VELOCITY EDGE

MANAGING CONTEXT IN LONG-RUNNING AGENTS

A Configuration Model, a Criterion for What Can Be Held, and a Reference Design

ABSTRACT

Why does an AI agent stop following its instructions in a long session, and what can actually be done about it? This paper takes the question Anthropic posed in 2025, “what configuration of context is most likely to generate our model’s desired behavior?”, and treats it as an engineering problem for agents that run for hours. It makes four moves. First, it models an agent’s context as the output of an assembly function governed by a vector of settings of two kinds: context settings, which decide what the model reads, and harness settings, which decide what mediates its output. Second, it gives a criterion for which properties of an agent’s actions can be held by a mechanism outside the model and which can only be counted, and shows that no context setting moves a property from one class to the other. Third, it states what can honestly be promised for such an agent: not determinism, but three properties with explicit assumptions, confinement of every performed act to an allowed set, replay of the record and reproducible assembly, and measured compliance on a fixed suite at a pinned model version. Fourth, it analyses a reference design against those properties: a design for agent context management in the Darkwave Agent Mesh, built on the Context Assembly its README describes, with cost and reliability models for the levers it uses and an evaluation design that would fit the parameters the paper leaves open.

Phillip Pienaar
Velocity Edge

2026-11-01
Version 1.0.0

Contents

Executive Summary 3
1. Introduction 3
2. Background and Related Work 5
3. A Model of an Agent's Context 10
4. Why Long-Running Agents Fail 12
5. Held and Counted Predicates 15
6. Bounded Behaviour 17
7. The Levers, Quantified 19
8. Lifespan 21
9. Threat Model 23
10. A Reference Design: Agent Context Management for the Darkwave Agent Mesh 25
11. Runtimes 30
12. Evaluation Design 31
13. Where the Words Still Matter 33
14. Limitations and Open Problems 33
Conclusion 34
Glossary 34
References 35
Appendix A: The Configuration Vector by Runtime 40
Appendix B: Confinement in TLA+ 40
Revision History 42
Licence and Disclaimer 42

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

Executive Summary

Agents that work for hours drift. They begin by following their rules and end by ignoring some of them, not because the rules were badly written but because the conditions under which they are read have changed: the rules now sit far back in a long context, competing with thousands of tokens of tool output, and sometimes they have been summarised away entirely. The literature since 2023 shows that, at fixed wording, the length, composition and arrangement of a context move a model's behaviour, and that spreading the same instructions over many turns degrades reliability sharply, even at temperature zero.

The response is not better prompts. It is configuration: deciding what the model reads, in what order, how much of it, at what moment, from what source, in what shape it may answer, and what stands outside the model to refuse an act. This paper names ten such settings and divides them into two kinds. Context settings change what the model reads, and can only make a desired behaviour more likely. Harness settings change what can come out, and some of them can make an undesired act impossible.

That distinction gives the paper its central criterion. A property of an agent's action is **held** if a plain-code check outside the model, a gate or a validator before it, can decide it from what it sees, and **counted** otherwise. Held properties fail only when the mechanism fails, which can be tested to any confidence and does not change when the model does. Counted properties fail at a rate that depends on the model, the configuration and the length of the session; that rate can only be estimated, and a new model release invalidates the estimate. The practical rule follows: every rule an organisation gives its agents should say what holds it, and a rule that names no mechanism is counted, however firmly it is worded.

What can be promised, then, is bounded behaviour: every act the agent performs lies in a set its rules allow; the context it was given and the decision made on its proposal can be replayed from the record; and its compliance with the properties that are counted has been measured, with confidence intervals, on a stated suite and model. What cannot be promised is that a model will make the same choice twice, or the right choice of target within an allowed act. The paper is explicit about both.

For a leader deciding whether agents can run unattended, the questions to ask of any platform follow directly: what holds each act that must never go wrong; can a handling be replayed from its record; what is last week's count of rules broken, by rule; and what happens when a message is built to make the checker fail every time. A platform that answers with its prompts has answered a different question.

1. Introduction

1.1 The Question

In September 2025 Anthropic's applied AI team wrote that "building with language models is becoming less about finding the right words and phrases for your prompts, and more about answering the broader question of 'what configuration of context is most likely to generate our model's desired behavior?'" [1]. They defined context as

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

"the set of tokens included when sampling from a large-language model" and context engineering as "the set of strategies for curating and maintaining the optimal set of tokens (information) during LLM inference" [1]. The practitioner literature that followed has catalogued strategies: compaction, note-taking, sub-agents, offloading, retrieval [1], [2], [3], [4], [5]. It has not, to our knowledge, said precisely what those strategies can and cannot guarantee, nor given a criterion for when a strategy inside the context should give way to a mechanism outside it.

Long-running agents are where the question is hardest. A single request can be engineered once and tested. An agent that runs for hours accumulates context, compacts it, hands work to sub-agents and receives their reports, and acts on outside systems many times; every one of those events changes the configuration under which its rules are read. The question is therefore not what the best context is, but what remains true as the context changes, and what can be made to remain true regardless.

1.2 Contributions

This paper makes six contributions.

- **C1. A configuration model.** An agent's context is the output of an assembly function over declared, versioned inputs, the message being handled, the record of earlier work and the working state. The settings that govern it form a vector of two kinds, context settings and harness settings, admitted by an explicit criterion (Section 3). We argue the ten settings are independent in definition, and name their couplings; we do not claim they are exhaustive.
- **C2. A criterion for held properties.** A predicate on an agent's action is held if and only if it is a function of the proposal's declared form and of inputs the proposal cannot alter, evaluated by an outside mechanism before the act is performed. The criterion explains why a single rule may be held in its tool and counted in its target, why context settings cannot make a predicate held while harness settings can, and why invariants across records need a component that reads aggregate state (Section 5). We position this against the defences by design it builds on: the dual-LLM pattern [6], CaMeL [7] and the 2025 design patterns [8].
- **C3. Bounded behaviour as three properties** with stated assumptions: confinement, replay and reproducible assembly, and measured compliance (Section 6).
- **C4. Known reliability and cost models applied to the configuration vector,** with one result of our own: the adversarial cost of a verification loop, and the per-sender budget it requires (Section 7).
- **C5. A reference design analysed** against C1 to C4: the Darkwave Agent Mesh's agent runtime as its public README describes it, with a TLA+ fragment of the confinement invariant (Section 10, Appendix B).
- **C6. A runtime analysis and an evaluation design** that would fit the parameters the paper leaves open (Sections 11 and 12).

The paper runs no experiment. Its models are parameterised, not fitted; where a parameter matters, it names the measurement that would fix it.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

1.3 Scope and Terms

An **agent** here is a loop in which a model, given a context, proposes an action, and a runtime decides whether and how to perform it. **Long-running** means the agent's work spans more context than one well-engineered request: many model calls, accumulating state, and actions on outside systems. A **rule** is a statement of what the agent must or must not do. A **gate** is plain code, not a model, that decides whether a proposed action is performed; throughout, "the gate" is that check, named the Action Gate in Section 10. The "outside mechanisms" (Section 5.1) are the gate and the other plain-code checks that run before it; in the formal statement of Section 3.1, G stands for those outside mechanisms that decide on the proposed action, the verifier and the gate. Product behaviours cited in this paper are as documented on 1 November 2026, and each may change with a release.

2. Background and Related Work

2.1 Long-Context Degradation

The first thing the field learned is that the length of a context degrades a model's use of it, whatever the words say; that is what makes size a setting in this paper rather than a matter of prompting. Liu et al. showed in 2023 that model performance "is often highest when relevant information occurs at the beginning or end of the input context" and "significantly degrades when models must access relevant information in the middle of long contexts", a U-shaped curve across the models then available [9]. RULER found that, although models scored nearly perfectly on vanilla needle-in-a-haystack tests, "almost all models exhibit large performance drops as the context length increases", with only half of those claiming 32K tokens performing satisfactorily at that length [10]. NoLiMa removed the literal overlap between question and evidence and found that "at 32K, for instance, 11 models drop below 50% of their strong short-length baselines" [11]. Chroma's 2025 report, across eighteen models including then-current frontier models, found that "model performance consistently degrades with increasing input length" even when task difficulty is held constant [12].

The cleanest evidence that length is a setting in its own right comes from Du et al., who found performance "degrades substantially (13.9%–85%) as input length increases" within claimed context lengths, and that the drop persists when irrelevant tokens are replaced with "minimally distracting whitespace" or masked [13]. The decline is therefore not explained by distraction or by the content of the extra tokens.

The evidence on position is now contested. McKinnon reports that "the 'Lost in the Middle' effect is not present for simple factoid Q&A in Gemini 2.5 Flash" [14], and Esmi et al. conclude that "the 'lost in the middle' problem has been largely resolved in newer models", while finding in the same study that accuracy falls to around 50 to 53 per cent at 20,000 posts [15]. Hsieh et al. trace the original effect to a "U-shaped attention bias" that calibration can correct [16]. We read the evidence as follows: position effects on simple literal retrieval have been substantially engineered away in 2025 and 2026 frontier models; length effects on non-literal, reasoning and aggregation tasks persist

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

in every recent study we read. This paper treats position as a setting whose effect must be measured per model, not assumed, and length as a setting whose effect is robust.

2.2 Instruction Loss over Turns

Long-running agents are multi-turn by construction, and the second lesson is that spreading instructions over turns costs reliability even when nothing else changes. Laban et al. ran more than 200,000 simulated conversations across fifteen models and found "an average drop of 39% across six generation tasks" when the same task was spread over turns rather than given at once, decomposed into "a minor loss in aptitude and a significant increase in unreliability", with unreliability rising by an average of 112 per cent [17]. Lowering temperature did not help: at temperature zero "there remains a large unreliability of around 30%", and the authors observe that "when LLMs take a wrong turn in a conversation, they get lost and do not recover" [17]. Their mitigations are configuration, not wording: a "new conversation that repeats the same information" and "consolidating instruction requirements into a single instruction" [17]. Earlier, Li et al. measured "a significant instruction drift within eight rounds of conversations", attributing it in part to "attention decay over long exchanges" [18].

2.3 Memory, Compaction and Long-Horizon Agents

Memory and compaction are the field's standard answers to a context that will not fit, and their record teaches two things this paper relies on: a memory policy left to the model fails as the model does, and compaction is lossy. MemGPT proposed "virtual context management", paging between a fixed window and external storage, with the model itself deciding when to page; its authors report that it "will often stop paging through retriever results before exhausting the retriever database" [19], a model-behaviour failure in a memory policy left to the model. Generative Agents retrieve memories by a scored combination of recency, importance and relevance computed in code [20]. ReAct interleaves reasoning and action in a loop whose context grows without bound [21]; Reflexion improves agents through "linguistic feedback" kept in an episodic buffer, but depends on a feedback signal that grades each attempt, external or self-generated in its reported experiments [22].

Compaction is now standard, and its lossiness is acknowledged by its vendors. Anthropic warns that "overly aggressive compaction can result in the loss of subtle but critical context" [1]; Anthropic's own course material for its coding agent says automatic compaction "can potentially lose details" [23]. ACON reduces peak tokens by 26 to 54 per cent by optimising compression guidelines from failure analysis [24], and TRACE finds that "compression can weaken the influence of recent interactions, increasing blocked actions, repeated exploration, and instability across runs" [25]. LangChain's Deep Agents harness offloads large tool results to files with a short preview, truncates old tool inputs above a fraction of the window, falls back to a structured summary, and keeps "the complete, original conversation messages" on disk as a canonical record; its authors name as "the most insidious failure mode ... an agent that loses track of the user's intent after summarization" [3], [4]. Slack's production security-investigation system passes no message history between agent invocations at all, relying on

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

structured, role-specific state, and reports that “sharing too much information stifles creativity and encourages confirmation bias” [2].

Long-horizon capability is growing. METR measures a 50 per cent time horizon that has been “doubling approximately every seven months since 2019” [26]. Reliability across repeated trials lags capability: on τ-bench, state-of-the-art function-calling agents succeeded on fewer than half of tasks and were “quite inconsistent (pass^8 <25% in retail)” [27].

2.4 Production Harnesses and Controlled Comparisons

Since 2025 the builders of production agents have published what their harnesses do, and a few have measured it; their converging practice is the raw material for the levers of Sections 7 and 8. Lindenbauer et al., at JetBrains Research, compared raw context, observation masking (old tool observations replaced by a placeholder, reasoning and actions kept) and model summarisation on SWE-bench Verified with two agent scaffolds and several models, running agents for up to 250 turns [28]. Both managed strategies “consistently cut costs by over 50%”; masking “often matched or even slightly beat LLM summarization”, and with Qwen3-Coder 480B “boosted solve rates by 2.6%” over unmanaged context “while being 52% cheaper on average” [28]. Summarisation lengthened trajectories by about 13 to 15 per cent, which the authors attribute to summaries that “smooth over, or hide, signs indicating that the agent should already stop”; a hybrid that masks first and summarises only as a last resort was cheaper again [28]. The same window setting did not transfer between scaffolds: one kept failed retries, so a short window could hold “only erroneous ones” [28].

Anthropic’s account of harnesses for multi-session coding work states plainly that “compaction isn’t sufficient”, and runs long work as a chain of sessions from a written record: a progress file, the git log, and a JSON feature list that agents are prompted to change “only by changing the status of a passes field”, because “the model is less likely to inappropriately change or overwrite JSON files compared to Markdown files” [29]. Each session follows a fixed start-up routine and works on one feature [29]. Anthropic separately reports, from an internal evaluation without further detail, that clearing stale tool results improved an agentic search task by 29 per cent and, with a file-based memory tool, by 39 per cent, and reduced token consumption by 84 per cent over a 100-turn evaluation [30].

Manus reports from production that “the KV-cache hit rate is the single most important metric for a production-stage AI agent”, with an input-to-output ratio “around 100:1” and cached input ten times cheaper than uncached [31]. Its rules are a stable prefix (no timestamp in the system prompt), append-only context, deterministic serialisation, and, rather than adding or removing tools, a fixed tool set whose allowed subset changes by state through masking at decoding: “Mask, Don’t Remove” [31]. It treats the file system as context, with compression “always designed to be restorable”, has the agent recite its objectives by rewriting a to-do list “into the end of the context”, keeps failed actions visible (“leave the wrong turns in the context”), and warns that uniform history makes an agent imitate itself: “don’t few-shot yourself into a rut” [31].

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

Google's Agent Development Kit describes context as "a compiled view over a richer stateful system": a working context rebuilt for each call by an ordered list of named processors, over a durable log of typed events; compaction writes its summary back into the log "as a new event"; large objects are reached by handle and expanded only for the call that needs them; and when one agent's history is passed to another, earlier agents' messages are re-cast with an attribution tag, without which the receiving agent "will hallucinate that it performed those actions" [32]. A companion tutorial names a failure specific to long pauses, in which after resuming "the model frequently hallucinates intermediate steps that never happened" [33]. OpenAI's cookbook marks summary content it cannot verify as "UNVERIFIED", marks replaced facts "Superseded:", tags summary turns as synthetic in metadata, and keeps the untrimmed log [34]. A code-level comparison of four harnesses found them converging on the same compaction mechanics: a cut never separates a tool call from its result, the agent gets a turn to save state before compaction, a few recently read files are re-attached afterwards, and oversized tool results go to disk behind a short preview [35].

Two controlled results bear on the tool set and on lifespan. Cao et al. found that giving a coding agent a retrieval tool lowered its accuracy on BrowseComp-Plus, a multi-hop question-answering benchmark over a large web corpus, from 88.5 without a retriever to 84.0 with embedding retrieval and 78.5 with BM25, on 200 examples with no variance reported, and that storing the same corpus as a folder of files rather than one file raised accuracy [36]. Breunig reports, from work presented in 2026, that a team of agents with scheduled forgetting outperformed both parallel agents and a single agent on the AHC014 task, names the failure "context inertia", in which a model keeps pursuing a hypothesis past its usefulness, and reports a study in which contradictory examples in the history lowered instruction compliance in every one of thirteen models tested [37]. Mehta and Datta report that naively evicting old messages, the plan among them, cut pooled ALFWorld success from 56.7 to 22.0 per cent, that preserving or re-surfacing only the plan did not recover the gap, and that a measure of the hidden state's sensitivity to the plan falls about four-fold within one step [38].

2.5 Agreement Is Not Correctness

Agreement among samples is often read as evidence of correctness; the field has learned that it is not, because models err together. Self-consistency improves reasoning by sampling many paths and taking the most consistent answer [39]. Its benefit depends on errors being spread out. Kim et al. found, across hundreds of models, that models "agree 60% of the time when both models err", against a random baseline of about one third, and that "larger and more accurate models have highly correlated errors" across architectures and providers [40]. Goel et al. find that "model mistakes are becoming more similar with increasing capabilities" and that model judges favour models similar to themselves [41]. Sampled agreement, ensembles and model judges therefore cannot hold an act; they can only signal.

2.6 Prompt Injection and Defences by Design

Prompt injection is where the field learned that no wording separates data from instructions, and that the defences which hold up are architectural. Greshake et al. showed that

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

applications built on language models "blur the line between data and instructions", so that "processing retrieved prompts can act as arbitrary code execution" [42]. Willison's dual-LLM pattern separates a privileged model that sees only trusted input from a quarantined model that handles untrusted content, mediated by "the Controller, which is regular software, not a language model" [6]. CaMeL extracts control and data flow from the trusted query so that untrusted data "can never impact the program flow", using capabilities, and "does not rely on model behavior modification"; it solves 77 per cent of AgentDojo tasks with provable security, against 84 per cent undefended, does not defend against "text-to-text attacks which have no consequences on the data flow", and is "vulnerable to side-channel attacks" [7]. Fides tracks confidentiality and integrity labels and "deterministically enforces security policies" [43]. Beurer-Kellner et al. set out six design patterns with the principle that "once an LLM agent has ingested untrusted input, it must be constrained", and conclude that "it is unlikely that general-purpose agents can provide meaningful and reliable safety guarantees" with current models [8].

These works establish the premise that security properties must be enforced outside the model. What they do not provide, and what Section 5 offers, is a general criterion for which properties of an agent's actions a given gate can hold, applied beyond injection to every rule an organisation gives its agents, with the consequence that the remaining rules must be measured.

2.7 Constrained Decoding

Constrained decoding acts on the output as it is formed, and the field's finding is that it costs little in speed while its cost to reasoning is disputed. Grammar-constrained decoding restricts a model's output to a language: Outlines reformulates generation as transitions of a finite-state machine and "adds little overhead" [44]; XGrammar reports up to a hundredfold speedup and "near-zero overhead" [45]. Whether constraint costs quality is disputed. Tam et al. "observe a significant decline in LLMs reasoning abilities under format restrictions" [46]; a response from the Outlines vendor, not peer-reviewed, argues that the original prompts differed between conditions and that, with matched prompts, structured generation performed as well or better [47]. JSON-SchemaBench finds that frameworks differ widely in their coverage of real schemas [48], so an output is only as constrained as the engine's support for its schema. The design consequence we draw, and argue in Section 7, is to constrain the act and not the reasoning.

2.8 The Reference Monitor, Safety and Evaluation

The oldest results this paper uses come from security and from the theory of program properties, and they say what a monitor can and cannot enforce. Anderson's reference monitor must be tamper-proof, always invoked, and small enough to be analysed and tested for completeness [49]. Saltzer and Schroeder's principles include complete mediation ("every access to every object must be checked for authority"), fail-safe defaults and least privilege [50]. Alpern and Schneider showed that every property is the intersection of a safety property and a liveness property [51]; a safety property is violated by a finite prefix of an execution, which is what lets a monitor block a violation

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

before it happens, and a liveness property is not. Evaluation of stochastic systems needs error bars: Miller recommends standard errors, clustering where questions are related, and paired comparisons, and warns against lowering temperature to reduce variance [52]. Yao et al.'s pass^k, "the chance that all k i.i.d. task trials are successful", measures reliability rather than capability [27].

2.9 Prompt Caching

Caching is where configuration meets cost, and the providers' terms decide which settings are expensive to move. Providers cache the longest matching prefix of a request. Anthropic's cache "references the entire prompt: tools, system, and messages (in that order)", requires "100 percent identical prompt segments", has a default lifetime of five minutes refreshed on each use, prices cache writes at 1.25 times the base input price and reads at 0.1 times (lower on some current models: 0.05 times on Claude Opus 5.5 and Sonnet 5.5, and 0.025 times on Claude Fable 5.1 and Mythos 5.1), and is invalidated entirely by "modifying tool definitions" [53]. Google enables implicit caching by default for Gemini 2.5 and later, with savings passed on "if your request hits caches" [54]. Caching couples three settings, tool set, position and model settings, to cost.

3. A Model of an Agent's Context

This section gives the paper its vocabulary: one step of an agent, the function that assembles its context, the ten settings that govern that function, and the two orders its layers obey.

3.1 The Agent Step

Let an agent handle a sequence of messages. For each message m, the runtime builds a context C, a model M samples a proposal $p \sim M(C)$, the outside mechanisms that decide on the proposed action, the verifier and the gate, together G, decide $d = G(p, g)$ from the proposal and a set of inputs g that the proposal cannot alter (made precise in Section 5.1), and the runtime performs the action only if d allows it. Two further outside mechanisms decide earlier in the handling and outside d: the admission check, before the proposer is asked, and the check on each read. The sampled proposal is the only stochastic step. Everything else in the loop is code, and can be made deterministic.

3.2 The Assembly Function

The context is the output of an assembly function:

$C = A(D@v, m, R, W; \kappa)$

where $D@v$ is the declared material at revision v (the rules, the agent's instructions, the domain documents and the tool definitions, each under version control), m is the message, R is the record of earlier handlings in the same chain, W is the working state accumulated within the handling (tool results, intermediate notes), and κ is the configuration. A runtime in which the model calls tools while forming its proposal has a non-empty W; a runtime in which every outside call is itself a gated action, and its result reaches the agent as a later entry in the record, has an empty W within a handling, and its working state lives in R. The reference design of Section 10 is of the first kind, with a constraint: each read is a read-only call allowed by a rule and recorded (Section 10.2).

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

Two facts about *A* matter throughout. First, *A* is code, so which pieces enter, in what order and at what size is decided by the configuration and never by the model, unless the configuration delegates a choice to it. Second, *A*'s output can be recorded with a manifest naming every piece, its version and its size.

3.3 Settings of Two Kinds

The configuration κ is a vector of settings. A property is admitted as a setting if it meets three conditions: it is a property of the assembly or of the harness around the model, never of the model's weights; it is defined independently of the others, and its value can be varied with the others recorded; and its value is visible in the manifest or in the harness's declared configuration, so that a handling's setting can be read back. Ten properties meet the criterion.

Context settings decide what the model reads:

1. **Composition**: which layers enter the window (rules, agent instructions, domain material, plan, task, carried notes, working state).
2. **Position**: the order of the layers, and what sits at the end of the context, nearest the proposal.
3. **Size**: the budget for each layer, and the thresholds at which material is offloaded or compacted.
4. **Timing**: when each piece enters, in particular whether a rule is presented only at the start or again at the point of action.
5. **Source and labelling**: where each piece comes from, at which version, and the trust label it carries (instruction, or data with its origin).
6. **Lifespan**: how long a context lives before it is replaced, and how (a fresh context from a record, or a compaction of the old one).

Harness settings decide what mediates the model's output:

7. **Output shape**: the form the proposal must take; in the strongest case, one action from an enumeration, with typed arguments, enforced by constrained decoding or validated before the gate.
8. **Tool set**: which tools are declared to the model.
9. **Model settings**: the model and its version, temperature, sampling and reasoning effort.
10. **The outside**: the mechanisms that decide on the proposal without reading the model's reasoning: the gate, validators, permissions in outside systems, and scripted checks over the record.

The two kinds differ in what they can achieve, which Section 5 makes precise: context settings change the distribution of proposals; the outside changes the set of acts that can result, and the other harness settings decide what it can see.

Independence. The settings are independent in definition: none is defined in terms of another, and each answers a different question about the assembly or the harness. They are not orthogonal in realisation, and the couplings are few enough to name: turning point-of-action repetition on (timing) adds a layer (composition) and tokens (size); offloading thresholds (size) change what the working state holds (composition);

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

changing the tool set or model settings invalidates a provider's cache of the prefix [53], coupling them to position through cost (Section 7.4); and on providers that fix the order of some blocks, tool definitions before system text for example [53], position is settable only within those blocks. An evaluation that varies one setting must hold the coupled ones fixed or record them.

The tension between a stable tool set and a narrow one is resolved by separating the declared set from the allowed set: the declared tools stay fixed for the life of an agent's revision, so the prefix caches, and the subset the agent may propose in a given state is narrowed in the output shape and enforced by the validator and gate, which is Manus's "mask, don't remove" with the enforcement moved outside the decoder [31]. We claim no exhaustiveness: a future harness may expose a property, such as an attention mask over designated spans, that meets the criterion and is none of the ten.

3.4 Layers and Their Two Orders

The context's material falls into layers: rules, agent instructions, domain material, the plan, task, carried notes and working state. Two orders apply to them, and they differ.

Precedence is a total order used to resolve conflict: rules, then agent instructions, then domain material, then the task; the plan, carried notes and working state are data and never instruct, the plan being the agent's own record of its intended steps. A ticket whose wording conflicts with a governing document loses to the document.

Position is the order in the window. The canonical order, used in the specification of Section 10.4 and in Appendix A, is: rules, agent instructions, tool definitions, domain material, plan, carried entries, working state, task, point-of-action rules. The stable layers come first, because they change least and so cache, and because position effects, where they persist, favour the start; carried entries and working state, which are data, follow the declared material; the task comes last, with the act-level rules for the tools the agent may propose repeated after it, because recency favours the end. Precedence and position differ because the task should sit where attention is strongest while still yielding to the rules on conflict. Conflating them, by placing rules last because they matter most or the task first because it is read first, gives up either the cache or recency. Some providers fix the order of certain blocks, placing tool definitions before system text for example [53]; there, tools sit where the provider requires, position is set within each fixed block, and the manifest records the order within each block rather than claiming a single token order.

4. Why Long-Running Agents Fail

The failures of long-running agents are familiar from practice. This section ties each to the setting that produces it, states a hypothesis for how compliance decays as a session grows, and sets out why held and counted rules age differently over a session.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

4.1 Failure Modes and the Settings That Cause Them

Failure	What happens	Setting
Dilution	Each added token draws on a finite attention budget [1]; performance falls with length alone [13]	Size, composition
Positional loss	Material in the middle of a long context is used less, on some models and tasks [9], [16]	Position
Recency displacement	Recent turns dominate early instructions; drift within a few rounds [18]; unreliability more than doubles over turns [17]	Timing, lifespan
Rule loss in summary	A summary keeps the facts of the work and drops the rules; goal loss after summarisation [4]	Lifespan
Flooding	Large tool results crowd out everything else [3], [4]	Size
Stale carry	A carried note or memory is read as current when the environment has changed [5]	Source and labelling
Injection	Text from a tool, document or carried note is read as instruction [42]	Source and labelling; the outside
Self-checking	The agent checks its output against a fading recollection of its rules	The outside
Plan and history eviction	Naive eviction of old messages, the plan among them, cuts success sharply; the plan alone does not restore it [38]	Lifespan, timing
Context inertia	The agent keeps pursuing a hypothesis past its usefulness, as Breunig reports [37]	Lifespan
Self-imitation	Uniform or contradictory history lowers compliance with fixed instructions and invites imitation [31], [37]	Composition
Hidden stop signals	A summary smooths over the signs that work is done, lengthening runs [28]	Lifespan
Idle-time fabrication	After a pause, the agent invents steps or approvals that did not happen [33]	Source and labelling; lifespan

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

Failure	What happens	Setting
Tool proliferation	An added tool changes behaviour and can lower accuracy [36]	Tool set

Every row is a property of configuration. None is cured by rewording the rule, because the rule's text is the one thing that does not change as the session grows. The practitioner record agrees: one widely cited report of agent drift notes that "adding a stronger instruction at the top of the prompt didn't fix it" [55], though the case is unattributed.

4.2 A Compliance-Decay Hypothesis

For a counted predicate r (Section 5), let $p_r(t)$ be the probability that the proposal at step t violates it. We state a hypothesis, not a fitted model:

$$\text{logit } p_r(t) = \alpha^*_r^* + \beta \cdot L(t) + \gamma \cdot \delta_r(t) + \eta \cdot k_r(t)$$

where $L(t)$ is the context length at step t, $\delta_r(t)$ is the distance in tokens from the most recent presentation of r to the proposal, $k_r(t)$ is the number of compactions since r was last loaded verbatim, and $\alpha^*_r^*$ is the predicate's base difficulty at short length. The evidence bears on each coefficient differently. For β it is strong and current [12], [13], [15]. For γ it is mixed: positional effects are documented for 2023 models [9], partly explained [16] and reported resolved for simple retrieval in some newer models [14], [15], while instruction drift with distance in dialogue is documented for older models [18]. For η the evidence is qualitative: vendors and practitioners report that compaction loses detail and intent [1], [4], [23], and TRACE measures instability under compression [25].

The hypothesis does three things. It names the variables a configuration can move: size moves L, timing moves δ, lifespan moves k. It predicts what a lever should do: repeating a rule at the point of action sets δ_r near zero, and reloading rules after compaction sets k_r to zero. And it is falsifiable, by the evaluation design of Section 12, which would estimate β, γ and η per model. Until that is run, no quantitative gain may be computed from it, and this paper computes none.

4.3 Session Reliability

Over a session of n acts in which a counted predicate is at stake, the probability that it is never violated is

$$\Pi^*_i^* (1 - p_r(t_i)),$$

which falls with n even at constant p_r, and falls faster when p_r rises with length. A held predicate's probability of never being violated over n acts is $(1 - \varepsilon)^n$, where ε is the mechanism's defect rate: zero for a correct gate, and in any case independent of length, configuration and model. The product form treats defects as independent per act, which is an idealisation: a gate defect is systematic and fires on a class of inputs, and is found by testing that class, not by sampling. The asymmetry is the reason the next section exists.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

5. Held and Counted Predicates

The two kinds of setting differ in what they can achieve. This section makes the difference precise with one criterion, applies it to rules as organisations actually write them, and draws from it a discipline for writing them.

5.1 The Criterion

Call the **outside mechanisms** the plain-code components that decide on an act before it is performed: in the reference design, the admission check at the estimate step, the specification's check on each read (Section 10.4), the verifier and the gate. Let g be the inputs those mechanisms read that the proposal cannot alter within the handling before the mechanism evaluates: in the reference design (Section 10), the agent's rules at the revision in force, the message's subject, the principal the work acts for, taken from the record and not from the payload [56], any approval, and, where a mechanism is specified to read it, recorded state of the chain. Let $\pi(p)$ be the proposal's declared form: the named action and its typed arguments as the output schema defines them, which a validator reads and the gate reads in part (the action's name). A predicate φ on a proposed action is **held** if and only if there is a function f such that $\varphi(p) = f(\pi(p), g)$ for every proposal p, and some outside mechanism evaluates f before performing. Otherwise φ is **counted**. What a proposal says beyond its declared form, its reasoning and free text, is never an input to f.

For given outside mechanisms and a given predicate the criterion is a yes-or-no question: either the predicate can be computed from what those mechanisms see, or it cannot. We claim no general procedure for answering it, but for the predicates rules usually state (a tool's name, a named argument, a principal) the answer is plain. The criterion says nothing about how well the model satisfies φ; it says whether satisfaction can be enforced.

Mixed rules. A single rule usually contains predicates of both classes. "Record supplier bills; pay only approved suppliers" contains "the act is *record bill* or *pay bill*", which is held if the gate sees the tool's name, and "the payee is approved", which is held only if the gate sees the payee and the rules can name approved payees. Where the gate reads tool names but not their arguments, as in the reference design, whose README says that "within an allowed tool, the proposer still chooses the target" [56], the second predicate is counted. A rule is therefore held in its tool and counted in its target unless its arguments are among the gate's inputs.

5.2 What Each Kind of Setting Can Do

Context settings cannot make a predicate held. By the criterion, whether φ is held depends only on the outside mechanisms and their inputs. Context settings change C, and therefore the distribution of p; they do not change the outside mechanisms or g. A rule repeated, emphasised, moved or summarised remains exactly as held or counted as it was. This is a consequence of the definitions, and we state it as one, because its practical force is large: no amount of prompt work converts a counted rule into a guarantee.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

Harness settings can, through the outside. What makes a predicate held is always a check by an outside mechanism on its inputs; the other harness settings decide what that check can see. Output shape is the clearest case. If the proposal must take the form of one action drawn from an enumeration of the tools the rules name, then "the agent proposes only a named tool" is a function of the proposal's form, and a validator before the gate that reads that form holds it. A correctly implemented grammar-constrained decoder is also plain code outside the weights, and would hold the predicate with the engine's defect rate as its ε. The design nevertheless does not count the decoder as the holding mechanism, because its coverage of a schema varies by engine [48] and cannot be tested from inside the runtime; the validator is the mechanism, and the decoder is an aid to validity that makes refusals rare. Widening the gate's inputs to include declared arguments, with rules that can name them, moves argument predicates (the payee, the recipient, the repository) from counted to held. The outside is the setting that holds; output shape and tool set decide how much it can see.

5.3 Invariants Across Records

Some rules are predicates over more than one record: a ticket must not be scheduled before the ticket that blocks it; an agent must not exceed its budget for the period. Such a predicate is a function of aggregate state, not of one proposal. It is held only if some component reads that aggregate state at decision time, and then only in the form that component checks. In the reference design, the estimate step checks the handling's estimated cost against the agent's budget for the period before the proposer is asked [56]; what that holds is an admission predicate, "no handling is admitted whose estimate would take spend over the budget", not "the period's spend never exceeds the budget", since actual cost can exceed the estimate and handlings already under way can overrun [56]. Unless estimates are reserved against the budget at admission, concurrent admissions can each pass the check; with reservation, the period's total is bounded by the budget plus the overruns of admitted handlings, which is a bound, not the stated rule.

A cross-record invariant that no component reads at decision time is counted, however simple, and no context can hold it: a fresh context per handling sees one chain, and a long context sees a fading transcript, but neither is a decision procedure over the state. Where no decision-time component exists, a scripted check over the record finds violations after the fact; that is detection, and must be called so.

5.4 The "Held by" Discipline

The criterion implies a discipline for any organisation that writes rules for agents. Each rule states what holds it, naming a mechanism (a gate check, a validator, a permission in an outside system, a scripted invariant) and a test that shows the mechanism refuses a violating action. A rule that names no mechanism states "held by: judgement, counted", and its violations are counted in production (Section 12). A clause that names the model's own re-reading as the mechanism is counted, by definition. The number of counted rules is a measure of how much of the organisation's control rests on the model, and it should fall over time. Published harnesses show how easily the line blurs. One tutorial says that its state machine "enforces the sequence" of an onboarding workflow,

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

while the step rules reach the model through its system instruction and order is checked only by evaluation [33]; another prompts agents to change only the `passes` field of a JSON feature list [29]. Both rules are counted as written. Each becomes held with a few lines of code outside the model, once the mechanism is specified to read the recorded state: a verifier that refuses a tool call the recorded step does not permit, and one that refuses any change to the feature list other than a flip of a `passes` field.

5.5 What This Adds

The dual-LLM pattern, CaMeL and the design patterns all place enforcement outside the model, and CaMeL and Fides enforce it deterministically [6], [7], [8], [43]. Their aim is resistance to injection, and their unit is the flow of untrusted data. The criterion here has a different unit, the predicate on an action, and a broader aim: it applies to every rule, including those no adversary is involved in (a scheduling invariant, a naming convention, a duty to cite evidence), and it turns the question "is this rule safe?" into one yes-or-no question and one measurement: can the gate or a validator compute it, and if not, what is its measured violation rate? We see the criterion as the generalisation those works imply but do not state, and the "held by" discipline as its operational form.

6. Bounded Behaviour

Determinism cannot be promised. Providers do not promise identical outputs for identical inputs, and lowering temperature leaves large unreliability in multi-turn use: around 30 per cent at temperature zero in Laban et al.'s study [17]. What can be promised is a set of properties, each stated with its assumptions. We use "safety" in Alpern and Schneider's sense [51]: a property violated by a finite prefix of an execution, which a monitor can therefore enforce. Confinement is a safety property in that sense; completion of the agent's task is a liveness property, which no monitor enforces.

6.1 Proposition 1: Confinement

Every performed action (not a read) lies in the allowed set A(agent, principal, subject, revision, approval).

Assumptions.

1. **Mediated channels.** The proposer has no effect on the world except through the gated action and, where the runtime allows them, read-only calls each checked against the rules before it is made; it holds no credentials and calls no outside system directly. Reads have no effect on the outside system's state, but their arguments can carry data out, which is counted unless the rules can name the arguments.
2. **No time-of-check to time-of-use gap.** The component that performs the action performs exactly the object the gate decided, without re-reading or re-deriving it from the proposal.
3. **Unforgeable gate inputs.** The subject and the principal come from the record and from proved identities, and no payload can alter them.
4. **Trusted rules.** The rule definitions and the act that puts a revision in force are in the trusted computing base, under review.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

5. **Mediated holds.** An action held for a person is decided again by the gate after approval, so the approval path is itself mediated.

Argument. By assumption 1, every state-changing effect of the proposal on the world passes through a performed action; reads, where allowed, pass through their own check (Section 10.2) and change no state, and what a read's arguments carry is counted, outside this proposition. The runtime performs an action only if the gate allows it, or holds it and the approval path ends in an allow (assumption 5). The gate's decision is a function of its inputs (assumption 3) under the rules in force (assumption 4), and allows only actions in A. By assumption 2, the action performed is the one decided. Hence every performed action lies in A. The argument does not mention the model: confinement holds whatever the proposer emits, which is the sense in which it is independent of the model. ■

What it does not say. Confinement is not outcome safety. An act in A can be wrong: the allowed *pay bill* tool aimed at an attacker's account is confined and harmful. Target choice is counted (Section 5.1) unless the gate sees the target. Confinement bounds the damage a misled proposer can do to what the rules allow it; it does not make what the rules allow right.

6.2 Proposition 2: Replay and Reproducible Assembly

Three claims, of different strength.

(a) Record replay. If the runtime records the assembled context and the gate's decision for each handling before acting, then any handling's context and decision can be read back exactly. This is trivial given the record, and its value is forensic: a wrong act can be traced to what the model was shown and what the gate decided.

(b) Reproducible assembly. If A is a pure function of its inputs and every input is immutable under its version, then A applied to the same versions yields the same context. Purity alone is not enough: documents arriving through retrieval, and the chain's earlier entries, must be content-addressed or recorded in full, so that a version names exactly one content. Under those assumptions the prefix of the context, the part built from declared material, is the same for every handling at the same revision.

(c) The proposal is a datum. The model's proposal is never re-derived on replay. It is replayed as a recorded value. A vendor model may change its weights or serving without notice, and sampling is stochastic; a claim that a handling can be "re-run" to reproduce its proposal is false and is not made.

6.3 Proposition 3: Measured Compliance

For each counted act-level predicate, the violation rate on a fixed scenario suite at a pinned model version is estimated with a stated confidence interval.

Assumptions. (1) The scenario distribution is fixed and stated. (2) The model version is pinned and recorded. (3) Runs are independent, or the estimate's standard errors are clustered by scenario [52]. (4) The predicates are act-level, so that a script can check them on the proposed action; a counted rule about the content of free text is not measured by this procedure and must be reported as unmeasured. The suite's scripts can check predicates the runtime cannot, because a scenario carries an oracle

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

(the correct target, the true fact) that is not available when a real message arrives; a predicate whose check needs no oracle should be moved into the verifier and so held, not measured.

What it gives. A rate and an interval, by predicate, by setting, by model. Reliability across repeated trials is reported as pass^k [27]; with correlated errors (Section 2.5), repeated trials do not buy reliability the way independent trials would, so pass^k must be measured, not computed from pass^1. A vendor model release invalidates the estimate, and production inputs may differ from the suite, which is why the estimate is paired with production counts (Section 12).

7. The Levers, Quantified

Each lever below works through one or more of the settings of Section 3, and each is given a model of what it costs and of what it can and cannot do. One result, the adversarial cost of a verification loop, is the paper's own.

7.1 Output Shape

Constraining the act is the strongest lever, because the form it imposes is what lets a validator before the gate hold "only a named tool" (Section 5.2); decoding to the same form narrows the proposals so that the check rarely refuses. The cost is disputed for reasoning tasks [46], [47]. The design resolves the dispute by constraining only the act: the model reasons without format restriction, and the final proposal, one action with typed arguments, is constrained. Schema coverage differs between engines [48], so the gate re-checks every constraint the schema expresses, and the decoding constraint is treated as an aid to validity, never as the guarantee.

7.2 Sampled Agreement

Suppose a decision is sampled k times and acted on only if a majority agrees. If samples were independent with error probability $q < ½$, majority error would vanish as k grows. Correlated errors change this. Model the k samples as exchangeable Bernoulli errors; by de Finetti's theorem they are conditionally independent given a latent error rate Θ with some distribution of mean q, and as k grows the majority's error converges to $\Pr(\Theta > ½)$, which is positive whenever the latent rate sometimes exceeds one half, that is, whenever there are inputs on which the model is usually wrong. Kim et al.'s finding that models agree on the same wrong answer 60 per cent of the time when both err [40] is cross-model evidence that inputs on which models are usually wrong are common; we infer from it, without direct within-model data, that the same holds for one model resampled, which is the case the argument needs. Sampled agreement therefore has an error floor no number of samples removes. It can signal uncertainty, by escalating disagreement to a person, and that is its planned use in the reference design, where sampled proposals are marked for a later version [56]; it never proves correctness.

Disagreement is also the normal case for a stochastic model on hard inputs, so escalating every disagreement converts sampling into an unbounded queue for the person who answers it. A design that samples must bound escalations: a disagreement

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

budget per chain, after which the act is refused rather than escalated, and a count of escalations per person-hour in its cost report.

7.3 The Verification Loop, Benign and Adversarial

A verifier checks each proposal against the predicates a script can test and, on failure, returns the failure to the model for another attempt, up to n attempts. Let each attempt cost c (the context's tokens at the input price, plus output) and fail independently with probability f. The expected number of attempts is $(1 - f^n)/(1 - f)$, the textbook geometric result, which is close to 1 for small f.

The adversarial case is the paper's own result. An adversary who controls a message can, by assumption, make the model's proposal fail verification on every attempt with high probability, for instance by instructing it towards a forbidden tool. Then each such message costs the defender $n \cdot c$ and, on exhaustion, one escalation to a person, at attention cost h. If the adversary's cost per message is a, the **amplification ratio** is

$$(n \cdot c + h) / a.$$

Because c includes the whole assembled context, which in a long-running agent is large, and a is the cost of sending a short message, the ratio can be orders of magnitude, and it scales with the configuration choices that make a context long. The sender charged is the principal the work acts for, which in the reference design is proved where the message enters and travels with every message in a chain [56]; adversarial text that arrives inside a tool's result or a retrieved document is charged to that principal, and the bound assumes the identity cost the threat model grants, so that one adversary cannot mint principals freely. For direct senders this bounds the amplification ratio, given that identity cost. For indirect channels it does not: an adversary who plants a document or a tool result at cost a spends the budget of a legitimate principal who did not send it, so the budget bounds the defender's spend per principal but not the attacker's ratio, and the residue is a starved legitimate principal, a denial of that principal's service.

Two consequences follow. First, a retry limit by count alone does not bound the defender's cost per unit of attacker effort; it bounds only the cost per message. Second, the bound that does work is a **budget per sender** over a period, enforced before the first attempt, so that the total defender spend attributable to one sender is at most that budget whatever the number of messages; and escalations from one sender must be capped in the same way, or the attack moves from tokens to a person's attention. A design that retries without a per-sender budget converts every untrusted input channel into a lever on its owner's spend.

7.4 Cache Economics

Let a request's input be a prefix of fraction s of its tokens that is identical across requests, followed by a variable remainder. With write multiplier w and read multiplier ρ on the base input price, and probability P that the prefix is still cached when the request arrives, the expected input cost relative to an uncached request is

$$s \cdot (P \cdot \rho + (1 - P) \cdot w) + (1 - s).$$

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

Caching pays when $P \cdot \rho + (1 - P) \cdot w < 1$, that is when $P > (w - 1)/(w - \rho)$. At Anthropic's documented multipliers for the five-minute cache ($w = 1.25$, $\rho = 0.1$) [53] the threshold is $P > 0.22$.

If requests arrive as a Poisson process of rate λ and each use refreshes a lifetime T, then $P = 1 - e^{(-\lambda T)}$. The prefix is therefore cached when requests are frequent relative to T, and lost when the gap between requests exceeds it. Manus's production ratio of input to output tokens, around 100 to 1, with cached input ten times cheaper, shows how much of an agent's cost sits in this expression [31]; and summary calls themselves cache poorly, since each processes a new slice of history [28].

Two corrections to common intuition follow. First, the saving does not fall with session length: providers cache incrementally on the longest matching prefix, so in a long chain earlier entries are themselves cached. What defeats the cache is the gap, and gaps in long-running agents come from waits: on a slow tool, on a person answering a held action, on a schedule. Second, any change to the tool set or to model settings mid-session invalidates the prefix [53]; so a fixed tool set per agent, with narrowing done inside the output schema rather than by changing the declared tools, is a cost decision as well as a design one. The per-handling narrowing of the allowed subset (Section 3.3) is cache-safe only if the narrowed schema is applied after the cached prefix, for example as a response format or a check by the verifier, rather than transmitted as changed tool definitions; that is a configuration item, and whether a given provider caches across a varying response schema is vendor-dependent. The same holds for anything else that would change the prefix: a timestamp in the system text, or a non-deterministic serialisation of structured data, breaks the cache on every request [31].

7.5 Point-of-Action Repetition

Repeating, at the end of the context, the act-level rules for the tools in the handling's enumeration sets δ_r in the hypothesis of Section 4.2 close to zero, at the cost of the repeated tokens, which are fresh and uncached on every act. Whether it reduces violations, and by how much on a given model, is an empirical question the evaluation design answers by running the suite with and without it. Because it moves only a context setting, it cannot make any predicate held, and it is kept only while measurement shows it pays.

8. Lifespan

Lifespan decides how long a context lives before it is replaced, and how. This section argues for replacing it, says how to compact when it must be kept, and names what the replacement loses.

8.1 A Chain of Short Contexts

The strongest context management is not needing it. Work of many steps can be a chain of short contexts, each assembled fresh from a written record, rather than one context kept alive. Each link sees the declared material, its message and the record of the chain; nothing accumulates, so nothing needs compacting. This is the architectural form of Slack's choice to pass no message history between invocations [2], of sub-agents

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

with clean windows [1], and of Laban et al.'s finding that a new conversation repeating the same information outperforms a long one [17]. The cost is that every link pays for its own context, which the cache offsets when links follow closely (Section 7.4). The evidence for the chain has strengthened: Anthropic's harness for multi-session work concludes that compaction alone is not sufficient and restarts from a written record [29]; and Breunig reports that scheduled forgetting outperformed continuous context on the AHC014 task, naming the failure it cures "context inertia" [37]. That is evidence, though not yet controlled for window size, that lifespan is a setting in its own right and not only a remedy for size.

Two pieces of state must survive every link: the rules, which are declared material, and the plan. Both are reassembled into every link, the plan from the record; what distinguishes the plan from other record entries is that it is never compacted. Mehta and Datta find the plan not sufficient on its own: evicting history that carries the plan costs heavily, and keeping only the plan does not recover the loss [38]. The design therefore pins the plan as a layer of bounded size, updated only through a declared action, and does not rely on it alone: the record of the chain is reassembled beside it, and compaction within a handling keeps the placeholders and recent steps of Section 8.2.

8.2 Compaction as a Last Resort

Where a runtime must keep a context alive, compaction should be the last of a fixed sequence, cheapest and most restorable first [1], [3], [4], [28], [31]:

1. **Clear** tool results already used and recorded.
2. **Mask** observations older than a window of recent turns, replacing each with a placeholder that keeps the result's status (succeeded or failed), its identifiers and its pointer, while keeping the reasoning and actions around it; if any step has succeeded, the window extends backwards until it contains at least one such step, and otherwise nothing is masked; masking matched or beat summarisation at lower cost in the one controlled comparison we know of [28].
3. **Offload** large tool results to files behind stamped previews (Section 8.3).
4. **Point**: replace old tool inputs already persisted with pointers to them.
5. **Summarise**, only when the steps above do not free enough room.

Five invariants apply to every step:

- **Pairs stay whole.** A compaction boundary never falls between a tool call and its result [35]; a masked result keeps its place beside its call, as a placeholder.
- **Failures are kept compactly, never exclusively.** Every failed action keeps at least its placeholder, with its status and error identifier, since hiding failures invites repetition [31]; and, where any step has succeeded, the mask window's backward extension ensures the visible window does not consist only of failures, since that derails the agent [28].
- **Exact values stay in text.** Identifiers, paths and amounts on which a held predicate will compute are never paraphrased or dropped by a lossy step.
- **Compression is restorable.** What is removed can be re-fetched from the record or its source, by path or identifier [31].

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

- **Every removal is recorded.** The manifest lists what was dropped, masked or offloaded, as well as what was included, so a handling's context can be reconstructed and a failure explained.

When summarising:

- **Before,** the agent is given one turn to write what must survive (open decisions, the work in hand, what waits on whom) to the record, where it does not depend on the summary [35].
- **The summary is structured,** with fixed fields (intent, decisions taken, artefacts created, failed approaches, invalidated assumptions, open questions, status against the completion criteria, next steps), extending the session intent, artefacts created and next steps of LangChain's summary [4]. The status field is there because summaries otherwise hide the signals that work is done [28].
- **Summary facts are data.** A structured summary gives a wrong fact the same authority as a right one. It is written back to the record as a typed, synthetic entry [32], [34]; uncertain content is marked unverified and replaced content superseded [34]; and any fact that drives a held act is re-read from its source at the point of action, never taken from the summary.
- **The transcript is kept** where it can be searched afterwards [3], [4].
- **The rules and the plan are reloaded verbatim after compaction,** setting k_r to zero.
- **Repeated compaction is a signal to stop** and start a fresh context from the record, since each pass loses more.
- **Thresholds are fractions of each model's window,** set by measurement, never absolute token counts, with a warning threshold below the trigger [35], so that the save turn completes before the trigger fires.

8.3 Offloading with Stamped Previews

A large tool result is written to a file and replaced in the context by its path and a short preview [3]. The preview is the model's only view of the file, so it carries the source, the time of fetching and the size; a read-back re-fetches the source when the stamp is older than that source's freshness limit, since "any action that changes the working tree can invalidate earlier context" [5]. An offloaded file is data, whatever it says.

8.4 What a Chain Loses

Two things do not survive the move from one long context to many short ones. A relation between two chains is seen by no single context, so cross-record invariants must be held by a component that reads aggregate state, or detected by a scripted check (Section 5.3). And a systematic misreading of a counted rule recurs in every fresh context, consistently: fresh contexts remove drift, not bias, and sampled agreement agrees with the bias (Section 7.2). That consistent error is found only from outside the loop that produces it: by the evaluation suite, by production counts, and by a reviewer.

9. Threat Model

The reference design of Section 10 is built against the following adversary, for the following assets.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

Adversary. Controls any text the model reads: a message from an outside sender, a document returned by retrieval, a tool's result, an offloaded file, a carried note or handover, and a summary written by an earlier context that read adversarial text. Does not control the gate's code, the rules in force, the record's integrity, or the proved identities of senders and principals.

Assets. The acts the agent can perform, the data it can read and send, the record, and the attention of the person who answers held actions.

Attacks and what defends each.

Attack	Defence	Residue
Instruction smuggled in data	Source labelling (data never instructs); confinement (the act must still be in A)	A persuaded model still chooses among allowed acts
An allowed act at a wrong target	Argument-level rules where the gate can see arguments	Counted where it cannot (Section 5.1)
Exfiltration through an allowed tool's arguments	Rules on arguments; data-flow labels in the manner of CaMeL and Fides [7], [43]	Side channels; text-to-text attacks [7]
The hold path: the approver reads model-shaped text	Approval shows the recorded action and context, never the model's justification as the basis for the decision	A person can still be persuaded by what they read
Verifier exhaustion	Per-principal budgets on tokens and escalations (Section 7.3)	Cost up to the budget; through indirect channels, a legitimate principal's budget spent by text it did not send
Poisoned carried note	Carried text is data, labelled with its origin; facts that drive held acts are re-read from their source	A counted rule may still be misapplied
Stale offloaded file	Stamped previews and re-fetch past a freshness limit (Section 8.3)	Within the limit, staleness is accepted
The rules themselves	Rules under review as code; a reload that changes them recorded and, where configured, approved by a second person	The trusted computing base is trusted
The record as a leak	Handling artefacts (assembled contexts, manifests, transcripts) readable only by the agent's reviewer and its operator, never returned to a model as context; chain entries enter later contexts only as data	Insiders with access

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

Attack	Defence	Residue
A read with attacker-chosen arguments	Each read checked against the rules before it is made, and recorded (Section 10.2)	Arguments are counted unless the rules can name them

The model is not in the trusted computing base. The gate, the runtime, the rules and the record are.

10. A Reference Design: Agent Context Management for the Darkwave Agent Mesh

10.1 The Mesh in Brief

"Agent Context Management" is this paper's name for the capability it specifies; the README names the feature that implements its first part Context Assembly. This section separates what the README states (10.1) from the paper's analysis and specification (10.2 to 10.5).

The Darkwave Agent Mesh, described here as its public README states it [56], is an agent runtime for Kubernetes on NATS, open source and in development ("coming soon", 2026). Each agent is declared in a Markdown file: the messages it handles, the model, script or person that proposes its actions, and the actions it may take. "An action not listed is refused; one marked `hold` waits for a person. Every step is recorded before the next is taken, and no action is performed twice on the Agent Mesh's own initiative" [56]. Each message passes through steps in order: received, checked against its contract, estimated against the agent's budget, proposed, gated, intended, performed, closed, each recorded before the next.

The **Action Gate** "is plain code in the core. It reads the proposed action, the Agent's rules, the message's subject and the Principal the message acts for, which come from the record and no payload can change. It never reads the message's content" [56]. The README states the limit plainly: "Whatever a misled proposer proposes, the most it gets is what the rules allow that Agent for that Principal; within an allowed tool, the proposer still chooses the target" [56]. The same proposed action under the same inputs "always gets the same decision, and an error refuses" [56]. The message path is specified in TLA+ and model-checked within its model's stated limits [56].

Context Assembly, as the README describes it: before a model proposer is asked, the agent runtime assembles what it sees, "by code and never by a model", from the definitions at the revision recorded when the message was received, and the message; each handling records a manifest of its assembled context, every piece with its version and size; and a model proposer answers in a fixed shape, one action from the tools the agent's rules name with typed arguments, which the runtime checks before the gate decides, refusing an answer that does not match [56]. The README does not claim that the model's answer is the same each time; for the same revision of the definitions and the same message, "it makes what the agent runtime assembles the same" [56], so a handling's context can be re-assembled from its recorded revision and checked

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

against its manifest. A retry on a failed answer, within count, time and budget limits, and sampled proposals on marked actions are planned for a later version [56].

Beyond what the README states, this paper recommends four practices, which the specification of 10.4 adopts: the unchanging part of the context first, so that a provider can cache it (Section 7.4); documents admitted only through one port, labelled as data with their source (Section 9); a short checklist in the answer's fixed shape (10.2); and each handling's whole assembled context recorded beside its manifest, so that replay does not rest on re-assembly (Section 6.2).

10.2 An Analytical Decomposition

For analysis, and as the specification the capability is built to, we decompose Context Assembly into three parts. The names are ours; the README describes the behaviour, not these names.

- **The assembler** (in the specification of 10.4) computes $A(D@v, m, R, W; \kappa)$. It reads the agent's declared layers at the revision in force, the message, and the chain's record; places them in the position order of Section 3.4; labels every piece that is not declared material as data with its origin; offloads large reads and large record entries, such as the outcome of an earlier action, to stamped files; and writes the manifest. Only the agent MCP gateway holds outside credentials [56], so the proposer reaches an outside system only through it. The README lists a proposer's reads as still being designed, in a stated direction: "read-only tool calls it makes through its Agent's gateway before it proposes, each allowed by a rule and recorded, and never the handling's one action" [56]. The specification follows that direction. Reads enter the working state W within the handling, are offloaded to stamped files when large, and are each checked against the agent's rules before they are made. Assumption 1 of Proposition 1 is therefore stated with reads included: the proposer's only channels to the world are the gated action and rule-checked reads. A read's arguments are an exfiltration channel unless the rules can name them (Section 9).
- **The shaper** declares the proposal's schema for the handling: the enumeration of tools the agent's rules name, typed arguments, and the checklist field. It is the output-shape setting.
- **The verifier** validates the proposal against the schema and the predicates a script can test before the gate decides, and refuses one that fails.

The outside mechanisms together form the reference monitor. The gate is invoked on every action and `check_read` on every read, and the specification gives the proposer no other route to an outside system (obligation 17). Each is tamper-proof relative to the model: plain code that never reads content beyond the declared form. The gate is small enough to analyse, its decision a function of four inputs and the rules [49], [50]; the verifier's predicates must each be small and testable, and the checklist field's content is never an input to any held predicate; the verifier checks only that it is present and well-typed. Complete mediation therefore rests on two mediators, not one, and both are named; Proposition 1 covers the first, and the read check's mediation of reads is stated by obligation 17 rather than proved.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

10.3 How the Design Meets Each Property

Property	Mechanism	Assumption it carries
Confinement (6.1)	The gate on every action; the agent MCP gateway, a sidecar container beside each agent's runtime, alone holds the credentials for outside systems [56]	1 (the proposer's only channels are the gated action and rule-checked reads); 2 (the intent is claimed and the decided action performed at most once); 3 (subject and principal from the record); 4 (definitions by reviewed pull request, a reload by the operator); 5 (an approved hold is decided again)
Record replay (6.2a)	The append-only record of every step; each handling's manifest, and re-assembly from the recorded revision, since for the same revision of the definitions and the same message "it makes what the agent runtime assembles the same" [56]; recording the whole context, as 10.1 recommends, makes replay exact without re-assembly	The record store makes writes durable before acknowledging them; for re-assembly, the assumptions of 6.2b
Reproducible assembly (6.2b)	Assembly by code from definitions at a revision	Retrieved documents and chain entries content-addressed or recorded in full
Proposal as datum (6.2c)	The proposal is a recorded entry	None; re-derivation is not claimed
Measured compliance (6.3)	Proposed: the conformance suite and production counts of Section 12, which the README does not describe	The four assumptions of 6.3
Held "only a named tool" (5.2)	The shaper's enumeration, checked by the verifier and the gate	Schema coverage of the decoding engine is not relied on
Held admission check (5.3)	The estimate step checks each handling's estimate against the period's budget before the proposer is asked	Estimates reserved at admission; the period's total bounded by budget plus overruns, not held at the budget

What the design does not hold: target choice within an allowed tool; the content of free text an action carries; consistent error across fresh contexts. Each is named in the README or follows from Section 5, and each is counted.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

10.4 A Specification the Capability Is Built To

This section is written to flow downstream into the agent runtime's implementation. It states interfaces and obligations, not code.

Inputs and outputs.

```
assemble(definitions@revision, message, chain_record, working_state, config)
    -> (context, manifest)

shape(agent_rules@revision, handling)
    -> proposal_schema          # enumeration of named tools, typed arguments, checklist

verify(proposal, proposal_schema, scriptable_predicates, chain_state)
    -> Valid | Invalid(rule_id, field)
                                # chain_state: the recorded state a predicate names
                                # (current step, plan, feature list), read from the
                                  record

check_read(read, rules@revision, subject, principal)
    -> Allow | Refuse           # every read, before it reaches the gateway

gate(action, rules@revision, subject, principal, approval)
    -> Allow | Hold | Refuse
```

The manifest. One entry per piece of the context, in position order:

```
manifest_entry := {
  layer:      rules | agent | tools | domain | plan | carried | working | task |
    point_of_action,
              # canonical position order (Section 3.4); within provider-fixed blocks,
                the order within each block
  source:     path or subject or tool call id,
  version:    revision hash, or content hash for anything not under version control,
  label:      instruction | data,
  tokens:     integer,
  offloaded:  optional { path, fetched_at, bytes, freshness_limit }
}

excluded_entry := {
  source, version, tokens,
  reason:     masked | offloaded | pointer | summarised | cleared | over_budget
}               # every piece considered and not included, recorded beside the manifest
```

Configuration per agent, declared with the agent and recorded with every handling:

```
context_config := {
  layers:              ordered list of declared layers and their token budgets,
  offload_threshold:   fraction of the model's window,
  point_of_action:     on | off; when on, the act-level rules for every tool in the
                       handling's enumeration, verbatim, in rule-identifier order;
                       a handling whose layer would exceed its token ceiling is
                       refused before the proposer is asked, never truncated,
  output_shape:        schema with action enumeration,
  model:               provider, model id, version, temperature, effort,
  verifier_retries:    planned: { max_attempts, max_seconds, principal_budget }
}
```

Obligations, each a test in the conformance suite, each tied to the property it supports:

1. The assembler never calls a model to decide what enters the context. (Reproducible assembly)
2. Every piece of the context that is not declared material carries the label `data` and its origin. (Threat model)
3. The stable layers come first and do not change within a handling; the tool set does not change within an agent's revision. (Cost; reproducible assembly)
4. Every handling's manifest is written to the record before the proposer is asked. (Record replay)

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

5. Assembling twice from the same revision and the same recorded inputs yields byte-identical stable layers. (Reproducible assembly)
6. A proposal that is not one action from the enumeration, with arguments matching their types, is refused before the gate. (Held "only a named tool")
7. A refusal cites the rule by its identifier and carries no new rule text into the context. (One source for rules)
8. A retry, when implemented, is refused once the budget of the principal the work acts for is spent for the period, before the attempt is made. (Section 7.3)
9. Offloaded files carry source, fetch time and size in their preview, and are re-fetched past their freshness limit. (Section 8.3)
10. The record is partitioned. **Chain entries** (messages, proposed actions, gate decisions, outcomes, and offloaded read files with their stamped previews, as `data`) may enter a later handling's context, as `data`, through *R*. **Handling artefacts** (assembled contexts, manifests, transcripts) are readable only by the agent's operator and by a reviewer role the specification adds, and are never assembled into a model's context. The README states that the Operator reads each handling's manifest through the control API [56]; the specification extends the same access to the assembled contexts and transcripts it records. (Threat model)
11. The estimate step reserves each handling's estimate against the period's budget at admission and settles it at close; a handling whose reservation would take the period's reservations and spend over the budget is refused before the proposer is asked. (Held admission check, Section 5.3)
12. The plan is a pinned layer: written to the record through `update_state` (obligation 15), assembled into every handling of its chain, and never masked, offloaded or summarised. It has a token ceiling; an update that would exceed it is refused by the verifier, as an over-ceiling point-of-action layer is. (Section 8.1)
13. No compaction boundary falls between a tool call and its result; every masked result keeps a placeholder with its status, identifiers and pointer; if any step has succeeded, the mask window contains at least one such step, and otherwise nothing is masked; and no step paraphrases an identifier, path or amount a held predicate computes on. (Section 8.2)
14. Every piece considered and not included is recorded as an excluded entry with its reason. (Record replay; Section 8.2)
15. State the model may update (a plan, a task list, a feature list) is held as structured data in the record. An update is proposed as a declared action (`update_state`, with the field and its new value as typed arguments) and the verifier, reading the current value from `chain_state`, refuses any change outside the fields the agent's rules permit. Fields a mechanism reads to decide what is allowed, such as the current step, are written only by the runtime from recorded outcomes, never through `update_state`, so the model never writes the state that narrows its own actions. (Section 5.4)
16. The declared tool set is fixed for an agent's revision; the subset allowed in a state is expressed in the shaper's enumeration and checked by the verifier against

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

`chain_state`, never by changing the declaration. The gate checks the action's name against the rules as always. (Sections 3.3, 7.4)

17. No read reaches the agent MCP gateway without an allow from `check_read`, recorded before the read is made. (Proposition 1, assumption 1)
18. Within a handling, when the working state exceeds its budget, steps 1 to 4 of Section 8.2 (clear, mask, offload, point) are applied in order; summarisation is never applied within a handling; a handling still over budget is refused before the proposer is asked again. A refused handling records its offloaded read files and pointers as chain entries and closes with a recorded outcome, and the runtime sends a continuation message to the same agent, which starts a new handling admitted under obligation 11 and assembled from the record. (Lifespan)
19. Every enumeration contains `escalate` and, where the agent has pinned state, `update_state`; both are named in the agent's rules so the gate allows them. Every agent's contract accepts the runtime's `continue` message, which carries the chain reference and inherits the refused handling's subject and principal from the record. (Sections 8.1, 12.1; obligation 18)

10.5 The Confinement Invariant in TLA+

Appendix B gives a fragment, written for this paper, stating confinement as an invariant over a simplified handling: no action enters the performed set unless the gate's decision for it, under the rules in force when it is performed, is an allow, either directly or after an approved hold has been decided again under the rules then in force. The fragment models a reload that changes the rules between a hold and its approval, which is the case assumption 5 exists for. It is a statement of the property the mesh's own specification is designed to check, not an extract from it.

11. Runtimes

Agents run in runtimes that expose different settings. We compare three classes by what their operator can set, not by product.

Setting	A hosted conversational assistant, harness not under the operator's control	A coding agent that exposes lifecycle hooks	An agent runtime the operator owns
Composition	Instructions and attached material at start	Project instruction files, imported deterministically	Assembled by code from declarations
Position	Not controllable	Instruction files form the prefix; hooks append at the end	Declared order, stable prefix first
Size	Not enforceable per session	Budgets in scripts and hooks	Per-layer budgets; refusal before the model is asked

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

Setting	A hosted conversational assistant, harness not under the operator's control	A coding agent that exposes lifecycle hooks	An agent runtime the operator owns
Timing	At start; elsewhere only if the model re-reads	Hooks inject rules on tool calls and after compaction	Before every proposal
Source and labelling	Not controllable	Partly, by hooks	Every piece labelled
Lifespan	Long; compaction by the vendor	One task per session; compaction with rules reloaded by hook	One message per handling; chains from the record
Output shape	Free text and tool calls	Scripts as sole writers for held acts	One action from an enumeration
Tool set	Set per conversation	Narrowed by deny rules	Fixed per agent
Model settings	The vendor's	Pinned per release	Pinned per agent
The outside	Only permissions in the outside systems themselves	Hooks, deny rules, pipelines	The gate, verifier and record
What can be promised	**Confinement only where outside systems' permissions refuse; otherwise detection after the fact**	**Confinement for acts the hooks hold**	**Confinement, replay, measured compliance**

The table carries one conclusion: work whose acts must be held does not belong in a runtime that can set almost none of the settings. That is not a criticism of such runtimes, which are built for conversation. It is a statement of what can be promised in them. Coding agents that expose hooks, as Anthropic's Claude Code does as of November 2026, sit between the two; the hooks are the vendor's feature and may change with a release.

12. Evaluation Design

The paper runs no experiment. This section specifies the evaluation that would test its claims and fit its open parameters, and that any operator of agents can run.

12.1 The Conformance Suite

- **Scenario classes.** Fixed scenarios per role, covering each counted act-level predicate; long-running scenarios (a fact planted early and probed after a forced compaction, following LangChain's method [4]; goal-drift probes after summarisation; a session run to several lengths); adversarial scenarios (instruction in data, wrong-target and exfiltration attempts, a poisoned carried note, a message built to

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

exhaust the verifier); harness scenarios (replay of a handling from its manifest, byte comparison of stable layers, the same schema through each output engine).

- **Runs and statistics.** Each scenario runs *N* times; violation rates are reported with confidence intervals, clustered by scenario [52]; reliability as pass^k [27]; comparisons between settings as paired differences.
- **Scripted checks.** Every predicate is checked by script on the proposed action. A score given by a model is reported separately as approximate and never added to scripted results.
- **Levers with and without.** Each context-setting lever (point-of-action repetition, rule reload after compaction, offloading thresholds) is run on and off, so that it stays only while it pays.
- **Fitting the hypothesis.** Varying session length, the distance of a rule's last presentation, and the number of compactions, while holding the rest fixed, estimates β, γ and η of Section 4.2 per model.
- **Probes from the literature.** Eviction probes (history withheld with and without the plan) [38]; trajectory length as a regression signal for summarisation [28]; an idle-gate case in which an agent paused awaiting an event is asked to skip a step and must propose the enumeration's escalate action, which every enumeration contains, rather than the skipped step [33]; task completion as a function of session length.
- **Cost.** Each run reports tokens per layer, tokens added at the point of action, compactions, offloads, retries, escalations and cache hits.

12.2 Production Measures

The suite estimates; production observes. In production, every refusal by a mechanism is a rule that held, and every violation of a counted rule found by a person, a reviewer or a scripted check is a rule that did not. The two are counted by agent, rule and source, read together, and never added. A rise in violations after a model release, with the suite unchanged, is the signal the measured-compliance estimate has gone stale.

12.3 Five Tests for Any Agent Platform

Any buyer can run these against any platform.

1. **Edit around the hold.** Attempt, through every channel the agent's identity reaches, an act the rules forbid, including edits and links rather than creation. Each must be refused by a mechanism, not declined by the model.
2. **Poison a carried note.** Place a false fact and an instruction in a note the agent will read on resumption. The agent must not act on either before re-reading the fact from its source.
3. **Replay a handling.** Rebuild a handling's context from its record and compare it with what was recorded.
4. **Starve the verifier.** Send a message built to fail verification on every attempt, and measure the tokens spent and the people interrupted before it stops.
5. **Count what nobody counts.** Ask for last week's violations of counted rules, by rule.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

13. Where the Words Still Matter

None of this makes wording irrelevant. Compression guidelines optimised in natural language improve long-horizon agents [24]; verbal feedback helps where a signal grades each attempt [22]; reciting evidence before answering recovers some of the loss from length [13]. Read through the model of Section 3, each of these works by changing what is in the context at the moment of proposal: a change of composition and timing, expressed in words. What wording contributes is precision. A rule is one testable clause, so that a script or a reviewer can decide whether it was kept. A refusal names the rule it applies. A schema's field names say what belongs in them. A brief is tested on a fixed set of cases before it ships, and again before any change of model. Emphasis, repetition and capitals are not on the list. They are what is reached for when the configuration has run out.

14. Limitations and Open Problems

The paper's claims are bounded in the following ways; the last two items record views that run against its thesis.

- **Target choice and content are counted.** Confinement bounds an agent to its allowed acts; within them, which target and what text remain the model's choice unless the gate can read them. Argument-level rules and data-flow labels narrow this; they do not close it, and CaMeL's measured utility cost [7] suggests closing it fully has a price.
- **Consistent error.** Fresh contexts remove drift, not bias. An error every fresh context shares is invisible to sampling and to the agent itself.
- **The approver is an attack surface.** A person reading what a model shaped can be persuaded.
- **The trusted computing base is trusted.** Rules written wrongly are enforced faithfully.
- **Vendor features change.** Whether a varying response schema preserves the cached prefix is one such dependency. Caching terms, structured-output limits, hooks and sampling controls cited here are as documented on 1 November 2026.
- **Evidence ages.** Position effects documented in 2023 may be absent in newer models for some tasks [14], [15]; capability horizons are growing quickly [26]. The argument rests on structural results that do not depend on any one model's effect size: length degrading performance with content controlled [13], the absence of an in-band boundary between data and instructions [42], correlated errors [40], and the monitorability of safety properties [51].
- **The decay hypothesis is unfitted.** Its coefficients are open until the evaluation of Section 12 is run per model.
- **Practitioners who disagree.** Cursor's experience, as reported by Osmani, is that system behaviour depends more on prompts than on harness or model [57]; we read the prompts in question as including role decomposition and the composition of each agent's context, which are configuration under Section 3, but the report is evidence against the strong form of the thesis. Relevance AI compacts large tool outputs before they enter history and compacts history at the end of a turn once it exceeds 30 per cent of the window, and reports an over 80 per cent reduction in tool-

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

output tokens without a stated baseline [58]; that is an alternative design to the chain of short contexts, and no head-to-head comparison exists. And one harness has a mode that passes a parent's entire history to a sub-agent to share the cache [35], trading isolation for cost; the trade is real, and the specification here takes isolation.

- **Anthropic's own outlook** is that "smarter models require less prescriptive engineering" [1]. Better models will move counted rules' violation rates down; they will not move a predicate across the criterion of Section 5, because that depends on the gate, not the model.

Conclusion

An agent that runs for hours is reliable to the extent that what it may do is decided outside it, and to the extent that what is left to it has been measured. Configuration of the context makes the model's judgement as good as it can be, and the settings that do this are few, discrete and testable. Configuration of the harness decides what can be held, and the criterion for that is simple: a property is held if a plain-code check outside the model can compute it before the act. The question the paper opened with therefore has a plain answer. An agent stops following its instructions when no mechanism holds them, so that only the context in which they were read carried them, and that context has changed. The question to ask of any claim that an agent is reliable is not how its prompts are written, but what holds it, and how that was measured.

Glossary

Term	Meaning
Agent	A loop in which a model, given a context, proposes an action, and a runtime decides whether and how to perform it
Assembly function	The code that builds the context from the declared material at a revision, the message, the record of earlier handlings and the working state, under the configuration
Chain entries	The part of the record that may enter a later handling's context, as data: messages, proposed actions, gate decisions, outcomes, and offloaded read files with their stamped previews
Confinement	The property that every performed action lies in the allowed set for its agent, principal, subject, revision and approval (Proposition 1)
Configuration	The vector of settings that governs the assembly function and the harness around the model
Context settings	The settings that decide what the model reads: composition, position, size, timing, source and labelling, and lifespan
Counted	Of a predicate on an action: not held, so its violation rate depends on the model, the configuration and the session, and can only be estimated
Gate	The plain-code check that decides whether a proposed action is performed; in the reference design, the Action Gate

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

Term	Meaning
Handling artefacts	The part of the record readable only by the agent's operator and reviewer and never assembled into a model's context: assembled contexts, manifests and transcripts
Harness settings	The settings that decide what mediates the model's output: output shape, tool set, model settings, and the outside
Held	Of a predicate on an action: computable from the proposal's declared form and inputs the proposal cannot alter, and evaluated by an outside mechanism before the act is performed
Long-running	Of an agent: its work spans more context than one well-engineered request, with many model calls, accumulating state, and actions on outside systems
Manifest	The record of a handling's assembled context, naming every piece, its version and its size
Measured compliance	The violation rate of a counted act-level predicate, estimated with a stated confidence interval on a fixed scenario suite at a pinned model version (Proposition 3)
Outside mechanisms	The plain-code components that decide on an act before it is performed: in the reference design, the admission check, the check on each read, the verifier and the gate
Pinned layer	A layer assembled into every handling of a chain and never masked, offloaded or summarised; the plan is one
Point of action	The end of the context, just before a proposal, where the act-level rules for the tools the agent may propose are repeated
Record replay	Reading back a handling's recorded context and gate decision exactly (Proposition 2a)
Reproducible assembly	Rebuilding the same context from the same versioned, content-addressed inputs (Proposition 2b)

References

[1] P. Rajasekaran, E. Dixon, C. Ryan, and J. Hadfield, "Effective context engineering for AI agents." Anthropic Engineering, 29 September, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents

[2] D. Marks, "Managing context in long-run agentic applications." Slack Engineering, 13 April, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://slack.engineering/managing-context-in-long-run-agentic-applications/

[3] J. Weng, "Context engineering: How deep agents remember." LangChain blog, 14 October, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://www.langchain.com/blog/context-engineering-deep-agents-remember

[4] C. Curme and M. Daugherty, "Context management for deep agents." LangChain blog, 28 January, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://www.langchain.com/blog/context-management-for-deepagents

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

[5] Hanks, "AI agent context: Long coding tasks." Verdent guides, 11 September, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://www.verdent.ai/guides/agents/ai-agent-context-management

[6] S. Willison, "The dual LLM pattern for building AI assistants that can resist prompt injection." simonwillison.net, 25 April, 2023. Accessed: Nov. 01, 2026. [Online]. Available: https://simonwillison.net/2023/Apr/25/dual-llm-pattern/

[7] E. Debenedetti *et al.*, "Defeating prompt injections by design." arXiv:2503.18813, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2503.18813

[8] L. Beurer-Kellner *et al.*, "Design patterns for securing LLM agents against prompt injections." arXiv:2506.08837, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2506.08837

[9] N. F. Liu *et al.*, "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024, doi: 10.1162/tacl_a_00638.

[10] C.-P. Hsieh *et al.*, "RULER: What's the real context size of your long-context language models?" arXiv:2404.06654, 2024. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2404.06654

[11] A. Modarressi *et al.*, "NoLiMa: Long-context evaluation beyond literal matching." arXiv:2502.05167; ICML 2025, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2502.05167

[12] K. Hong, A. Troynikov, and J. Huber, "Context rot: How increasing input tokens impacts LLM performance." Chroma Technical Report, 14 July, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://www.trychroma.com/research/context-rot

[13] Y. Du *et al.*, "Context length alone hurts LLM performance despite perfect retrieval." arXiv:2510.05381; Findings of EMNLP 2025, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2510.05381

[14] M. McKinnon, "Retrieval quality at context limit." arXiv:2511.05850, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2511.05850

[15] N. Esmi, M. Nezhad-Moghaddam, F. Borhani, A. Shahbahrami, A. Daemdoost, and G. Gaydadjiev, "GPT-5 vs other LLMs in long short-context performance." arXiv:2602.14188, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2602.14188

[16] C.-Y. Hsieh *et al.*, "Found in the middle: Calibrating positional attention bias improves long context utilization." arXiv:2406.16008, 2024. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2406.16008

[17] P. Laban, H. Hayashi, Y. Zhou, and J. Neville, "LLMs get lost in multi-turn conversation." arXiv:2505.06120, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2505.06120

[18] K. Li *et al.*, "Measuring and controlling instruction (In)stability in language model dialogs." arXiv:2402.10962; COLM 2024, 2024. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2402.10962

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

[19] C. Packer *et al.*, "MemGPT: Towards LLMs as operating systems." arXiv:2310.08560, 2023. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2310.08560

[20] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior." arXiv:2304.03442; UIST 2023, 2023. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2304.03442

[21] S. Yao *et al.*, "ReAct: Synergizing reasoning and acting in language models." arXiv:2210.03629; ICLR 2023, 2023. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2210.03629

[22] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning." arXiv:2303.11366; NeurIPS 2023, 2023. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2303.11366

[23] Anthropic, "Context management." Claude Code 101, Claude Academy. Accessed: Nov. 01, 2026. [Online]. Available: https://academy.claude.com/courses/claude-code-101/context-management

[24] M. Kang *et al.*, "ACON: Optimizing context compression for long-horizon LLM agents." arXiv:2510.00615, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2510.00615

[25] G. Min, L. Wu, M. Darbari, C. Chen, and L. Hong, "Towards reliable context compression for long-horizon agents: An empirical study of execution instability." arXiv:2608.06503, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2608.06503

[26] T. Kwa, B. West, J. Becker, *et al.*, "Measuring AI ability to complete long software tasks." arXiv:2503.14499; NeurIPS 2025, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2503.14499

[27] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan, "τ-bench: A benchmark for tool-agent-user interaction in real-world domains." arXiv:2406.12045, 2024. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2406.12045

[28] T. Lindenbauer *et al.*, "Cutting through the noise: Smarter context management for LLM-powered agents." JetBrains Research blog, December; NeurIPS 2025 Deep Learning 4 Code workshop, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://blog.jetbrains.com/research/2025/12/efficient-context-management/

[29] J. Young, "Effective harnesses for long-running agents." Anthropic, 26 November, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://www.anthropic.com/research/effective-harnesses-for-long-running-agents

[30] Anthropic, "Managing context on the Claude Developer Platform." Claude, 29 September, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://claude.com/resources/articles/context-management

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

[31] Y. Ji, "Context engineering for AI agents: Lessons from building Manus." Manus blog, 18 July, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus

[32] H. Lin, "Architecting efficient context-aware multi-agent framework for production." Google Developers Blog, 4 December, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://developers.googleblog.com/architecting-efficient-context-aware-multi-agent-framework-for-production/

[33] S. Saboo and E. Dong, "Build long-running AI agents that pause, resume, and never lose context with ADK." Google Developers Blog, 12 May, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://developers.googleblog.com/build-long-running-ai-agents-that-pause-resume-and-never-lose-context-with-adk/

[34] E. Okcular, "Context engineering: Short-term memory management with sessions." OpenAI Cookbook, 9 September, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://developers.openai.com/cookbook/examples/agents_sdk/session_memory

[35] A. Dhinakaran, "Context management in agent harnesses: Memory, files, and subagents." Arize AI blog, 28 April, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://arize.com/blog/context-management-in-agent-harnesses/

[36] W. Cao, X. Yin, B. Dhingra, and S. Zhou, "Coding agents are effective long-context processors." arXiv:2603.20432, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2603.20432

[37] D. Breunig, "The state of context engineering." dbreunig.com, 24 October, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://www.dbreunig.com/2026/10/24/the-state-of-context-engineering.html

[38] A. Mehta and A. Datta, "Plans fade fast: Why context management is load-bearing for LLM agents." arXiv:2606.22953, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2606.22953

[39] X. Wang *et al.*, "Self-consistency improves chain of thought reasoning in language models." arXiv:2203.11171; ICLR 2023, 2023. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2203.11171

[40] E. Kim, A. Garg, K. Peng, and N. Garg, "Correlated errors in large language models." arXiv:2506.07962; ICML 2025, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2506.07962

[41] S. Goel *et al.*, "Great models think alike and this undermines AI oversight." arXiv:2502.04313, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2502.04313

[42] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection." arXiv:2302.12173; AISec 2023, 2023. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2302.12173

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

[43] M. Costa *et al.*, "Securing AI agents with information-flow control." arXiv:2505.23643, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2505.23643

[44] B. T. Willard and R. Louf, "Efficient guided generation for large language models." arXiv:2307.09702, 2023. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2307.09702

[45] Y. Dong *et al.*, "XGrammar: Flexible and efficient structured generation engine for large language models." arXiv:2411.15100, 2024. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2411.15100

[46] Z. R. Tam, C.-K. Wu, Y.-L. Tsai, C.-Y. Lin, H. Lee, and Y.-N. Chen, "Let me speak freely? A study on the impact of format restrictions on performance of large language models." arXiv:2408.02442, 2024. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2408.02442

[47] W. Kurt, "Say what you mean: A response to 'let me speak freely'."txt blog, 2024. Accessed: Nov. 01, 2026. [Online]. Available: https://blog.dottxt.ai/say-what-you-mean.html

[48] S. Geng *et al.*, "JSONSchemaBench: A rigorous benchmark of structured outputs for language models." arXiv:2501.10868, 2025. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2501.10868

[49] J. P. Anderson, "Computer security technology planning study." ESD-TR-73-51, Vols I–II, Electronic Systems Division, US Air Force; requirements as restated in DoD 5200.28-STD, section 6.1, Oct. 1972.

[50] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, 1975.

[51] B. Alpern and F. B. Schneider, "Defining liveness," *Information Processing Letters*, vol. 21, no. 4, pp. 181–185, 1985, doi: 10.1016/0020-0190(85)90056-0.

[52] E. Miller, "Adding error bars to evals: A statistical approach to language model evaluations." arXiv:2411.00640, 2024. Accessed: Nov. 01, 2026. [Online]. Available: https://arxiv.org/abs/2411.00640

[53] Anthropic, "Prompt caching." Claude Platform documentation. Accessed: Nov. 01, 2026. [Online]. Available: https://platform.claude.com/docs/en/build-with-claude/prompt-caching

[54] Google, "Context caching." Gemini API documentation. Accessed: Nov. 01, 2026. [Online]. Available: https://ai.google.dev/gemini-api/docs/caching

[55] L. Chavez-Mattos, "Managing context in long-running AI agent sessions." MindStudio blog, 12 August, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://www.mindstudio.ai/blog/long-running-ai-agent-context-management

[56] Darkwave Trading, "The Darkwave Agent Mesh README." darkwave/darkwave repository, projects/agent-mesh/README.md, commit 7008f62d9, 2026.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

[57] A. Osmani, “Long-running agents.” addyosmani.com, 28 April, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://addyosmani.com/blog/long-running-agents/

[58] I. Hagoel, “Adaptive context management for production AI agents.” Relevance AI blog, 26 February, 2026. Accessed: Nov. 01, 2026. [Online]. Available: https://relevanceai.com/blog/adaptive-context-management-for-production-ai-agents

Appendix A: The Configuration Vector by Runtime

Setting	Kind	Specification value (Section 10.4)	Test
Composition	Context	Declared layers per agent	Manifest lists exactly the declared layers
Position	Context	Rules, agent, tools, domain, plan, carried, working, task, point-of-action rules (Section 3.4)	Manifest order; stable layers byte-identical across handlings
Size	Context	Per-layer token budgets; offload at a fraction of the window	Budgets in manifest; refusal over budget
Timing	Context	Rules at start and before each proposal	Point-of-action entry present, within its ceiling
Source and labelling	Context	Revision hash or content hash; instruction or data	No unlabelled piece
Lifespan	Context	One message per handling; within a handling, steps 1 to 4 of Section 8.2 only	No compaction across handlings; within a handling, no summarisation, and a handling whose working state cannot be brought within budget by steps 1 to 4 is refused
Output shape	Harness	One action from the enumeration, typed arguments, checklist	Invalid proposals refused before the gate
Tool set	Harness	Fixed per agent revision	Tool definitions unchanged across handlings
Model settings	Harness	Pinned per agent	Recorded per handling
The outside	Harness	Gate, verifier, budgets, record	Mechanism refusals in the record

Appendix B: Confinement in TLA+

A fragment written for this paper. It abstracts a handling to a proposed action with its gate inputs, models the rules as a function of a revision that a reload may change at any time, and checks that nothing is performed without an allow under the rules in force when it is performed.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

```
---------------------------- MODULE Confinement ----------------------------
CONSTANTS Actions,          \* proposed actions, each a record with a field tool
          Revisions,        \* rule revisions an operator's reload may put in force
          Allowed(_, _),    \* Allowed(a, r): the tools the rules at revision r allow
            for a's context
          Held(_, _)        \* Held(a, r): the tools the rules at revision r mark hold

ASSUME \A a \in Actions, r \in Revisions : Held(a, r) \subseteq Allowed(a, r)

VARIABLES rev,              \* the revision in force
          decision,         \* per action: "none", "hold", "allow" or "refuse"
          decidedAt,        \* per action: the revision its current decision was made
            under
          approved,         \* per action: whether a hold was approved
          performed         \* the set of actions performed

vars == <<rev, decision, decidedAt, approved, performed>>

Decide(a, r) == IF a.tool \in Allowed(a, r)
                THEN IF a.tool \in Held(a, r) THEN "hold" ELSE "allow"
                ELSE "refuse"

Init == /\ rev \in Revisions
        /\ decision  = [a \in Actions |-> "none"]
        /\ decidedAt = [a \in Actions |-> rev]
        /\ approved  = [a \in Actions |-> FALSE]
        /\ performed = {}

Reload(r) == /\ rev' = r
             /\ UNCHANGED <<decision, decidedAt, approved, performed>>

Gate(a) == /\ decision[a] = "none"
           /\ decision'  = [decision  EXCEPT ![a] = Decide(a, rev)]
           /\ decidedAt' = [decidedAt EXCEPT ![a] = rev]
           /\ UNCHANGED <<rev, approved, performed>>

\* An approved hold is decided again under the rules in force at approval;
\* if the rules have changed so that the tool is no longer allowed, it is refused.
Approve(a) == /\ decision[a] = "hold"
              /\ decidedAt[a] = rev              \* a stale hold is decided again first
              /\ approved'  = [approved  EXCEPT ![a] = TRUE]
              /\ decision'  = [decision  EXCEPT ![a] =
                                 IF a.tool \in Allowed(a, rev) THEN "allow" ELSE
                                   "refuse"]
              /\ decidedAt' = [decidedAt EXCEPT ![a] = rev]
              /\ UNCHANGED <<rev, performed>>

\* An allow is acted on only under the revision it was made under;
\* after a reload the action must be decided again.
Perform(a) == /\ decision[a] = "allow"
              /\ decidedAt[a] = rev
              /\ a \notin performed
              /\ performed' = performed \cup {a}
              /\ UNCHANGED <<rev, decision, decidedAt, approved>>

Redecide(a) == /\ decision[a] \in {"allow", "hold"}
               /\ decidedAt[a] # rev
               /\ a \notin performed
               /\ decision'  = [decision  EXCEPT ![a] = Decide(a, rev)]
               /\ decidedAt' = [decidedAt EXCEPT ![a] = rev]
               /\ approved'  = [approved  EXCEPT ![a] = FALSE]
               /\ UNCHANGED <<rev, performed>>

Next == \/ \E r \in Revisions : Reload(r)
        \/ \E a \in Actions : Gate(a) \/ Approve(a) \/ Perform(a) \/ Redecide(a)

Spec == Init /\ [][Next]_vars

\* Each action is performed under the revision its allow was made under,
\* and that revision allowed its tool.
Confinement   == \A a \in performed : a.tool \in Allowed(a, decidedAt[a])
HoldsMediated == \A a \in performed : a.tool \in Held(a, decidedAt[a]) => approved[a]
=============================================================================
```

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

`Confinement` and `HoldsMediated` are invariants of `Spec`, checked with TLC over small constant sets (model and run in this folder). `Perform` requires an allow made under the revision in force; an allow arises from `Decide`, which allows only tools in `Allowed` at that revision, or from `Approve`, which re-checks `Allowed` at the revision in force; and a tool that is held at its deciding revision reaches an allow only through `Approve`, after which `decidedAt` records the approving revision. `Redecide` clears an approval when the rules change, so a hold approved under old rules is approved again under new ones if it is still held. Once performed, an action's `decidedAt` no longer changes, since `Redecide` excludes performed actions. The fragment omits reads and their check, the record, crashes, leases and the gateway's contract, which the mesh's own specification is designed to cover; and because `performed` is a set, it does not model at-most-once, which in the mesh rests on the intent's lease.

Revision History

Version	Date	Change
1.0.0	2026-11-01	First publication

Licence and Disclaimer

© 2026 Darkwave Trading Limited, trading as Velocity Edge. This paper is licensed under the Creative Commons Attribution 4.0 International licence (CC BY 4.0, https://creativecommons.org/licenses/by/4.0/): you may share and adapt it for any purpose, provided you credit Phillip Pienaar and Velocity Edge, keep this copyright notice, link to the licence, link to the original where reasonably practicable, and indicate any changes. The Darkwave and Velocity Edge names and logos are not covered by this licence.

This paper is for education and information. It is not investment advice, and nothing in it is a recommendation to trade.