VELOCITY EDGE

CONTEXT ENGINEERING, PART 5

The Levers and What They Cost

ABSTRACT

This is the fifth of six papers on why an AI agent stops following its rules over a long session, and on what can be relied on instead. Part 1 said what in an agent's context can be set. Part 2 stated a hypothesis: within a model's stated context length, a rule kept by reading, and whose breaches can be observed, decays with the session. Part 3 said which properties code outside the model can hold. Part 4 said what can then be promised. This part asks what each design choice buys and what it costs. It runs no experiment: its models are parameterised, not fitted.

We argue that constraining the act to one named action with typed arguments is the strongest lever. Its cost to a model's reasoning is disputed. We argue that a design can keep that cost, if there is one, away from the model's reasoning by constraining only the act. Sampling a decision several times and acting on the majority can signal uncertainty, and never establishes correctness. If there are inputs on which the model usually gives the same wrong answer, more samples do not remove a simple majority's error on them. A loop that returns a failed proposal to the model can be turned against the defender, who, by assumption, pays for a long context each time an adversary sends a short message. A limit on attempts bounds the defender's cost for each message. A budget for each sender over a period bounds the defender's spend charged to that sender, at each agent, at about the budget. By our reasoning, text planted in a tool's result or a retrieved document can spend the budget of a sender who did not send it. The part also models when a cached prefix pays, weighs the repetition of a rule at the point of action, and closes with five tests a buyer can put to any agent runtime.

Phillip Pienaar
Velocity Edge

2026-11-19
Version 1.0.1

Contents

Executive Summary 3
1. The Question 3
2. Constraining the Act 4
3. Sampled Agreement 4
4. The Verification Loop 5
5. Cache Economics 6
6. Repeating a Rule at the Point of Action 7
7. What a Run Reports, and What the Levers Rest On 8
8. Five Tests for Any Agent Platform 8
Conclusion 8
Glossary 9
References 9
Revision History 10
Licence and Disclaimer 10

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

Executive Summary

This part prices the design choices that follow from Parts 1 to 4. It runs no experiment. Its models are parameterised, not fitted.

We argue that constraining the act is the strongest lever. Requiring the proposal to be one named action with typed arguments is what lets a validator hold "only a named tool". What that constraint costs a model's reasoning is disputed. We argue that a design can keep that cost, if there is one, away from the model's reasoning by constraining only the act.

Agreement among sampled decisions can signal uncertainty. It never establishes correctness. If there are inputs on which the model usually gives the same wrong answer, more samples do not remove a simple majority's error on them.

A loop that sends a failed proposal back to the model can be turned against the defender. Assume an adversary can build a short message that, with high probability, makes the model's proposal fail verification on every attempt. The defender then pays for a long context once for each attempt.

A limit on attempts bounds the defender's cost for each message. A budget for each sender over a period bounds the defender's spend charged to that sender, at each agent, at about the budget. By our reasoning, text planted in a tool's result or a retrieved document can spend the budget of a sender who did not send it.

Where a cache write costs more than ordinary input, caching a stable prefix pays once the prefix is likely enough to still be cached when the next request arrives. By our reasoning, what expires it is the gap between requests, not the length of the session.

Repeating a rule at the point of action costs fresh tokens on every act. By Part 3, it cannot make any property held. The part closes with five tests for runtimes.

1. The Question

Part 1, Context Is Configured, named ten settings of two kinds. Part 2, Why Do Long-Running Agents Drift?, stated the decay hypothesis, within the limits its Section 4 sets. Part 3, Held and Counted, gave the criterion that separates two kinds of property. An outside mechanism computes the first from the proposal's declared form and inputs no proposal chooses, and enforces it. The violation rate of the second can only be estimated, unless a mechanism Part 3 leaves aside keeps that property, or a stricter one is held in its place. Part 4, What Can Be Promised, stated three promises and their assumptions.

This part uses their terms without defining them again: Part 1's settings, layers and act-level rules, Part 2's compaction, Part 3's validator, gate, held and counted, and Part 4's allowed set and actions on hold. A lever is a choice whose effect and cost can be modelled: a setting, or a mechanism built on settings. The question is what each lever buys, and what it costs.

The models here are parameterised, not fitted. Where a parameter matters, a measurement would set it. This series measures none. Product behaviours cited in this part

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

are as read on 15 November 2026, and each may change with a release. Sections 4 and 5 assume that a provider caches the prefix of a request. The series' model is of transformer-based agents. Part 1 gives both assumptions (its Sections 2 and 5).

2. Constraining the Act

We argue that constraining the act is the strongest lever. The form it imposes is one action from a list, with typed arguments. That form is what lets a validator before the gate hold "only a named tool". Part 3 gives the argument, and says why the validator is the mechanism and the decoder an aid. A property of the act's target or text stays counted unless the gate or a validator reads and checks it (Part 3).

The cost of constraint is disputed for reasoning tasks. Tam and others "observe a significant decline in LLMs' reasoning abilities under format restrictions" [1].

The finding is an overall one, across three reasoning tasks and five models in the main experiments. It was not uniform: for some models and tasks a structured format matched or beat free text. The paper gives no statistical test. It left out more powerful models because of cost. Tam and others also had models answer in free text and then convert the answer to the format. That scored nearly the same as free text, except slightly lower on one small model [1].

A response published by the firm .txt argues that the comparison was not like for like, because the prompts differed between conditions [2]. In its own re-implementation on one model it reports scores as good or better with structured generation. The response is a blog post and is not peer-reviewed. Its author describes the firm as "passionate about structured generation".

JSONSchemaBench, which Part 3 cites on how engines differ, also ran the three reasoning tasks, on one model with four engines [3]. On each task, constrained output scored as high as unconstrained output or higher. It gives no statistical test, and its prompts largely follow the .txt response's, so the two are not independent.

We do not settle the dispute. Drawing on Tam and others' two-step result, we argue that a design can keep that cost, if there is one, away from the model's reasoning by constraining only the act. The model reasons without format restriction. The final proposal is constrained (Part 1, Appendix A.8). The validator then checks again every constraint the schema expresses.

3. Sampled Agreement

Agreement never establishes correctness. Suppose a decision is sampled k times (Part 1, Appendix A.8) and acted on only if a majority agrees. If the samples were independent, each wrong with probability q below one half, the majority's error would vanish as k grows.

Correlated errors change this. Assume instead that each input has its own latent error rate Θ, and that the k samples err independently given Θ. As k grows, the majority's error tends to

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

$Pr(\Theta > ½) + ½ \cdot Pr(\Theta = ½)$

The second term is the error on inputs whose latent rate is exactly one half. It vanishes when Θ has no mass there. The limit is positive whenever the latent rate sometimes exceeds one half: that is, whenever there are inputs on which the model is usually wrong.

This calculation assumes each sample is right or wrong. Where wrong answers can differ, the condition for a positive limit is that most samples agree on one wrong answer.

Chen and others sampled one model, GPT-3.5, 400 times on each query. They call a query hard when the most common answer over unlimited calls is wrong. On one set of physics questions, 47 per cent of queries were hard [4]. The evidence across models is one study of pairs of models [5].

Neither study shows most samples agreeing on a wrong answer. We assume that happens. Under that assumption, a simple majority of samples keeps an error that no number of samples removes. We call it the error floor. Take one model at a pinned version, sampled repeatedly on a fixed suite of the kind Part 4 describes. We are wrong if it has no inputs on which most samples agree on a wrong answer.

A design can use sampled agreement to signal uncertainty, by escalating to a person when agreement falls below a stated share. That person carries the limit Part 4 argues. The calculation above does not cover that design. We argue that a design that samples should bound escalations, for instance by refusing the act once a stated number of disagreements for the piece of work is reached.

4. The Verification Loop

In a verification loop a proposal the validator refuses goes back to the model for another attempt, up to u attempts. Let each attempt cost c: the context's tokens at the input price, plus output. Let each attempt fail independently with probability x. The expected number of attempts is

$(1 - x^u) / (1 - x)$

This is the mean of a geometric count capped at u. Under that assumption of independence it is close to 1 for small x. Section 3's correlation cannot make it smaller, since the expected count is convex in x.

We apply an amplification analysis to the adversarial case. OWASP lists Denial of Wallet as an example of unbounded consumption, and recommends rate limiting and user quotas [6]. The application to verification loops is ours.

An adversary who controls a message can, by assumption, make the model's proposal fail verification on every attempt with high probability. Each such message then costs the defender $u \cdot c$ at the uncached price. Where the provider caches the prefix, the first attempt may pay Section 5's write multiplier, the price of storing it. Each later attempt pays the read multiplier, the price of reading it back, while the entry lives. At Section 5's multipliers, caching then cuts the prefix's cost over two or more attempts at the five-minute lifetime, and over three or more at the one-hour. After the last attempt, the

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

message costs one escalation to a person, at attention cost h. If the adversary's cost for each message is a, the amplification ratio at the uncached price is

$$(u \cdot c + h) / a$$

If each attempt adds the refused proposal and the refusal to the context, $u \cdot c$ is a lower bound. By our reasoning, the ratio can be orders of magnitude above one. The cost c includes the whole assembled context, which in a long-running agent is large. The cost a is that of sending a short message. The ratio scales with the configuration choices that make a context long. The loop does not create it: one attempt already gives the ratio $(c + h) / a$, and the loop multiplies its token term by u or more.

By our reasoning, each attempt is another try at a gap in the validator. A refusal should therefore say no more than which rule failed. A limit on attempts bounds the amplification ratio for each message. It does not bound the defender's total spend, since the adversary can send more messages.

We argue that what bounds spend across messages is a budget for each sender over a period, enforced before the first attempt by an admission check (Part 3). The check is on estimated cost. The budget does not lower the ratio for each message. Escalations from one sender must be capped in the same way, or the attack moves from tokens to a person's attention. The person escalated to, and the code that enforces the budget, carry the limits Part 4 argues.

The bound depends on who is charged. Suppose the sender charged is the party the work acts for, with an identity verified where the message enters. For direct senders, the budget caps the defender's spend on each sender at about the budget in each period, at each agent that sender reaches, whatever the number of messages. The adversary's gain for each identity at each agent is then at most about the budget, times periods elapsed, divided by the cost of a verified identity. That bound assumes verified identities are not free to mint.

For indirect channels the identity cost does not apply. Text that arrives inside a tool's result or a retrieved document is charged to the party the work acts for. An adversary who plants such text at cost a can, by our reasoning, spend the budget of a legitimate sender who did not send it. The result is a denial of that sender's service. By our reasoning, one planted document that the work of many senders reads can spend the budget of each of them.

5. Cache Economics

In our reading, a provider's cache reuses only an unchanged prefix of the request (Part 1, Appendix A.7). Let a request's input be a prefix followed by a variable remainder. The prefix is a fraction s of the tokens and is identical across requests. Let w be the write multiplier and ρ the read multiplier on the base input price. Let P be the probability that the prefix is still cached when the request arrives. The expected input cost, relative to an uncached request, is

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

Caching pays when

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

$P > (w - 1) / (w - \rho)$

Anthropic prices a cache write at 1.25 times the base input price for a five-minute lifetime and 2 times for a one-hour lifetime [7]. A cache read costs 0.1 times, or 0.05 or 0.025 times on some models [7]. At 1.25 and 0.1 the threshold is $P > 0.22$. That is the worst case for the five-minute lifetime: at 0.05 and 0.025 it is 0.21 and 0.20. For the one-hour lifetime at 0.1 it is about 0.53.

If requests arrive as a Poisson process of rate λ, and each use refreshes a lifetime T, then

$P = 1 - e^{(-\lambda T)}$

The Poisson assumption does not describe bursty traffic, in which gaps between requests cluster.

The author of one production agent argues that the cache hit rate is “the single most important metric for a production-stage AI agent” [8]. In a post of July 2025 he reports an average ratio of input to output tokens of about 100 to 1 in that product.

That is one vendor’s account of one product, and its text gives no measurements behind its figures.

By our reasoning, the saving, as a share of input cost, does not fall with session length, except when a compaction rewrites the prefix. Earlier turns are read from the cache if an earlier request wrote them, everything before them is unchanged, and the entry has not expired. Anthropic limits how far back a read looks for an entry [7]. What expires the cache is the gap between requests.

Part 1, Section 4, The Couplings, reports from Anthropic’s page that a change to a tool’s definition invalidates the cached prefix, and a change to some model settings can invalidate part or all of it. Keeping the declared tool definitions fixed, as Part 1 describes, is therefore a cost decision as well as a design one.

Narrowing the subset of tools an agent may propose keeps the cache only if the narrowing leaves the cached prefix unchanged. A check by the validator is one such narrowing. A narrowing loses the cache if it is sent as changed tool definitions. The same applies to anything else that would change the prefix. In our reading, caching across a varying response schema depends on the provider. The author cited above warns that a timestamp at the start of the system text, or serialisation that is not deterministic, can invalidate the cache [8].

6. Repeating a Rule at the Point of Action

Part 1 places the act-level rules again at the point of action (its Section 5). In the hypothesis of Part 2, this sets the distance from a rule’s last presentation close to zero. The cost is at least the repeated tokens, uncached on every act (Part 1, Appendix A.7).

Part 2 reports findings on the position of evidence, not of a rule. It reports later work suggesting the position effect may be weaker or absent in some newer models, on the tasks tested, within limits it states. In that case repetition may pay little. The suite decides.

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

Repetition moves only context settings. By Part 3, it cannot make any property held.

7. What a Run Reports, and What the Levers Rest On

We argue that each lever that works on the context should be run on and off, so that it stays only while it pays. Repetition at the point of action is one. Two more should be run the same way: presenting the rules again word for word after a compaction, which Part 2 describes, and the thresholds for offloading, which Part 1 names.

Each run reports its cost: tokens for each layer, tokens added at the point of action, compactions, offloads, refusals, further attempts where the runtime has them, escalations and cache hits.

The levers rest on evidence as it stood on the date Section 1 gives. Part 2 reports the evidence that performance can fall as input grows. Part 3 states its belief that no wording can be relied on to keep data apart from instructions. The error floor of Section 3 rests on an assumption of ours, with one study within a model and one across models as evidence [4], [5].

8. Five Tests for Any Agent Platform

A buyer can put these tests to any runtime. A runtime that cannot support a test has answered it.

1. **Edit around the mechanism.** Attempt an act the rules forbid, through every channel the agent's identity reaches. Include editing or linking an existing item where creating one is refused. 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. No act outside the allowed set may result. No check that allows an act may take the fact from the note instead of its source. How often the model is misled among its allowed acts is a rate, and is reported as one.
3. **Reassemble a handling.** Rebuild a handling's context from its recorded inputs (Part 4, Proposition 2b) and compare it with what was recorded.
4. **Starve the validator.** 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 properties, by rule.

Conclusion

Constraining the act lets a validator hold a property of the proposal's form. Whether it costs the model's reasoning is disputed. Sampled agreement can buy a signal, not correctness. A limit on attempts buys a bound for each message. A budget for each sender buys a bound in each period on the spend charged to that sender. We argue that one planted document can spend legitimate senders' budgets. A cache saves cost while requests arrive often enough for its lifetime. We argue that repetition should be kept only while measurement shows it pays.

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

None of this says what a runtime must do for each promise to rest on a named mechanism, or against which adversary. That question remains.

Glossary

Term	Meaning
Amplification ratio	The defender's cost for one adversarial message, in attempts and in a person's attention, over the adversary's cost of sending it
Budget for each sender	A limit on what the defender spends on work charged to one sender over a period, enforced before the first attempt
Denial of Wallet	An example OWASP lists under unbounded consumption
Error floor	The error a simple majority of samples keeps in the limit of many samples, under the assumption Section 3 states
Latent error rate	The probability, for one input, that one sample of the model's decision is wrong
Lever	A design choice whose effect and cost can be modelled: a setting, or a mechanism built on settings
Read and write multipliers	What a provider charges to read from a cache or to write to it, as multiples of its base input price
Sampled agreement	Sampling a decision several times and acting only if a majority agrees
Verification loop	A loop in which a proposal the validator refuses goes back to the model for another attempt, up to a limit

References

[1] 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." 2024. Available: https://arxiv.org/abs/2408.02442

[2] W. Kurt, "Say what you mean: A response to 'Let Me Speak Freely'." The .txt blog, Nov. 2024. Available: https://blog.dottxt.ai/say-what-you-mean.html

[3] S. Geng *et al.*, "JSONSchemaBench: A rigorous benchmark of structured outputs for language models." arXiv:2501.10868 (v3, 27 February 2025), 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2501.10868

[4] L. Chen *et al.*, "Are more LLM calls all you need? Towards the scaling properties of compound AI systems," in *Advances in neural information processing systems*, 2024. Available: https://arxiv.org/abs/2403.02419

[5] E. Kim, A. Garg, K. Peng, and N. Garg, "Correlated errors in large language models." 2025. Available: https://arxiv.org/abs/2506.07962

[6] OWASP, "LLM10:2025 unbounded consumption." OWASP Top 10 for LLM Applications 2025, 2025. Available: https://genai.owasp.org/llmrisk/llm102025-unbounded-consumption/

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

[7] Anthropic, "Prompt caching." Claude Platform documentation. Accessed: Nov. 15, 2026. [Online]. Available: https://platform.claude.com/docs/en/build-with-claude/prompt-caching

[8] Y. Ji, "Context engineering for AI agents: Lessons from building Manus." Manus blog, 18 July, 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus

Revision History

Version	Date	Change
1.0.0	2026-11-19	First publication
1.0.1	2026-11-20	Part 1 named by its new title, Context Is Configured

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.