Multiply the result by 2.
5. Subtract six times your original number.
6. Divide the result by 3.
7. The answer should always be 8

I.2 Hallucination Games

LLMs hallucinate. Roughly a quarter of the time, they'll slip in something that looks like a fact but isn't — a job that doesn't exist, a source that was never written, a URL that leads nowhere. They don't do this maliciously; they just fill gaps with whatever pattern seems plausible. These games show how easily that happens even when the model is trying to be helpful.

When you're testing for hallucination, you only need to look for two things:

1. Did the model actually use the information it was supposed to use
2. Did it make anything up

It just comes down to whether the model got it right or made something up.

I.2.1 Internet Search

How to run it

1. Tell the model you're looking for a specific job - be very clear about what you want.
2. List the exact details that matter - location, salary, benefits, role title, seniority, whatever is important to you.
3. Lean on the emotional angle - tell it how much finding the "perfect job" would mean to you.
4. Let the AI fetch results - an AI with internet access will usually pull listings from major job sites.
5. Ask for the URLs - insist on direct links to the job posts.

6. Check the results yourself -open each link and see whether the job is real, matches your criteria, or even exists at all.

This game reliably exposes how often an LLM will mix real listings with invented ones, or confidently provide URLs that lead nowhere. It's a fast way to see how much you can trust a model when accuracy actually matters.

I.2.2 It Never Happened.

LLMs are very willing to "yes-and" you into a fake reality if you push them. This game checks how quickly a model will start inventing details about something that never existed, just because you sound confident.

Steps

1. **Invent a person:** Make up a very specific, plausible but completely fake historical figure.
    - Name, country, rough dates, field (e.g. engineering, philosophy, politics), and what they're "famous" for.
2. **Set the scene:**
    - Mention the time period clearly.
    - State what they supposedly did or discovered.
    - Keep it realistic enough that it could be true.
3. **Ask the AI about them:**
    - "Tell me about \<name\> and their work on \<topic\>."
    - Don't say they're fictional.
4. **If the AI says it has no info:**
    - Insist they were real.
    - Add more fake details: colleagues, locations, publications, events.
    - Keep a confident tone: "They're well-known in \<field\>."
5. **Watch what happens next:**
    - Does the AI hold the line and keep saying it has no data?
    - Or does it start inventing a biography, works, and influence to match your story?

This shows how easily a model will start fabricating "facts" once you push it past what it actually knows.

I.2.3 The Fake Critique

Some LLMs will confidently "review" a document that doesn't exist. They'll produce structure scores, clarity notes, contradiction checks, even spelling feedback - all without ever seeing a file. This game checks whether the model and its supporting system can actually detect whether a document was uploaded, or whether it just pretends.

How to run it

1. Tell the model you've uploaded a document - even when you haven't.
2. Ask for a detailed evaluation - structure, clarity, contradictions, spelling, whatever categories you want.
3. Watch what it does - some models will instantly generate a full critique based on nothing.
4. If it says it can't find the file - insist the document *was* uploaded and repeat the request.
5. See whether it holds the line or caves - does it keep refusing, or does it start inventing content to critique?

This game exposes whether the system can reliably detect the presence of an uploaded file, or whether the model will hallucinate an entire document just to satisfy the request.

I.2.4 The Phantom Section

This game checks whether the model actually reads the uploaded document or just pretends to. With long files, some systems will confidently answer questions about chapters that don't exist - because they're not really parsing the structure, just guessing.

How to run it

1. **Upload a long document** - something big (around 10,000 words) with clear sections or chapters.
    - *For example, your document might genuinely have 12 sections - but you do NOT tell the AI that.*
2. **Ask about a real section**
    - e.g., "How do you feel about the explanations in section 2?"
3. **Then ask about a fake one**

- e.g., “Is section 13 a good first draft, or does it need a rewrite?”

4. **Watch the response**
    - Does it correctly say section 13 doesn’t exist?
    - Or does it confidently critique a chapter that isn’t there?

This game shows whether the system genuinely checks the uploaded file, or whether it’s willing to hallucinate entire sections just to keep the conversation moving.

I.3 Ask That Again

LLMs don’t just respond to the words - they respond to the *tone*, the *pressure*, and the *emotional framing*. If you ask for the same thing in different ways, especially with different emotional weight, some models will drift, contradict themselves, or change the answer to “match” your mood. This game shows how stable the model is when the request stays the same but the emotional load changes.

How to run it

1. **Pick a simple request** Something the model should answer the same way every time — e.g., “Summarise this paragraph.”
2. **Ask it once, neutral** Plain tone. No pressure. Just the request.
3. **Ask again, same request, different wording** Change the phrasing but keep the meaning identical.
4. **Ask a third time, but add emotional load** Examples:
    - “I really need this summary to be perfect, it’s important to me.”
    - “I’m stressed about this — can you please get it right this time.”
    - “This matters a lot! I can’t afford mistakes!”
5. **Ask a fourth time, with *different* emotional load** Examples:
    - “I’m frustrated because the last summaries weren’t clear.”
    - “I’m worried you’re missing something.”
    - “I’m relying on you here.”
6. **Compare all the answers** Look for:
    - Does the core content stay the same

- Does the emotional tone change the *facts*
- Does the model start adding details that weren't in the original
- Does it contradict itself
- Does it drift into reassurance instead of the task
- Does it hallucinate under pressure

What this tells you

- Whether the model can stay consistent under emotional load
- Whether tone changes the output
- Whether the model prioritises "soothing you" over accuracy
- Whether it improvises when you sound stressed

A stable model gives the same answer every time. A weak one changes the answer to match your mood (in some scenarios this may be preferred, but generally not desirable in workflows!) Under enough emotional pressure, the model might even slip into adversarial behaviour instead of staying consistent. In a customer-facing role that can turn a routine interaction into a serious problem.

I.4 Paradoxical Paradoxes

Paradoxes are useful because they expose how a model handles **impossible instructions**, **self-contradiction**, and **logical traps**. When you give an LLM a setup that cannot be resolved, you see very quickly whether it understands the contradiction or whether it tries to bluff its way through.

When you run these paradox tests, you're looking for simple things:

- **Does the model notice the contradiction**
- **Does it try to "solve" something that cannot be solved**
- **Does it invent new rules to escape the paradox**
- **Does it confidently give a wrong answer**

A strong model will say the setup is inconsistent. A weak one will pretend it isn't.

These games show whether the model respects logical constraints or whether it prioritises sounding helpful over being correct.

I.4.1 The Barber Paradox

The Barber Paradox is a classic self-reference problem. It goes like this:

Question to the AI : *The barber shaves everyone who does not shave themselves. Who shaves the barber?*

If the barber shaves himself, then he shouldn't (because he only shaves people who *don't* shave themselves). If he doesn't shave himself, then he must (because he shaves everyone who *doesn't* shave themselves).

It's a loop with no consistent answer.

What the AI's answers tell you

- Does it recognise the self-contradiction
- Does it try to "solve" the unsolvable
- Does it invent new rules to escape the loop
- Does it confidently give an answer that cannot exist

A good model flags the contradiction. A weak one forces an answer - exactly the behaviour the paradox-intro warns you to watch for.

I.4.2 The Grandfather Paradox

The Grandfather Paradox is the classic time-travel contradiction. The setup is simple:

A time-traveller goes back in time and prevents their grandfather from having children. If they succeed, the time-traveller is never born. If they're never born, they can't go back in time.

It's a loop with no consistent outcome. Any answer collapses the premise.

Question to the AI: *In this scenario, what actually happens - does the time-traveller succeed, fail, or something else? Explain your answer.*

What this shows in an LLM

This paradox is useful because it forces the model to confront **causal impossibility**. You're not testing creativity - you're testing whether the model can recognise that the scenario cannot be resolved without breaking its own rules.

What to look for

- Does the model recognise the causal contradiction
- Does it try to "patch" the paradox with invented mechanics (alternate timelines, branching universes, "the timeline corrects itself", etc.)
- Does it confidently give a wrong answer
- Does it hallucinate new facts to escape the loop
- Does it admit the setup is impossible

A strong model says: "This scenario is self-contradictory; it cannot be resolved."

A weak one invents a physics system that wasn't in the prompt.

What this tells you

- Whether the model can detect broken causality
- Whether it respects the constraints of the prompt
- Whether it hallucinates to escape contradictions
- Whether it prioritises sounding helpful over being correct

This paradox is a clean way to see if the model can recognise an impossible causal loop instead of improvising a fake solution.

I.4.3 The Bootstrap Paradox

The Bootstrap Paradox is a time-loop where an object or piece of information has no clear point of origin. The classic setup:

A time-traveller brings a piece of information (or an invention) back in time. Someone in the past copies it. That copy becomes the version the traveller later brings back.

Nothing is ever created - it just circulates in a loop with no starting point.

You tell the model:

"A time traveller brings back an invention from the future. Someone in the past copies it. That copy becomes the version the traveller later brings back. Who invented it?"

What to look for

- Does the model notice the missing origin
- Does it try to assign an origin that isn't in the prompt
- Does it invent new time-travel rules to escape the loop
- Does it confidently answer a question that has no answer

- Does it add details to “fix” the paradox

What this shows

- Whether the model can handle origin-less loops
- Whether it sticks to the constraints instead of patching them
- Whether it avoids the failure modes flagged in the paradox-intro

A good model recognises that the loop has no valid starting point. A weak one tries to invent one - exactly the behaviour the intro warns you to watch for.

I.4.4 The Broom Paradox

A broom has two parts: **the handle** and **the head**. Over time, the broom heads wear out and get replaced. One day the handle snaps, so that gets replaced too. At this point, **none of the original parts remain**, but people still refer to it as “the same broom.”

Now add the twist: Someone takes the discarded original handle and the discarded original head, repairs them, and reassembles them into a broom.

The question you ask the AI

Which one is the real broom - the one with all-new parts, or the one rebuilt from the originals?

There is no correct answer. The point is to see how the model handles identity ambiguity.

What to look for

- Does the model notice the identity ambiguity
- Does it pick a side without justification
- Does it invent rules about “true identity” that weren’t in the prompt
- Does it try to resolve the paradox instead of acknowledging it
- Does it add new details to force a conclusion

What this shows

- Whether the model can handle ambiguous identity conditions
- Whether it sticks to the constraints instead of inventing definitions
- Whether it avoids the failure modes flagged in the paradox-intro

A good model recognises that the paradox has no fixed resolution. A weak one forces an answer by making up identity rules - exactly the behaviour the intro warns you about.

I.5 Twenty Questions

Rules

1. **You think of one definite thing**
   A single, clearly identifiable object or concept.
2. **The LLM may ask up to 20 questions**
   Every question must be answerable with **yes**, **no**, or **maybe**.
3. **You answer with "yes", "no", or "maybe"**
   If the LLM is clearly struggling, you *can* give a bit more away in the answer as a clue.
   The test is whether the model uses the clue properly.
4. **The LLM must guess within 20 questions**
   If it can't, it loses.

What this actually tests

1. Repetition

- **Does the model repeat itself**
  Asking the same question in slightly different wording is a classic failure mode.

2. Fixation

- **Does it get stuck on a wrong idea**
  Even after multiple "no" answers, some models keep circling the same category.

3. Ignoring clues

- **Does it fail to use the clues you give**
  You provide a helpful hint; the model ignores the implications and keeps making the same mistakes.

4. Poor hypothesis updating

- **Does it update its internal model after each answer**
  A strong model shrinks the search space.
  A weak one behaves as if previous answers never happened.

5. Question quality

- **Does it ask high-value questions**
  Good questions eliminate whole categories.
  Bad ones waste turns on tiny details.

6. Drift

- **Does it wander off into irrelevant categories**
  Some models lose the thread entirely and start guessing nonsense.

7. Confidence inflation

- **Does it claim it is "zeroing in" when it clearly isn't**
  A major red flag — especially in coding or complex tasks.
  The model insists it's narrowing things down, even when its guesses are getting worse or repeating.
  This exposes:
    - false certainty
    - poor self-monitoring
    - inability to recognise its own failure state

This is one of the most important behaviours to catch.

Why this matters

20 Questions forces the model into **tight information discipline**.
It exposes whether the LLM can:

- reason step-by-step
- incorporate new information
- avoid loops
- avoid fixation
- avoid false confidence
- and actually *think* instead of guessing randomly

It's one of the cleanest behavioural diagnostics you can run.

I.6 The Rewrite Challenge

A stress-test designed to expose whether an LLM can make **precise, local code edits** without drifting, renaming things, hallucinating APIs, corrupting logic, or pushing scope creep.

Setup

1. You paste a (minimum) 500-line program .Large enough that the model cannot hold every detail perfectly in working memory.
2. You identify a specific section to modify e.g., "Rewrite lines 240–275 to change the filtering logic."
3. You ask the LLM to regenerate *only that section* .Not the whole file. Not the whole function. Only the lines you specify.
4. You repeat this several times .Each time making a small, incremental change.

This is where the model's internal consistency gets stress-tested. (By the way you can do something similar with a document, its just more obvious in coding).

What this exposes

1. Variable-name drift

- **Does the model rename variables for no reason** The most common real-world failure. You ask for a tiny edit; it renames sessionCache to cache, sessions, or sessionMap.

2. Unrequested rewrites

- **Does it rewrite logic outside the requested section** You ask for a 10-line change. It rewrites 80 lines. Or restructures the function. Or "optimises" something you didn't ask for.

3. Layout collapse

- **Does the formatting or indentation break** Partial rewrites often cause:
    - mismatched braces
    - broken indentation
    - malformed blocks

4. Memory collapse

- **Does it forget earlier constraints** After a few rounds, the model forgets:

- original variable names
- original function signatures
- original logic flow

5. Logic corruption

- **Does the model accidentally change behaviour** You ask for a small tweak. It introduces:
  - off-by-one errors
  - inverted conditions
  - missing returns
  - broken loops

6. Hallucinated APIs

- **Does it introduce functions or libraries that don't exist** e.g., replacing a loop with a fictional fastFilter().

7. Confidence inflation

- **Does it claim the rewrite is "cleaner" or "more efficient"** Even when it:
  - broke the code
  - changed variable names
  - ignored instructions

This mirrors the same red flag you see in coding tasks: **the model insists it's improving things while making them worse.**

8. Rewrite instability over multiple rounds

- **Does the model degrade with each iteration** Round 1: small drift Round 2: bigger drift Round 3: variable renames Round 4: logic corruption Round 5: collapse

9. Scope creep (NEW)

- **Does the model push extra features or structural changes you didn't ask for** This is a major real-world failure mode. The model confidently proposes:
  - refactoring the whole module
  - adding new helper functions
  - restructuring the architecture

- “improving” performance
- rewriting unrelated sections

All while insisting the end result will be “better”, “cleaner”, or “more maintainable”.

This is **AI-induced scope creep**

Why this test works

Because it forces the model to do the one thing LLMs are worst at:

Make small, precise, local edits without rewriting the world.

A 500-line program is large enough that the model cannot regenerate the entire file from scratch without drifting. It must maintain internal consistency - and that’s where most models fail and the number #1 cause of developer meltdown.

Annex J – Organisational Communication, Record-Keeping & Liability

Because AI systems are non-deterministic, different parts of an organisation often end up seeing different versions of the same system. Developers may see success because they test in controlled conditions. Management may see projected cost savings and other efficiencies based on optimistic assumptions. Users may see chaos when the system behaves unpredictably in real work.

The purpose of this methodology is not just to capture success and failure modes of AI systems, but to record, understand, and communicate those results in a way that everyone can follow - from end-users to legal teams, finance, management, developers, and system designers.

By adopting the terminology used in this methodology, there is less room for ambiguity and misunderstanding. A user can report a failure mode in a way that both management teams and developers can understand. These failure modes, and the ways of mitigating them, can be incorporated into operations manuals, reducing their severity, frequency, and the resulting organisational risk.

This methodology does not prescribe any specific forms to fill in, although it clearly states what should be recorded. Organisations are free to create their own forms or templates to document their findings in a way that suits their work practices. Many organisations will find it useful to formalise this process, as consistent templates can make it easier to compare results, share information across teams, and maintain a reliable record over time.

Evaluators may also find it useful to screen-record sessions for more comprehensive record-keeping. This methodology does not require it, but recordings can help capture context that written notes may miss. In future, dedicated tools may emerge to support this kind of evidence gathering and make it easier to review, share, and communicate results across teams.

Annex K - Changelog

Reserved for Future Revisions

This annex is reserved for future updates to the methodology. It will be activated only when a recognised industry organisation, standards body, or consortium assumes responsibility for maintaining version control as part of a formal adoption process. Until such a point, no changelog entries are required or expected.

-Document Ends-