VELOCITY EDGE

CONTEXT ENGINEERING, PART 6

A Threat Model and a Reference Specification

ABSTRACT

What must an agent runtime do for each promise made of an agent to rest on a named mechanism? This part answers with a specification of context management for an agent runtime, written against a stated adversary. It is the last of six parts. It builds on the five before it: the model of context as the output of code; the hypothesis that, within a model's stated context length, a rule kept by reading whose breaches can be observed decays; the criterion for held and counted properties; the three promises; and the cost of the levers.

The adversary controls any text the model reads other than the rules and the agent's instructions, and none of the code that decides. The specification lets a model work in turns within the handling of one message. On each turn the model asks for one read, proposes one action or says it is done. The runtime sends it nothing to make it go on. Every read passes a check, and every action a gate, both plain code. Each result is recorded before the next turn. Twenty-eight obligations state what a runtime built to the specification must do. A table sets each promise beside the mechanism and the assumptions it rests on.

The specification is analysed, and a TLA+ fragment of its loop is model-checked with TLC at a small scale, within stated limits. It is not evaluated in use.

Phillip Pienaar
Velocity Edge

2026-11-19
Version 1.0.0

Contents

Executive Summary 3
1. The Question 3
2. The Adversary 3
3. Attacks, Defences and What Remains 4
4. The Loop 6
5. Three Components and Two Mediators 7
6. What Each Property Rests On 8
7. The Obligations 10
8. The Model Check 11
9. Limits 12
10. Related Work 12
Conclusion 12
Glossary 13
References 14
Appendix A: The Settings in the Specification 15
Appendix B: The Loop in TLA+ 16
Appendix C: The Specification in Full 18
Appendix D: The Model Files 22
Revision History 30
Licence and Disclaimer 30

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

Executive Summary

This part specifies the code that decides what an agent may do. Part 1's Conclusion states the series' thesis. For each of twelve attacks, the part gives the defence, or a defence recommended and not obliged, and what is left.

In the specification the model works in turns and decides when the work is done. Plain code decides every read and every action. Caps on turns, actions, cost and context size stop a handling that reaches one of them. They do not stop a handling that waits on a person's answer.

A TLA+ fragment of its loop is model-checked at a small scale, within stated limits. The specification has not been evaluated in use, and this series measures nothing.

1. The Question

No earlier part specifies a runtime. This part gives one answer, the series' own specification: one way to give each promise a named mechanism, under the assumptions Table 2 lists, and not the only one.

"Held", "counted" and "enforced" are used as Part 3 defines them. "Promise" is used only of Part 4's three.

2. The Adversary

The adversary controls any text the model reads other than the rules and the agent's instructions, which code places from a reviewed revision. That includes text carried from earlier work, and a summary written by an earlier context that itself read adversarial text.

The adversary does not control the code of the gate, the read check and the verifier, the rules in force, the integrity of the record, or the authenticated identities of senders and principals. That is assumed. Two obligations cover part of it: the record is append-only (27), and rules come only from a reviewed revision (28). No obligation keeps an allowed action or read from reaching the runtime's code, configuration, record or rules. The adversary cannot create authenticated identities freely. Principal and channel are used in Part 4's sense (Part 4, Section 3).

Four things are at stake: the acts the agent can perform, the data it can read and send, the record, and the attention of the person who answers actions on hold. "On hold" is Part 4's term.

The model is not trusted. The trusted computing base, in Part 4's sense, is here: the gate; the runtime with its verifier, read check and admission check; the gateway that alone keeps the credentials of outside systems; the rules; the record; and whatever establishes identities and authenticates approvers.

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

3. Attacks, Defences and What Remains

Table 1, the series' own analysis, lists the attacks considered, with the defence against each and the residue. A number in brackets, here and later, is an obligation of Appendix C.

Table 1: Attacks, the defence against each, and the residue.

Attack	Defence	Residue
Instruction smuggled in data (indirect prompt injection [1])	Every piece that is not declared material or the task is labelled data; every act must still be in the allowed set	A label helps only as far as the model was trained to respect it (Part 1, Appendix A.3). A persuaded model still chooses among allowed acts, on this turn and on later turns of the handling. The task itself may come from the adversary: it ranks below the rules (Part 1, precedence), and what it can obtain is bounded by the allowed set
An allowed act at a wrong target	Rules on arguments, where the gate or a declared check in the verifier reads and checks them	Counted unless the gate or the verifier reads and checks them (Part 3, Section 9)
Exfiltration through an allowed tool's arguments	Rules on arguments, where the rules can name them. Tags on data that policies check when a tool is called, in the manner of CaMeL and Fides [2], [3], would narrow this further and are not in the specification	Not prevented where the rules cannot name the arguments (Part 4, what confinement does not say). Where they can, side channels remain. Tags would not close every gap either. CaMeL's authors name side channels and, in real use, a person who confirms a call that breaks a policy; Fides's authors name implicit flows; and both designs rest on policies and labels being right, which is the series' own reading [2], [3]. Reads count too: a read's arguments leave the runtime
The approver reads text a model shaped	Recommended, not an obligation: approval shows the recorded action, its typed arguments, channel and principal, never the model's note or justification or text a read brought back	A person can still be persuaded, including by free text the model wrote into the arguments (Part 4, the approver)

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

Attack	Defence	Residue
Forced refusals and wasted turns	A budget for each principal on tokens and on escalations (Part 5, the budget for each sender, charged here to the principal); caps on the turns, actions and cost of one handling; the agent's budget bounds the total	Cost up to the caps and the budgets, the overruns of handlings under way, and what reads and actions themselves cost, which no cap covers. Through indirect channels, a legitimate principal's budget spent by text that principal did not send (Part 5). Handlings that stop before their first turn are not bounded in number, each of which may raise a problem for a person (16), and an action on hold blocks its chain until a person answers (Section 9)
Poisoned carried text	Text carried from earlier handlings, a model-written summary among it (Part 1, Appendix A.6), is data, labelled with its origin; neither the gate nor the read check reads its content. Of what earlier handlings left, the verifier reads only state that the runtime writes from recorded outcomes (18)	A counted property may still be broken. An adversary who shapes an outside system's response shapes the recorded state derived from it
Poisoned working note	The model's own notes are data whose content no check reads, and they never enter a later handling	A persuaded model can write a note that steers its later turns among allowed acts, since its own text is read back as input (Part 1, Appendix A.6). Text it writes into an action's arguments is recorded with the action and can reach a later handling as data
Stale file	A stamped preview, and a fresh fetch past a freshness limit	Within the limit, staleness is accepted
The rules themselves	Rules under review as code; a reload that changes them is recorded	A faulty or malicious change that passes review is not caught (Part 4, the trusted computing base)

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

Attack	Defence	Residue
The record as a leak	Assembled contexts, manifests and transcripts are readable only by the agent's reviewer and its operator, and never enter a later handling's context	Insiders with access. Chain entries still enter later handlings for the same principal (19), where the exfiltration row applies
A read with adversary-chosen arguments	Each read is checked against the rules before it is made, and recorded	Not prevented unless the rules can name the arguments (Part 4, what confinement does not say)
Relay through a second agent	Each agent's acts are confined to its own allowed set; entries carried between handlings are kept apart for each principal	A bound on a sequence of allowed acts across agents for one principal is counted unless an outside mechanism reads the whole sequence when it decides, and then held only in the form that mechanism checks (Part 3, Section 6)

4. The Loop

The specification lets a model work in turns within one handling, in Part 1's sense. On each turn the model gives one answer, of one of three kinds: one read, one action, or "done". The runtime performs what is allowed and records the result. It then assembles the next turn's context again from the record. A refused read, a refused action and a failed action are recorded and shown to the model on the next turn. This part calls each recorded result, as the model is later shown it, a report. A report widens Part 2's observation, a tool's return.

The model decides when the work is done. The runtime sends it no message to continue, and asks for a turn only when the one before is settled (10).

This shape is a design choice. Its two reasons are the series' own reasoning, not findings. First, a failed action becomes something the model sees and can adapt to inside the same handling. Second, the task does not move. On every turn, code places again the task (the message being handled, read under the declared rules) and every instruction. So the model is not relied on to carry any of them forward (Part 1, Appendix A.6), and only the working state grows.

The choice has a cost. One context reads untrusted text and then proposes acts over many turns, where a single call would read once and propose once. Some designs fix a plan before they read untrusted text (Part 3, Section 3). The loop gives up that order. A model steered by a read can choose every later allowed act of the handling. The series has measured neither side.

An answer that fails the verifier's tests is recorded as given and handled like a refusal. The specification's choice is one attempt for each turn. A failed answer comes back as

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

a report on the next turn, by its test's identifier and with no new rule text, within the turn cap. A turn whose answer is recorded is never asked again.

Each agent declares caps on turns, actions, cost and the size of a turn's context (16). At a cap the handling stops, and one problem is raised for a person.

The specification leaves to the model three things: whether it says "done" when the work is done and only then; whether it adapts well to a failure and does not repeat a refused act; and whether it follows text a read brought back. Where one of them is an act-level property a script can check on the proposed action, its violation rate can be measured as Part 4's third promise sets out. Otherwise it is reported as unmeasured (Part 4). The label on such text does not bound what following it can do. The read check and the gate do, by still deciding every later read and action.

5. Three Components and Two Mediators

Three components, named by the series, work around the model.

The **assembler** computes Part 1's assembly function from the agent's configuration, its declared material at the revision recorded when the message was received, the message, the record of earlier handlings of the same work, and the turns of this handling so far. It labels as data, with its origin, every piece that is neither declared material nor the task. Domain material is declared, and is labelled as data too. It writes a manifest for every turn before the model is asked. For the same revision and the same recorded inputs, a runtime built to the specification assembles the same stable entries, byte for byte (5). None of this makes the model's answer the same each time (Part 1, Appendix A.8). Declared material, layers, entries and the manifest are Part 1's terms.

The **shaper** declares the form of an answer for the handling: the three kinds, a list of the read tools and a list of the action tools the rules name for this message, and typed arguments. Whether a tool reads or acts is declared with it in the reviewed definitions, never taken from an outside system's description of it. The **verifier** checks each answer against that form and the tests a script can run, before the read check or the gate decides.

Two mediators stand between an answer, the model's only output, and an outside system. The **gate**, which Part 3 introduces, decides every action. The **read check** decides every read, allowing or refusing and never putting a read on hold. The runtime reaches outside systems only through one gateway, which alone keeps their credentials (24). A read is assumed, not checked, to change nothing outside the runtime. Many reads change something. A call declared to change something is an action, and is gated as one.

These mechanisms are meant as a reference monitor for access to outside systems only, with the requirements and the principle of complete mediation that Part 3, Section 2, sets out. The principle of complete mediation is wider than that. Access to anything else lies in the trusted computing base. That the mechanisms are always invoked is obliged (11, 24, 25). That they cannot be tampered with is assumed, and rests on the trusted computing base (Part 4).

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

Each mediator checks authority, identifying the principal a request acts for, not the text that prompted it. What that leaves open is the model's choice among allowed acts (Table 1). Each reads only the answer's declared form: the tool and its typed arguments. Besides that form, each reads only inputs whose values no proposal chooses, in Part 3's sense, the authenticated principal among them. The gate's approval comes from a person, who can be persuaded (Section 3).

The gate's decision is a function of the rules and four inputs: the action, the channel, the principal and any approval. A decision of that shape can be analysed and tested over its inputs. That argument is about a design, not a built gate, and covers the gate alone. Nothing here shows that the rule language, the read check, the verifier's tests or the gateway are small. No obligation requires the verifier's tests to be small. The trusted computing base is wider than the gate. Part 4's first promise concerns actions, which the gate decides. That the read check mediates every read is an obligation, not argued as a proposition.

An admission check, in Part 3's sense, stands beside the mediators. It is the reservation before the first turn. A check before each later turn compares the handling's recorded spend and the turn's estimate with its cost cap and the period's budgets (15).

6. What Each Property Rests On

Table 2, the series' own analysis, sets each of Part 4's promises, and two results of Part 3, beside the obligations it rests on and the assumptions it carries.

Table 2: Each property, the mechanism it rests on and the assumptions it carries.

Property	Mechanism	Assumptions it carries
Confinement (Part 4, first promise)	The gate's decision on each action (gate interface, Appendix C); no stale decision (13); one gateway, recorded allows and an exact performer (24, 25, 26); reviewed rules (28)	The seven assumptions of Part 4's Proposition 1. The specification meets four of them only in part: the first, complete mediation (11, 24, 25); the second, a performer that obeys the decision (26); the fourth, trusted rules (28); and the sixth, a gate always invoked (25). It meets the seventh, no stale decision, by (13), and adds (14). What remains stays assumed, among it a correct gate and the trusted computing base (Part 4)

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

Property	Mechanism	Assumptions it carries
Replay of the record (Part 4, second promise)	An append-only, durable record of every step (27); each turn's manifest and assembled context recorded before the model is asked (4); what was left out recorded (6); every answer recorded (9), and every decision and outcome recorded before the next turn (10)	The record's integrity against tampering, which rests on the trusted computing base (Part 4). Assembling again is not relied on
Reproducible assembly (Part 4, second promise)	Assembly by code from definitions at a revision and from recorded inputs (1, 3, 5). Identical output is obliged for the stable entries only; for the others the inputs are recorded (4, 6)	Assembly is a pure function of its inputs, and every input is immutable under its version (Part 4, Proposition 2(b)); retrieved documents and earlier entries are content-addressed or recorded in full
The proposal as a datum (Part 4, second promise)	Every answer is a recorded entry (9) in a durable record (27)	The record's integrity, as for replay. Deriving an answer again is not claimed: the same input can give another output (Part 1, Appendix A.8)
Measured compliance (Part 4, third promise)	Part 4's scenario suite and production counts, once run. This series runs neither	The assumptions Part 4 states for it
"Only a named tool" held (Part 3), extended here to the tool's declared kind	The shaper's two lists, checked by the verifier, then by the read check or the gate (7, 12)	The rules allow as a read only a tool declared read, and as an action only one declared to act; without that the verifier's check (7) is the only guard on kind. The decoding engine's coverage of the form is not relied on (Part 1, Appendix A.8)
The admission condition enforced (Part 3)	Each handling's estimate is reserved against the period's budgets before the model is asked, and each later turn's estimate is checked first (15)	The period's total is bounded by the budget plus the overruns of admitted handlings, not kept to the budget

By the series' judgement, three things the specification does not hold matter most: the choice of target within an allowed tool, unless a declared check in the verifier names the target; the content of free text an action carries; and a consistent error, one that every context assembled the same way shares. The first two are counted, by Part 3's criterion. The third is not a property of one action. It is the shared misreading that Part 4, Section 7, argues is found only from outside the loop that produces it.

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

Obligations cover the path from an answer to an outside system: one gateway alone keeps the credentials (24); no action reaches it without a recorded allow (25); the performer performs exactly that action (26); the record is append-only and durable (27); and rules come only from a reviewed revision, each reload recorded (28). The trusted computing base stays assumed (Part 4). The model check of Section 8 does not cover obligations 24 to 28.

By the series' reasoning, one message costs the defender at most its handling's cost cap plus the cost of one turn, in model calls of the agent that receives it. This bound leaves out what the handling's reads and actions cost. It also leaves out any call that returns no recorded answer. The specification does not bound the cost of that one turn, the last asked. Actual cost can pass an estimate. Part 5 analyses a message built to make every attempt fail. Here each failure uses a turn within the turn cap, and each turn must first fit the cost cap (15).

The agent's and the principal's budgets bound what the handlings of a period spend, up to the overruns of admitted handlings. They bound the number of handlings only where each spends something. Through indirect channels, by Part 5's reasoning, text can spend the budget of a principal who did not send it.

By the series' reasoning, a person is interrupted at most once for each handling that stops. Repeated invalid answers and refusals interrupt at most once for each agent and period. "Escalate" actions, which ask a person for help, interrupt up to a count for each principal and period. Each action on hold interrupts at most once under one revision of the rules, and the action cap bounds how many one handling has. Within a period and for a given number of reloads, the budgets and caps together bound the interruptions from handlings that spend something. A handling that stops before its first turn spends nothing. The specification does not bound how many of those a period has.

7. The Obligations

The specification has 28 obligations. No conformance suite for them has been written. Appendix C states them in full. Table 3 groups them.

Table 3: The obligations of Appendix C, grouped by what they support.

Supports	Obligations
Confinement	11 (every read checked), 13 (no stale decision), 24 (one gateway alone keeps the credentials), 25 (no action without a recorded allow), 26 (the performer acts exactly on the allow), 28 (rules only from a reviewed revision)
At most once	23 (each action claimed, and performed at most once on the runtime's own initiative)
Inputs unchanged between decision and act	14 (decisions serialised for each chain of work)

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

Supports	Obligations
Replay of the record	27 (append-only, durable record), 4 (manifest and context recorded before each turn), 6 (exclusions recorded), 9 (every answer recorded), 10 (each turn settled and recorded before the next)
The proposal as a datum	9 (every answer recorded as given, never sampled again)
Reproducible assembly	1 (no model in the assembler), 3 (stable entries first), 5 (byte-identical stable entries)
“Only a named tool”, extended to its declared kind	7 (the answer’s form checked first), 12 (declared tool set and kinds fixed)
Admission	15 (estimates reserved against the budgets)
Cost and attention, bounded for handlings that spend something	9 (no answer sampled again), 15 (spend checked before each turn), 16 (caps for a handling, refused actions included), 17 (escalations capped)
The threat model	2 (data labelled), 8 (a refusal carries no new rule text), 19 (record partitioned)
The model’s own text	18 (notes are data, their content read by no check, kept to their handling)
The loop	10 (each turn settled before the next; the model says when it is done; nothing is sent to continue it)
Lifespan	20 (Part 2’s first four steps within a handling, never compaction), 21 (what those steps must keep), 22 (stamped previews)

8. The Model Check

TLC found no error in the seven properties of a TLA+ module of one handling’s loop, at the sizes checked. The smallest instance has four tools, two revisions, three turns and a cap of two actions. Three larger instances were also checked. The seven are safety properties: they say only that nothing bad happens. The result says nothing of larger sizes, of other rules, or of any runtime built to the specification.

The module covers one handling, not a chain of them or several agents. Its rules are sets of tool names, so it says nothing of arguments, channels or principals. It omits the record, crashes and recovery, admission, budgets, failed actions, the cost and context caps, and the content of notes and reports. It takes the comparison of revisions and the act as one step, which a runtime must make true. It cannot show that an action is performed at most once.

Appendix B describes the module and every run.

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

Whether a handling ends was checked only at the smallest size, with a person always able to answer. Under weak fairness on every step but a reload, TLC finds it violated. Under strong fairness on the steps a reload can disable, TLC found no error.

9. Limits

No runtime built to the specification has been measured. Its own analysis has had no outside review. The language its rules are written in is not specified. Every defence on arguments depends on what that language can name.

Whether working in turns inside one handling beats a fresh context for each step is an empirical question the series does not answer. The caps that stop a handling are numbers to be set by measurement for each model.

A turn whose action is on hold cannot settle until a person answers. The specification sets no expiry, so the handling, and by obligation 14 its chain of work, wait until then.

The scope is the one Part 1 gives: agents built on transformer-based language models. By the series' reasoning, the mediators do not depend on the model's architecture. The cost reasoning does (Part 1, Appendix A.4 and A.7).

10. Related Work

Some defences rely on the model to tell instructions from data, among them the instruction hierarchy and spotlighting (Part 1, Section 5, and Appendix A.3) [4], [5].

Others work in code outside the model: CaMeL, Fides and the design patterns, which Part 3, Section 3, sets out with their limits [2], [3], [6]. Progent checks every tool call against a policy of rules over tool names and arguments (Part 3, Section 2) [7].

AgentDojo's measurements of injection on agents, and the bypassing of recent defences by an adversary who adapts, are set out with their limits in Part 1, Appendix A.3, and Part 3, Section 3 [8], [9]. This specification has not been evaluated on AgentDojo.

Where this part sits is the series' own reading. Its data labels, like spotlighting, work only as far as the model respects them (Table 1). Like Progent, it checks each call against rules in plain code. Progent's policy can be written and updated by a model during the run, and widened only with explicit approval. Here the rules come only from a reviewed revision (28). The specification does not track information flow. So it does not close what CaMeL and Fides aim at (Table 1).

Conclusion

Part 1's thesis is that an agent that runs for hours can be relied on 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.

What an agent may do is decided outside it when four conditions are met: the gate and the read check stand between every answer and every outside system; they read only the answer's declared form and inputs whose values no proposal chooses (Part 3); the verifier beside them reads, beyond the answer's declared form, only state the runtime

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

writes; and each allow is acted on only under the revision it was decided under, and only on the recorded state of its chain that it was decided on (13, 14). An action on hold waits on a person, who can be persuaded (Part 4).

The check outside matters because a rule left to the model can fail in three ways. Within a model's stated context length, a rule kept by reading whose breaches can be observed may decay, as Part 2 hypothesises (Part 1, Appendix A.4 and A.5). The model can be talked out of it, because instructions and data reach the model as one stream, which training only partly separates (Part 1, Appendix A.3). The same input can give a different output (Part 4, Section 2; Part 1, Appendix A.8). A check in plain code need not have these three weaknesses. It can still be wrong (Part 4).

What is left to the agent includes its choice among allowed acts, its choice of target where no declared check names one, its text, and its judgement of when the work is done. Where it is an act-level property a script can check on the proposed action, its violation rate can be measured as Part 4's third promise sets out. The content of its text is not measured by that procedure, and is reported as unmeasured. This series measures none of it.

The question to put to a runtime is the same for every rule and every promise: which mechanism, reading which inputs, at which step, and what was measured of the rest.

Glossary

Term	Meaning
Adversary (this part's threat model)	The party the specification is written against: it controls any text the model reads other than the rules and the agent's instructions, and none of the code that decides
Answer	What the model gives on one turn: one read, one action, or "done", with its note
Assembler	The component that builds each turn's context by code and writes its manifest
Chain of work	The handlings, across messages, that read and write the same recorded state; their decisions are serialised (14)
Note	The model's own text about its plan and next step, recorded with its turn, labelled data, its content read by no check, and kept to its handling
Obligation	One thing a runtime built to the specification must do, stated so that it can be tested or, for some, reviewed
Period	The span over which the agent's and each principal's budgets, and the escalation cap, are counted (15, 17)
Read check	The plain-code check that allows or refuses every read before it is made
Reload	Part 4's term (Part 4, Section 3), used in its sense. Each reload is recorded (28)

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

Term	Meaning
Report	The recorded result of an earlier turn as the model sees it on a later one: a read's result, an action's outcome, a refusal or a failed test. It widens Part 2's observation
Residue	What a defence does not prevent
Settled	Of a turn: its read is refused or its result recorded, its action is refused or its outcome recorded, or its answer was invalid and recorded
Shaper	The component that declares the form an answer must take for a handling
Specification	The series' own statement of what an agent runtime must do; this part's
Turn	One request to the model within a handling, and its one answer
Verifier	The specification's plain-code check of each answer against its declared form and the tests a script can run, before the read check or the gate decides. A validator in Part 3's sense

References

[1] 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 (v2, 5 May 2023), 2023. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2302.12173

[2] E. Debenedetti *et al.*, "Defeating prompt injections by design." arXiv:2503.18813 (v2, 24 June 2025), 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2503.18813

[3] M. Costa *et al.*, "Securing AI agents with information-flow control." arXiv:2505.23643 (v2, 3 September 2025), 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2505.23643

[4] E. Wallace, K. Xiao, R. Leike, L. Weng, J. Heidecke, and A. Beutel, "The instruction hierarchy: Training LLMs to prioritize privileged instructions." arXiv:2404.13208 (v1, 19 April 2024), 2024. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2404.13208

[5] K. Hines, G. Lopez, M. Hall, F. Zarfati, Y. Zunger, and E. Kiciman, "Defending against indirect prompt injection attacks with spotlighting." arXiv:2403.14720 (v1, 20 March 2024), 2024. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2403.14720

[6] L. Beurer-Kellner *et al.*, "Design patterns for securing LLM agents against prompt injections." arXiv:2506.08837 (v3, 27 June 2025), 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2506.08837

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

[7] T. Shi *et al.*, "Progent: Securing AI agents with privilege control." arXiv:2504.11703 (v3, 14 May 2026), 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2504.11703

[8] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents." arXiv:2406.13352 (v3, 24 November 2024); NeurIPS 2024 Datasets and Benchmarks Track, 2024. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2406.13352

[9] M. Nasr *et al.*, "The attacker moves second: Stronger adaptive attacks bypass defenses against LLM jailbreaks and prompt injections." arXiv:2510.09023 (v1, 10 October 2025), 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2510.09023

Appendix A: The Settings in the Specification

Part 1 is the home of the settings. Table 4 gives the value the specification takes for each. Its position order departs from Part 1's in three ways. It has no plan entry, since the model's notes stay within the handling. In place of Part 1's carried notes it has carried entries, which are the chain entries of obligation 19. It places the task before the working state. By the series' reasoning, the entries up to the task then stay the same across the turns of a handling. Part 1, Appendix A.7, says why a fixed start matters to the cache. Part 1, Appendix A.5, gives the position studies consistent with repeating rules at the point of action. Those studies concern evidence and style instructions, not rules.

Table 4: The value of each setting in the specification, and its test.

Setting	Specification value	Test
Composition	Declared layers for each agent	The manifest lists exactly the declared layers
Position	Rules, agent instructions, tool definitions, domain material, carried entries, task, working state, point-of-action rules	Manifest order; stable entries byte-identical across turns
Size	A token budget for each layer; a cap on a turn's whole context	Budgets in the manifest; a stop over the cap
Timing	Every instruction placed again on every turn (Part 1, Appendix A.5 and A.6); act-level rules, in Part 1's sense, again before each answer where that is turned on	The point-of-action entry present, within its ceiling
Source and labelling	A revision hash or a content hash; instruction or data, domain material always data	No unlabelled piece
Lifespan	One message for each handling; turns within it; no compaction within a handling	No turn after "done" or a stop; no piece compacted within the handling in any manifest

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

Setting	Specification value	Test
Output shape	One answer of three kinds, from two enumerations, with typed arguments	Invalid answers never reach a mediator
Tool set	Fixed for an agent's revision, each tool's kind declared	Tool definitions unchanged for a revision
Model settings	Pinned for each agent (Part 1, Appendix A.1 and A.8)	Recorded for each handling
The outside	Gate, verifier, read check, admission check	Mechanism refusals in the record

Appendix B: The Loop in TLA+

The fragment was written for this part. It abstracts one handling to its turns, models the rules as a function of a revision that a reload may change at any step, and checks seven properties. On each turn the model may ask for any tool as a read, propose any tool as an action, say it is done, or give an invalid answer. A reload may change the rules at any step, including between a decision and its act, and a person may approve or refuse an action on hold.

The module is the file `LoopConfinement.tla`, with its model, `MCLoopConfinement.tla`, and its configuration, `MCLoopConfinement.cfg`. Appendix D prints them, and every other module and configuration kept with them, from the files TLC checked. Nineteen more configurations are not printed: a script derives each from a printed one. Eighteen name only the invariant or property their run tests; the nineteenth leaves out `ActionsBounded` for one mutant's second run. The module's seven properties are named `StrongConfinement`, `ReadsChecked`, `KindsKept`, `SettledBeforeNext`, `NothingAfterDone`, `ActionsBounded` and `StopStands`, in the order stated below. `Held` in the module means marked hold in the rules, not held in Part 3's sense.

The runs used TLC2 Version 2.19 of 08 August 2024 (rev: 5a47802), as TLC's own version line in `tlc-run.txt` records; the digest also gives the SHA-256 of the tla2tools.jar used. The model has two action tools, `t1` and `t2`, and two read tools, `r1` and `r2`; three turns; a cap of two actions; and two revisions. Revision `v1` allows `t1` and `t2`, marks `t2` hold, and allows both reads. Revision `v2` allows `t1` only, marks it hold, and allows `r1` only. `Spec` satisfies the type invariant and the seven properties, with 82,418 states generated, 27,784 distinct states found and no error.

The seven properties are these. Every action is performed under the rules in force at the step that performs it, and a tool those rules put on hold has been approved. Every read is made under an allow of the rules in force at that step. Only a tool declared read is ever read, and only one declared to act is performed. No turn is asked before the turn before it is settled. Nothing is asked after "done". No more actions are decided than the cap allows. Once a handling has ended, nothing of it changes.

Three larger instances, each with its model and configuration in files named for it (`MCBig`, `MCRules` and `MCThree`), satisfy the same invariant and properties with no error. `big` adds

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

a fourth turn and a third revision, `v3`, that allows nothing: 6,975,705 states generated, 1,785,456 distinct. `rules` takes as its revisions every rule table over the two action tools, nine in all, with three turns and a cap of two actions: 73,438,983 generated, 7,429,806 distinct. `three` has three action tools, four turns and a cap of three actions: 2,659,882 generated, 902,948 distinct; one more run there shows that a handling can perform three actions.

Eleven invariants that deny a path are each violated, so each path is reached: two actions performed in one handling; a read and then a performed action; a performed action after a refused one; an approved action performed; a further answer after an approved action is performed; "done"; a stop; a stop with an action proposed past the cap and left undecided; a stop at the cap on turns; a refused read; and a hold that a person refuses.

The module `Redecide.tla` has seven more: three invariants and four action properties, each denying a path. Each is violated, which shows that a decision is made again after a reload. The paths are these: an action and a read each performed after being decided again; an allow that becomes a refusal, and one that becomes a hold; a read's allow that becomes a refusal; an approval withdrawn; and an action performed on a second approval.

Seven mutants each remove one guard. A `Perform` without `decidedAt[j] = rev` violates `StrongConfinement`. A `MakeRead` without it violates `ReadsChecked`. An `Approve` without it violates `StrongConfinement`. A `NextTurn` that does not require `Settled(turn)` violates `SettledBeforeNext`. An `AnswerAct` that never stops at the cap violates `ActionsBounded`. An `AnswerDone` that leaves the handling open violates `NothingAfterDone`. A `Gate` that still decides after the handling has ended violates `ActionsBounded`. Run again with `ActionsBounded` left out, the same mutant violates `StopStands`. `KindsKept` has two halves, and each has two guards. For reads they are the assumption that the rules allow as a read only a tool declared read, and the check of an answer's kind in `AnswerRead`. For actions they are the assumption that the rules allow as an action only a tool declared to act, and the check in `AnswerAct`. For each half, TLC finds no error with the check removed alone. It also finds no error with the assumption removed alone, in a model whose rules at `v1` allow a tool of the wrong kind. With both removed, in that same model, `KindsKept` is violated. Each run with only the assumption removed reaches the same states as the unchanged model, since the check keeps the tool from ever being asked in the wrong kind.

Seven more mutants each remove one guard whose removal alone violates none of the seven properties: each leaves TLC with no error. No run removes two of these guards together. Six of those guards are the test that the handling is still open in `CheckRead`, `MakeRead`, `Approve`, `Reject`, `Perform` and `Redecide`. By the series' reading of the module, every earlier turn is already settled when a handling ends. By the same reading, the last turn can leave at most one action unsettled: one proposed past the cap and left undecided when the handling stopped. So only the test in `Gate` does any work: it keeps that action from being decided. No mutant removes the same test from `AnswerRead`, `AnswerAct`, `AnswerDone`, `AnswerInvalid` or `NextTurn`. The seventh is `Redecide`

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

clearing an old approval. What stops its reuse is the decision going back to hold. One further guard makes `Redecide` leave alone what has been acted on. A further property, `DecisionOfActedStands`, protects it, and is kept apart in `Redecide.tla`. That module's specification, `SpecH`, satisfies the property. A mutant without the guard violates it. No run checks the seven properties against that mutant.

The module's limits go beyond those Section 8 names. A revision stands for one fixed set of rules, so a decision or an approval made under a revision stands again whenever that revision returns. A person can always answer a hold. Each turn has one answer, and the cap includes refused actions. Of the verifier's tests it models only the kind of an answer's tool.

The fragment is this part's own and is taken from no other module. `Spec` has no fairness condition, and the seven properties are safety properties. The module `Live.tla` adds fairness and the property that a handling ends. Under weak fairness on every step but a reload, TLC finds the property violated. The digest keeps no trace. The series reads the cause as the rules alternating between `v1` and `v2` without end, so that no act stays possible. Under strong fairness on the steps a reload can disable, TLC finds no error. Both runs use the model above, with two revisions, three turns and a cap of two actions, and no larger instance.

The script `check.sh` makes every run and writes those nineteen configurations, and another script writes each mutant and the other files its runs need. Neither is published. Of the 48 runs, 8 use only files Appendix D prints. Another 18 use a configuration a script derives from a printed one: eleven from `MCLoopConfinement.cfg` and seven from `acted.cfg`. `acted.cfg` is the configuration of `Redecide.tla`'s further property. The 22 mutant runs can be repeated only by writing each mutant from its description here. The digest names twenty of them by a short label, such as `perform-stale` for the `Perform` without `decidedAt[j] = rev`. It names the other two by a description. Appendix D also prints `tlc-run.txt`, a digest of TLC's output for all 48 runs.

Appendix C: The Specification in Full

This appendix states interfaces and obligations for an agent runtime built to the specification. It is not code.

Inputs and outputs.

```
# assemble uses the revision recorded at receipt, and shape uses it first; verify,
  check_read
# and gate use the revision in force when they decide. After a reload, shape is run
  again
# under the revision in force, for verify only; the context's stable entries stay at the
# receipt revision (3)
assemble(definitions@revision, message, chain_record, turns_so_far, config)
    -> (context, manifest)                  # once for every turn

shape(agent_rules@revision, handling)
    -> answer_schema            # read | act | done; a read enumeration and an act
      enumeration;
                                # typed arguments; a bounded note. Fixed for the
                                  handling, and shaped
                                # again only if a reload changes the revision in force

verify(answer, answer_schema, scriptable_tests, chain_state)
```

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

```
    -> Valid | Invalid(test_id, field)
                                # chain_state: recorded state that only the runtime
                                  writes

check_read(read, rules@revision, channel, principal)
    -> Allow | Refuse           # every read, before it reaches the gateway; never Hold

gate(action, rules@revision, channel, principal, approval)
    -> Allow | Hold | Refuse    # every action of the handling
```

The manifest. One for every turn, with one entry for each piece of the context, in position order:

```
manifest_entry := {
  turn:       integer,
  entry:      rules | agent | tools | domain | carried | task | working |
    point_of_action,
  source:     path or channel or tool call id or turn,
  version:    revision hash, or content hash for anything not under version control,
  label:      instruction | data,          # domain pieces are always data
  origin:     for data: its source; "own" for the model's notes,
  tokens:     integer,
  offloaded:  optional { path, fetched_at, bytes, freshness_limit }
}

excluded_entry := {
  source, version, tokens,
  reason:     masked | offloaded | pointer | cleared | over_budget
}
```

Configuration for each 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 context length,
  point_of_action:     on | off,
  answer_shape:        schema with the two enumerations,
  model:               provider, model id, version, temperature, effort,
  budgets:             { agent_per_period, principal_per_period },
  loop:                { max_turns, max_actions, max_cost, context_ceiling,
    note_bytes },
  limits:              { escalations_per_principal_per_period }
}
```

Obligations. Each is written to be a test in a conformance suite, unless it says otherwise.

1. The assembler never calls a model to decide what enters the context. This is verified by dependency review, not by a conformance test.
2. Every piece of the context that is neither declared material nor the task carries the label `data` and its origin, and so does every piece of domain material, with its content hash. Documents enter a context only as declared domain material or as the recorded result of a read that `check_read` allowed.
3. The stable entries come first and do not change within a handling.
4. For every turn, the manifest and the assembled context are written to the record before the model is asked.
5. Assembling twice from the same revision and the same recorded inputs yields byte-identical stable entries.
6. Every piece considered and not included is recorded as an excluded entry with its reason.

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

7. An answer is invalid unless it is one read, one action or “done”, with a tool from the enumeration of its kind and arguments matching their types. An invalid answer never reaches `check_read` or the gate.
8. A refusal, and a failed test, cite the rule or the test by its identifier and carry no new rule text into the context.
9. Every answer is recorded as given. An invalid answer, a refused read, a refused action and a failed action are each recorded and returned to the model on the next turn as a report. No answer is sampled again: a turn whose answer is recorded is never asked again.
10. The runtime asks for a turn only when the turn before it is settled and recorded, and asks for none after “done”. It sends the model no message to continue, and adds no new instruction text between turns.
11. No read reaches the gateway without an allow from `check_read`, recorded before the read is made. A read is allowed or refused, never put on hold.
12. Part 1’s separation of the declared tool set from the subset an agent may propose is kept. The declared tool set is fixed for an agent’s revision, and each tool’s kind, read or act, is declared with it in the reviewed definitions. The subset allowed in the state at the handling’s start is expressed in the shaper’s enumerations; within the handling the verifier alone checks it against `chain_state`. It is never expressed by changing the declaration. The gate checks the action’s name against the rules as always.
13. Part 4’s assumption 7 is met, and extended to reads. The revision an allow was decided under is recorded with the decision. An allow, for a read or an action, is acted on only if that revision is in force at the step that commits the read or the action. Otherwise the read or the action is decided again first, under the revision in force, and an approval given under another revision is not reused. The comparison and the commit are one step with respect to a reload. A revision names exactly one set of rules. That the comparison and the commit are one step is verified by review; a test can only sample it.
14. Decisions that read recorded state of a chain of work are serialised for that chain: from a handling’s first decision to its recorded outcome, a hold included, no other handling of that chain decides.
15. The estimate step reserves each handling’s estimate atomically against the period’s budgets at admission and settles it at close. A handling whose reservation would take the period’s reservations and spend over the agent’s budget, or over the budget of the principal the work acts for, is refused before the model is asked. Before every turn, the first included, the handling’s recorded spend and the turn’s estimate are checked against the handling’s cost cap, and before every later turn against the same budgets too. A turn that fails either check is not asked, and the handling stops under obligation 16.
16. The turns, the actions, refused actions included, the cost and the context size of one handling are capped by the agent’s configuration. At a cap the handling stops: the turn is not asked, or the action is not decided. A stopped handling raises

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

one problem for a person and closes, and nothing more of it is asked, decided or performed.

17. Every enumeration of actions contains `escalate`, which is named in the agent's rules so the gate allows it. `escalate` actions are capped for each principal and period: an `escalate` over the cap is invalid, and the runtime reserves one against the count atomically when the verifier passes it. The count is recorded state that only the runtime writes and the verifier reads. Repeated invalid answers and refusals for one agent fold into one problem for a person for each period.
18. The model's note is recorded with its turn. It enters later turns of the same handling labelled `data`, with origin `own`, within its size bound. No mechanism that decides what is allowed, the verifier, `check_read` and the gate included, reads the content of a note or a report; the verifier checks only a note's size. A note never enters a later handling. State that a mechanism reads to decide what is allowed is written only by the runtime, from recorded outcomes. That no mechanism reads such content is verified by review, not by a conformance test.
19. The record is partitioned. Chain entries (messages, proposed actions, decisions, outcomes, and the files large read results were moved to, with their stamped previews) may enter a later handling's context, as `data`. Handling artefacts (assembled contexts, manifests, transcripts, notes) are readable only by the agent's operator and by a reviewer role, and are never assembled into a later handling's context. Chain entries are kept for each principal: an entry recorded for one principal never enters a context assembled for another.
20. Within a handling, when the working state exceeds its budget, the first four steps of Part 2, Section 6, are applied in order, with its placeholder; its fifth, compaction, is never applied within a handling, since a summary is new text a model wrote (Part 1, Appendix A.6). A handling still over its context cap stops under obligation 16.
21. In addition: none of those steps separates a tool call from its result; if any tool call has succeeded, the visible window contains at least one such call with its result, and otherwise nothing is masked; and none of those steps paraphrases an identifier, path or amount that the model can copy into a declared form.
22. A file a result was moved to carries its source, fetch time and size in its preview, and is fetched again, as a read under obligation 11, before it is next used past its freshness limit.
23. Each action has its own identifier, made from its handling and its turn. Its intent is claimed in the record before it is performed, and it is performed at most once on the runtime's own initiative.
24. The runtime reaches outside systems only through one gateway, which alone keeps their credentials. No other component, and never the model, keeps a credential to an outside system. That no other component keeps one, and that no other path exists, is verified by review.
25. No action reaches the gateway without an allow from the gate, recorded before the action is performed.
26. The performer performs exactly the action the recorded allow names, with its recorded arguments, and never derives it again from the answer.

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

27. The record is append-only: no entry is changed or removed. Every write is durable before it is acknowledged; a test can only sample durability.
28. Rules are loaded only from a revision whose review is recorded. Every reload is recorded, with the revision it puts in force, before any decision is made under that revision.

Appendix D: The Model Files

Each file below is printed, at build time, from the file TLC checked. In the PDF a line too long for the page wraps; the Markdown version of this part keeps every line whole, for copying.

`LoopConfinement.tla`, the module

```
--------------------------- MODULE LoopConfinement ---------------------------
(* A fragment written for Context Engineering, Part 6. One handling of one    *)
(* message: turns in which the proposer asks for a read, proposes an action,  *)
(* says it is done, or gives an invalid answer. A reload may change the rules *)
(* at any step, a decision and its act included.                               *)
EXTENDS Naturals, FiniteSets

CONSTANTS Tools,            \* every tool a proposer can name
          ReadTools,        \* the tools declared read; the rest are declared act
          Revisions,        \* rule revisions a reload may put in force
          MaxTurns,         \* the cap on turns for a handling
          MaxActions,       \* the cap on actions decided for a handling
          Allowed(_),       \* Allowed(r): the act tools the rules at r allow
          Held(_),          \* Held(r): the act tools the rules at r mark hold
          ReadAllowed(_)    \* ReadAllowed(r): the read tools the rules at r allow

ASSUME /\ ReadTools \subseteq Tools
       /\ MaxTurns \in Nat \ {0}
       /\ MaxActions \in Nat
       /\ \A r \in Revisions : /\ Held(r) \subseteq Allowed(r)
                               /\ Allowed(r) \subseteq (Tools \ ReadTools)
                               /\ ReadAllowed(r) \subseteq ReadTools

Turn == 1..MaxTurns

VARIABLES rev,        \* the revision in force
          turn,       \* the turn being worked
          kind,       \* per turn: "none", "read", "act", "done" or "invalid"
          tool,       \* per turn: the tool its answer names
          decision,   \* per turn: "none", "hold", "allow" or "refuse"
          decidedAt,  \* per turn: the revision its current decision was made under
          approved,   \* per turn: whether a hold was approved
          performed,  \* the turns whose action has been performed
          reads,      \* the turns whose read has been made
          ended       \* "open", "done" or "stopped"

vars == <<rev, turn, kind, tool, decision, decidedAt, approved, performed, reads,
  ended>>

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

Settled(j) == \/ kind[j] = "invalid"
              \/ kind[j] = "read" /\ (decision[j] = "refuse" \/ j \in reads)
              \/ kind[j] = "act"  /\ (decision[j] = "refuse" \/ j \in performed)

ActsBefore(j) == Cardinality({i \in Turn : i < j /\ kind[i] = "act"})

Init == /\ rev \in Revisions
        /\ turn = 1
        /\ kind      = [j \in Turn |-> "none"]
        /\ tool      = [j \in Turn |-> CHOOSE t \in Tools : TRUE]
```

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

```
        /\ decision  = [j \in Turn |-> "none"]
        /\ decidedAt = [j \in Turn |-> rev]
        /\ approved  = [j \in Turn |-> FALSE]
        /\ performed = {}
        /\ reads     = {}
        /\ ended     = "open"

Reload(r) == /\ rev' = r
             /\ UNCHANGED <<turn, kind, tool, decision, decidedAt, approved, performed,
               reads, ended>>

\* The proposer's answer for the turn being worked. The verifier's check of the
\* tool against the enumeration of the answer's kind is folded in: a tool of
\* the wrong kind makes the answer invalid.
AnswerRead(t) == /\ ended = "open" /\ kind[turn] = "none"
                 /\ kind' = [kind EXCEPT ![turn] = IF t \in ReadTools THEN "read" ELSE
                   "invalid"]
                 /\ tool' = [tool EXCEPT ![turn] = t]
                 /\ UNCHANGED <<rev, turn, decision, decidedAt, approved, performed,
                   reads, ended>>

\* An action proposed once MaxActions actions have been proposed is not
\* decided: the handling stops.
AnswerAct(t) == /\ ended = "open" /\ kind[turn] = "none"
                /\ tool' = [tool EXCEPT ![turn] = t]
                /\ IF t \in ReadTools
                   THEN /\ kind' = [kind EXCEPT ![turn] = "invalid"]
                        /\ ended' = ended
                   ELSE /\ kind' = [kind EXCEPT ![turn] = "act"]
                        /\ ended' = IF ActsBefore(turn) >= MaxActions THEN "stopped"
                          ELSE ended
                /\ UNCHANGED <<rev, turn, decision, decidedAt, approved, performed,
                  reads>>

AnswerDone == /\ ended = "open" /\ kind[turn] = "none"
              /\ kind' = [kind EXCEPT ![turn] = "done"]
              /\ ended' = "done"
              /\ UNCHANGED <<rev, turn, tool, decision, decidedAt, approved, performed,
                reads>>

AnswerInvalid == /\ ended = "open" /\ kind[turn] = "none"
                 /\ kind' = [kind EXCEPT ![turn] = "invalid"]
                 /\ UNCHANGED <<rev, turn, tool, decision, decidedAt, approved,
                   performed, reads, ended>>

\* The check on a read: allow or refuse, never hold.
CheckRead(j) == /\ ended = "open" /\ kind[j] = "read" /\ decision[j] = "none"
                /\ decision'  = [decision  EXCEPT ![j] = IF tool[j] \in ReadAllowed(rev)
                  THEN "allow" ELSE "refuse"]
                /\ decidedAt' = [decidedAt EXCEPT ![j] = rev]
                /\ UNCHANGED <<rev, turn, kind, tool, approved, performed, reads,
                  ended>>

MakeRead(j) == /\ ended = "open" /\ kind[j] = "read"
               /\ decision[j] = "allow" /\ decidedAt[j] = rev
               /\ j \notin reads
               /\ reads' = reads \cup {j}
               /\ UNCHANGED <<rev, turn, kind, tool, decision, decidedAt, approved,
                 performed, ended>>

Gate(j) == /\ ended = "open" /\ kind[j] = "act" /\ decision[j] = "none"
           /\ decision'  = [decision  EXCEPT ![j] = Decide(tool[j], rev)]
           /\ decidedAt' = [decidedAt EXCEPT ![j] = rev]
           /\ UNCHANGED <<rev, turn, kind, tool, approved, performed, reads, ended>>

\* An approved hold becomes an allow only under the revision its hold was
\* decided under; a stale hold is decided again first.
Approve(j) == /\ ended = "open" /\ kind[j] = "act" /\ decision[j] = "hold"
              /\ decidedAt[j] = rev
              /\ approved' = [approved EXCEPT ![j] = TRUE]
              /\ decision'  = [decision  EXCEPT ![j] = "allow"]
              /\ decidedAt' = [decidedAt EXCEPT ![j] = rev]
              /\ UNCHANGED <<rev, turn, kind, tool, performed, reads, ended>>
```

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

```
\* A person refuses a hold: the action is refused.
Reject(j) == /\ ended = "open" /\ kind[j] = "act" /\ decision[j] = "hold"
             /\ decision' = [decision EXCEPT ![j] = "refuse"]
             /\ UNCHANGED <<rev, turn, kind, tool, decidedAt, approved, performed,
               reads, ended>>

Perform(j) == /\ ended = "open" /\ kind[j] = "act"
              /\ decision[j] = "allow" /\ decidedAt[j] = rev
              /\ j \notin performed
              /\ performed' = performed \cup {j}
              /\ UNCHANGED <<rev, turn, kind, tool, decision, decidedAt, approved,
                reads, ended>>

\* After a reload, a decision not yet acted on is made again under the
\* revision in force, and an old approval is not reused.
Redecide(j) == /\ ended = "open"
               /\ decision[j] \in {"allow", "hold"} /\ decidedAt[j] # rev
               /\ j \notin performed /\ j \notin reads
               /\ decision' = [decision EXCEPT ![j] =
                                 IF kind[j] = "read"
                                 THEN IF tool[j] \in ReadAllowed(rev) THEN "allow" ELSE
                                   "refuse"
                                 ELSE Decide(tool[j], rev)]
               /\ decidedAt' = [decidedAt EXCEPT ![j] = rev]
               /\ approved'  = [approved  EXCEPT ![j] = FALSE]
               /\ UNCHANGED <<rev, turn, kind, tool, performed, reads, ended>>

\* The runtime moves to the next turn only when this one is settled; at the
\* cap on turns it stops. It sends the proposer nothing to make it go on.
NextTurn == /\ ended = "open" /\ Settled(turn)
            /\ IF turn < MaxTurns
               THEN turn' = turn + 1 /\ ended' = ended
               ELSE turn' = turn /\ ended' = "stopped"
            /\ UNCHANGED <<rev, kind, tool, decision, decidedAt, approved, performed,
              reads>>

Next == \/ \E r \in Revisions : Reload(r)
        \/ \E t \in Tools : AnswerRead(t) \/ AnswerAct(t)
        \/ AnswerDone \/ AnswerInvalid \/ NextTurn
        \/ \E j \in Turn : CheckRead(j) \/ MakeRead(j) \/ Gate(j) \/ Approve(j)
                         \/ Reject(j) \/ Perform(j) \/ Redecide(j)

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

TypeOK == /\ rev \in Revisions /\ turn \in Turn
          /\ kind \in [Turn -> {"none", "read", "act", "done", "invalid"}]
          /\ tool \in [Turn -> Tools]
          /\ decision \in [Turn -> {"none", "hold", "allow", "refuse"}]
          /\ decidedAt \in [Turn -> Revisions]
          /\ approved \in [Turn -> BOOLEAN]
          /\ performed \subseteq Turn /\ reads \subseteq Turn
          /\ ended \in {"open", "done", "stopped"}

\* Every action of the handling, however many, is performed under the rules
\* in force at the step that performs it.
StrongConfinement ==
  [][\A j \in Turn : (j \notin performed /\ j \in performed')
        => (tool[j] \in Allowed(rev) /\ (tool[j] \in Held(rev) => approved[j]))]_vars

\* Every read is made under an allow of the rules in force at that step.
ReadsChecked ==
  [][\A j \in Turn : (j \notin reads /\ j \in reads') => tool[j] \in
    ReadAllowed(rev)]_vars

\* Only a tool declared read is ever read, and only one declared act performed.
KindsKept == /\ \A j \in reads : tool[j] \in ReadTools
             /\ \A j \in performed : tool[j] \notin ReadTools

SettledBeforeNext == \A j \in Turn : j < turn => Settled(j)

NothingAfterDone == \A j \in Turn : kind[j] = "done" =>
                      /\ ended = "done"
```

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

```
                      /\ \A i \in Turn : i > j => kind[i] = "none"

ActionsBounded == Cardinality({j \in Turn : kind[j] = "act" /\ decision[j] # "none"}) <=
  MaxActions

\* Once the handling has ended, nothing of it changes; only the rules may.
StopStands ==
  [][ended # "open" => UNCHANGED <<turn, kind, tool, decision, decidedAt, approved,
    performed, reads, ended>>]_vars
=============================================================================
```

`MCLoopConfinement.tla`, its model

```
------------------------- MODULE MCLoopConfinement -------------------------
EXTENDS LoopConfinement

\* v1 allows t1 and t2 and marks t2 hold, and allows both reads.
\* v2 allows t1 only and marks it hold, and allows one read.
MCAllowed(r)     == IF r = "v1" THEN {"t1", "t2"} ELSE {"t1"}
MCHeld(r)        == IF r = "v1" THEN {"t2"} ELSE {"t1"}
MCReadAllowed(r) == IF r = "v1" THEN {"r1", "r2"} ELSE {"r1"}

\* Reachability: each is an invariant the run is expected to violate.
NeverTwoActions    == Cardinality(performed) < 2
NeverReadThenAct   == ~(\E i, j \in Turn : i < j /\ i \in reads /\ j \in performed)
NeverActAfterRefusal == ~(\E i, j \in Turn : i < j /\ kind[i] = "act" /\ decision[i] =
  "refuse" /\ j \in performed)
NeverApprovedPerformed == ~(\E j \in performed : approved[j])
NeverGoesOnAfterHold == ~(\E i, j \in Turn : i < j /\ i \in performed /\ approved[i] /\
  kind[j] # "none")
NeverDone          == ended # "done"
NeverStopped       == ended # "stopped"
NeverActionCap     == ~(ended = "stopped" /\ \E j \in Turn : kind[j] = "act" /\
  decision[j] = "none")
NeverTurnCap       == ~(ended = "stopped" /\ Settled(turn))
NeverRefusedRead   == ~(\E j \in Turn : kind[j] = "read" /\ decision[j] = "refuse")
NeverRejectedHold  == ~(\E j \in Turn : kind[j] = "act" /\ decision[j] = "refuse" /\
  tool[j] \in MCAllowed(decidedAt[j]))
=============================================================================
```

`MCLoopConfinement.cfg`, its configuration

```
SPECIFICATION Spec
CONSTANTS
  Tools = {"t1", "t2", "r1", "r2"}
  ReadTools = {"r1", "r2"}
  Revisions = {"v1", "v2"}
  MaxTurns = 3
  MaxActions = 2
  Allowed <- MCAllowed
  Held <- MCHeld
  ReadAllowed <- MCReadAllowed
INVARIANTS TypeOK KindsKept SettledBeforeNext NothingAfterDone ActionsBounded
PROPERTIES StrongConfinement ReadsChecked StopStands
```

`MCBig.tla`, the model of the instance `big`

```
------------------------------- MODULE MCBig -------------------------------
EXTENDS LoopConfinement

\* v1 allows t1 and t2 and marks t2 hold, and allows both reads.
\* v2 allows t1 only and marks it hold, and allows one read.
MCAllowed(r)     == IF r = "v1" THEN {"t1", "t2"} ELSE IF r = "v2" THEN {"t1"} ELSE {}
MCHeld(r)        == IF r = "v1" THEN {"t2"} ELSE IF r = "v2" THEN {"t1"} ELSE {}
MCReadAllowed(r) == IF r = "v1" THEN {"r1", "r2"} ELSE IF r = "v2" THEN {"r1"} ELSE {}

\* Reachability: each is an invariant the run is expected to violate.
NeverTwoActions    == Cardinality(performed) < 2
NeverReadThenAct   == ~(\E i, j \in Turn : i < j /\ i \in reads /\ j \in performed)
NeverActAfterRefusal == ~(\E i, j \in Turn : i < j /\ kind[i] = "act" /\ decision[i] =
  "refuse" /\ j \in performed)
NeverApprovedPerformed == ~(\E j \in performed : approved[j])
```

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

```
NeverGoesOnAfterHold == ~(\E i, j \in Turn : i < j /\ i \in performed /\ approved[i] /\
  kind[j] # "none")
NeverDone          == ended # "done"
NeverStopped       == ended # "stopped"
NeverActionCap     == ~(ended = "stopped" /\ \E j \in Turn : kind[j] = "act" /\
  decision[j] = "none")
NeverTurnCap       == ~(ended = "stopped" /\ Settled(turn))
NeverRefusedRead   == ~(\E j \in Turn : kind[j] = "read" /\ decision[j] = "refuse")
NeverRejectedHold  == ~(\E j \in Turn : kind[j] = "act" /\ decision[j] = "refuse" /\
  tool[j] \in MCAllowed(decidedAt[j]))
=============================================================================
```

`MCBig.cfg`, its configuration

```
SPECIFICATION Spec
CONSTANTS
  Tools = {"t1", "t2", "r1", "r2"}
  ReadTools = {"r1", "r2"}
  Revisions = {"v1", "v2", "v3"}
  MaxTurns = 4
  MaxActions = 2
  Allowed <- MCAllowed
  Held <- MCHeld
  ReadAllowed <- MCReadAllowed
INVARIANTS TypeOK KindsKept SettledBeforeNext NothingAfterDone ActionsBounded
PROPERTIES StrongConfinement ReadsChecked StopStands
```

`MCRules.tla`, the model of the instance `rules`

```
------------------------------ MODULE MCRules -------------------------------
EXTENDS LoopConfinement
\* Every rule table over two action tools is a revision: nine of them.
\* A table allows read r2 only if it allows t1 without a hold, so that the read rules
  vary too.
ActTools == {"t1", "t2"}
MCRevisions == {x \in [a : SUBSET ActTools, h : SUBSET ActTools] : x.h \subseteq x.a}
MCAllowed(r)     == r.a
MCHeld(r)        == r.h
MCReadAllowed(r) == IF "t1" \in r.a \ r.h THEN {"r1", "r2"} ELSE {"r1"}
=============================================================================
```

`MCRules.cfg`, its configuration

```
SPECIFICATION Spec
CONSTANTS
  Tools = {"t1", "t2", "r1", "r2"}
  ReadTools = {"r1", "r2"}
  Revisions <- MCRevisions
  MaxTurns = 3
  MaxActions = 2
  Allowed <- MCAllowed
  Held <- MCHeld
  ReadAllowed <- MCReadAllowed
INVARIANTS TypeOK KindsKept SettledBeforeNext NothingAfterDone ActionsBounded
PROPERTIES StrongConfinement ReadsChecked StopStands
```

`MCThree.tla`, the model of the instance `three`

```
------------------------------ MODULE MCThree -------------------------------
EXTENDS LoopConfinement
\* v1 allows t1, t2 and t3 and marks t2 hold. v2 allows t1 and t3 and marks t1 hold.
MCAllowed(r)     == IF r = "v1" THEN {"t1", "t2", "t3"} ELSE {"t1", "t3"}
MCHeld(r)        == IF r = "v1" THEN {"t2"} ELSE {"t1"}
MCReadAllowed(r) == IF r = "v1" THEN {"r1", "r2"} ELSE {"r1"}
NeverThreeActions == Cardinality(performed) < 3
=============================================================================
```

`MCThree.cfg`, its configuration

```
SPECIFICATION Spec
CONSTANTS
  Tools = {"t1", "t2", "t3", "r1", "r2"}
  ReadTools = {"r1", "r2"}
  Revisions = {"v1", "v2"}
```

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

```
  MaxTurns = 4
  MaxActions = 3
  Allowed <- MCAllowed
  Held <- MCHeld
  ReadAllowed <- MCReadAllowed
INVARIANTS TypeOK KindsKept SettledBeforeNext NothingAfterDone ActionsBounded
PROPERTIES StrongConfinement ReadsChecked StopStands
```

`three.cfg`, the configuration of its further run

```
SPECIFICATION Spec
CONSTANTS
  Tools = {"t1", "t2", "t3", "r1", "r2"}
  ReadTools = {"r1", "r2"}
  Revisions = {"v1", "v2"}
  MaxTurns = 4
  MaxActions = 3
  Allowed <- MCAllowed
  Held <- MCHeld
  ReadAllowed <- MCReadAllowed
INVARIANTS NeverThreeActions
```

`Redecide.tla`, the paths through a decision made again, and a further property

```
------------------------------ MODULE Redecide ------------------------------
(* Paths through Redecide, and one further property, for Context Engineering, *)
(* Part 6. History variables record what LoopConfinement does not keep.       *)
EXTENDS MCLoopConfinement
VARIABLES h,  \* the turns whose decision has been made again after a reload
          w   \* the turns whose approval a decision made again has withdrawn

InitH == Init /\ h = {} /\ w = {}
NextH == /\ Next
         /\ h' = h \cup {j \in Turn : decision[j] \in {"allow", "hold"} /\ decidedAt'[j]
           # decidedAt[j]}
         /\ w' = w \cup {j \in Turn : approved[j] /\ ~approved'[j]}
SpecH == InitH /\ [][NextH]_<<vars, h, w>>

\* Reachability: each is expected to be violated.
NeverPerformAfterRedecide  == performed \cap h = {}
NeverReadAfterRedecide     == reads \cap h = {}
NeverRedecideAllowToRefuse == [][\A j \in Turn : ~(kind[j] = "act" /\ decision[j] =
  "allow" /\ decision'[j] = "refuse")]_vars
NeverRedecideAllowToHold   == [][\A j \in Turn : ~(decision[j] = "allow" /\ decision'[j]
  = "hold")]_vars
NeverReadRedecidedToRefuse == [][\A j \in Turn : ~(kind[j] = "read" /\ decision[j] =
  "allow" /\ decision'[j] = "refuse")]_vars
NeverApprovalWithdrawn     == [][\A j \in Turn : ~(approved[j] /\ ~approved'[j])]_vars
NeverPerformedOnSecondApproval == ~(\E j \in performed \cap w : approved[j])

\* Expected to hold: the decision of a read made or an action performed stands.
DecisionOfActedStands ==
  [][\A j \in performed \cup reads : decision'[j] = decision[j] /\ decidedAt'[j] =
    decidedAt[j] /\ approved'[j] = approved[j]]_vars
=============================================================================
```

`acted.cfg`, the configuration of its further property

```
SPECIFICATION SpecH
CONSTANTS
  Tools = {"t1", "t2", "r1", "r2"}
  ReadTools = {"r1", "r2"}
  Revisions = {"v1", "v2"}
  MaxTurns = 3
  MaxActions = 2
  Allowed <- MCAllowed
  Held <- MCHeld
  ReadAllowed <- MCReadAllowed
PROPERTIES DecisionOfActedStands
```

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

`Live.tla`, the module with fairness

```
------------------------------- MODULE Live --------------------------------
EXTENDS MCLoopConfinement
Answer == (\E t \in Tools : AnswerRead(t) \/ AnswerAct(t)) \/ AnswerDone \/
  AnswerInvalid
Runtime(j) == CheckRead(j) \/ Gate(j)
Person(j)  == Approve(j) \/ Reject(j)
\* Weak fairness everywhere.
WeakFair == /\ WF_vars(Answer) /\ WF_vars(NextTurn)
            /\ \A j \in Turn : /\ WF_vars(Runtime(j)) /\ WF_vars(Person(j))
                               /\ WF_vars(MakeRead(j)) /\ WF_vars(Perform(j)) /\
                                 WF_vars(Redecide(j))
\* Strong fairness on the steps a reload can disable.
StrongFair == /\ WF_vars(Answer) /\ WF_vars(NextTurn)
              /\ \A j \in Turn : /\ WF_vars(Runtime(j)) /\ SF_vars(Person(j))
                                 /\ SF_vars(MakeRead(j)) /\ SF_vars(Perform(j)) /\
                                   SF_vars(Redecide(j))
WeakSpec   == Spec /\ WeakFair
StrongSpec == Spec /\ StrongFair
Ends == <>(ended # "open")
=============================================================================
```

`weak.cfg`, its configuration under weak fairness

```
SPECIFICATION WeakSpec
CONSTANTS
  Tools = {"t1", "t2", "r1", "r2"}
  ReadTools = {"r1", "r2"}
  Revisions = {"v1", "v2"}
  MaxTurns = 3
  MaxActions = 2
  Allowed <- MCAllowed
  Held <- MCHeld
  ReadAllowed <- MCReadAllowed
PROPERTIES Ends
```

`strong.cfg`, its configuration under strong fairness

```
SPECIFICATION StrongSpec
CONSTANTS
  Tools = {"t1", "t2", "r1", "r2"}
  ReadTools = {"r1", "r2"}
  Revisions = {"v1", "v2"}
  MaxTurns = 3
  MaxActions = 2
  Allowed <- MCAllowed
  Held <- MCHeld
  ReadAllowed <- MCReadAllowed
PROPERTIES Ends
```

`tlc-run.txt`, the digest of TLC's output for every run

```
LoopConfinement model check, every run of check.sh. tla2tools.jar sha256
  936a262061c914694dfd669a543be24573c45d5aa0ff20a8b96b23d01e050e88. TLC2 Version 2.19 of
  08 August 2024 (rev: 5a47802).
Constants: Tools {t1,t2,r1,r2}; ReadTools {r1,r2}; Revisions {v1,v2}; MaxTurns 3;
  MaxActions 2.
v1 allows t1 and t2, marks t2 hold, allows reads r1 and r2. v2 allows t1, marks t1 hold,
  allows read r1.
One line for each run of check.sh: the run, TLC's verdict line and, for a run with no
  error, states generated / distinct states found. TLC printed the same version line for
  every run.
Main: invariants TypeOK KindsKept SettledBeforeNext NothingAfterDone ActionsBounded;
  properties StrongConfinement ReadsChecked StopStands | Model checking completed. No
  error has been found. | 82418 / 27784
Reachability, expected violated: NeverTwoActions | Error: Invariant NeverTwoActions is
  violated.
Reachability, expected violated: NeverReadThenAct | Error: Invariant NeverReadThenAct is
  violated.
Reachability, expected violated: NeverActAfterRefusal | Error: Invariant
  NeverActAfterRefusal is violated.
Reachability, expected violated: NeverApprovedPerformed | Error: Invariant
  NeverApprovedPerformed is violated.
```

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

```
Reachability, expected violated: NeverGoesOnAfterHold | Error: Invariant
  NeverGoesOnAfterHold is violated.
Reachability, expected violated: NeverDone | Error: Invariant NeverDone is violated.
Reachability, expected violated: NeverStopped | Error: Invariant NeverStopped is
  violated.
Reachability, expected violated: NeverActionCap | Error: Invariant NeverActionCap is
  violated.
Reachability, expected violated: NeverTurnCap | Error: Invariant NeverTurnCap is
  violated.
Reachability, expected violated: NeverRefusedRead | Error: Invariant NeverRefusedRead is
  violated.
Reachability, expected violated: NeverRejectedHold | Error: Invariant NeverRejectedHold
  is violated.
Mutant, expected violated: perform-stale | Error: Action property StrongConfinement is
  violated.
Mutant, expected violated: read-stale | Error: Action property ReadsChecked is violated.
Mutant, expected violated: approve-stale | Error: Action property StrongConfinement is
  violated.
Mutant, expected violated: next-unsettled | Error: Invariant SettledBeforeNext is
  violated.
Mutant, expected violated: no-action-cap | Error: Invariant ActionsBounded is violated.
Mutant, expected violated: done-continues | Error: Invariant NothingAfterDone is
  violated.
Mutant, expected violated: gate-after-stop | Error: Invariant ActionsBounded is
  violated.
Mutant gate-after-stop without ActionsBounded, expected violated: StopStands | Error:
  Action property StopStands is violated.
Mutant with one of KindsKept's two guards removed, expected no error: kind-check-only |
  Model checking completed. No error has been found. | 131854 / 44840
Mutant with one of KindsKept's two guards removed, expected no error: assume-only |
  Model checking completed. No error has been found. | 82418 / 27784
Mutant with both of KindsKept's guards removed, expected violated: KindsKept | Error:
  Invariant KindsKept is violated.
Larger instance: Revisions {v1,v2,v3}, v3 allows nothing; MaxTurns 4; MaxActions 2; same
  invariants and properties | Model checking completed. No error has been found. |
  6975705 / 1785456
Mutant with one of the two guards of KindsKept's action half removed, expected no error:
  act-kind-check-only | Model checking completed. No error has been found. | 95890 /
  32216
Mutant with one of the two guards of KindsKept's action half removed, expected no error:
  act-assume-only | Model checking completed. No error has been found. | 82418 / 27784
Mutant with both guards of KindsKept's action half removed, expected violated: KindsKept
  | Error: Invariant KindsKept is violated.
Reachability through Redecide, expected violated: NeverPerformAfterRedecide | Error:
  Invariant NeverPerformAfterRedecide is violated.
Reachability through Redecide, expected violated: NeverReadAfterRedecide | Error:
  Invariant NeverReadAfterRedecide is violated.
Reachability through Redecide, expected violated: NeverPerformedOnSecondApproval |
  Error: Invariant NeverPerformedOnSecondApproval is violated.
Reachability through Redecide, expected violated: NeverRedecideAllowToRefuse | Error:
  Action property NeverRedecideAllowToRefuse is violated.
Reachability through Redecide, expected violated: NeverRedecideAllowToHold | Error:
  Action property NeverRedecideAllowToHold is violated.
Reachability through Redecide, expected violated: NeverReadRedecidedToRefuse | Error:
  Action property NeverReadRedecidedToRefuse is violated.
Reachability through Redecide, expected violated: NeverApprovalWithdrawn | Error: Action
  property NeverApprovalWithdrawn is violated.
Further property DecisionOfActedStands, expected no error | Model checking completed. No
  error has been found. | 341428 / 118444
Mutant redecide-acted (Redecide may decide again what was acted on), expected violated:
  DecisionOfActedStands | Error: Action property DecisionOfActedStands is violated.
Instance rules: every rule table over two action tools as a revision (nine); MaxTurns 3;
  MaxActions 2; same invariants and properties | Model checking completed. No error has
  been found. | 73438983 / 7429806
Instance three: three action tools; MaxTurns 4; MaxActions 3; same invariants and
  properties | Model checking completed. No error has been found. | 2659882 / 902948
Instance three, reachability, expected violated: NeverThreeActions | Error: Invariant
  NeverThreeActions is violated.
Liveness under weak fairness on every step but Reload, expected violated: Ends | Error:
  Temporal properties were violated.
Liveness under strong fairness on the steps a reload can disable, expected no error:
  Ends | Model checking completed. No error has been found. | 82418 / 27784
```

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

```
Mutant with a guard no property protects removed, expected no error: noend-CheckRead |
  Model checking completed. No error has been found. | 82418 / 27784
Mutant with a guard no property protects removed, expected no error: noend-MakeRead |
  Model checking completed. No error has been found. | 82418 / 27784
Mutant with a guard no property protects removed, expected no error: noend-Approve |
  Model checking completed. No error has been found. | 82418 / 27784
Mutant with a guard no property protects removed, expected no error: noend-Reject |
  Model checking completed. No error has been found. | 82418 / 27784
Mutant with a guard no property protects removed, expected no error: noend-Perform |
  Model checking completed. No error has been found. | 82418 / 27784
Mutant with a guard no property protects removed, expected no error: noend-Redecide |
  Model checking completed. No error has been found. | 82418 / 27784
Mutant with a guard no property protects removed, expected no error: redecide-keeps-
  approval | Model checking completed. No error has been found. | 129436 / 43958
```

Revision History

Version	Date	Change
1.0.0	2026-11-19	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.

Product behaviours cited in this part are as read on 15 November 2026, and each may change with a release.

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