VELOCITY EDGE

CONTEXT ENGINEERING, PART 1

Context Is Configured

ABSTRACT

Over a long session, the context an AI agent reads changes as the agent works. This first part of the Context Engineering series models that context as the output of code. In this part's model, the runtime (the software that runs the agent) assembles a context for each call to the model. A model proposes an action from it. The action takes effect only if mechanisms outside the model allow it. This part writes the assembly of the context as a function. The function's output, the context, can be recorded item by item.

Ten settings, named in this part, configure that function and the harness (all other code around the model). The list is not claimed to be complete. Six are context settings, which decide what the model reads. Each of them can at most raise the chance of a desired behaviour. Withholding (leaving material out) is the one exception: what the context lacks cannot be found in it, though a model may still generate it from its training. The other four are harness settings. One of these, the outside (the mechanisms that decide on an act before it happens), can make an undesired act impossible, under conditions this part leaves open. The part ranks the kinds of material that instruct, to settle conflict. It sets the order in which kinds of material sit in the context. It also says what wording still adds.

This part runs no experiment and makes no measurement. Its scope is agents whose models are transformer-based. It states the thesis that later parts argue.

Phillip Pienaar
Velocity Edge

2026-11-19
Version 1.1.0

Contents

Executive Summary 3
1. The Question 3
2. The Agent Step 4
3. The Assembly Function 5
4. Ten Settings of Two Kinds 5
5. Layers and Their Two Orders 8
6. Where the Words Still Matter 9
Conclusion 9
Glossary 9
References 11
Appendix A: How a Transformer Reads Its Context 14
Revision History 27
Licence and Disclaimer 28

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

Executive Summary

An AI agent that works for hours reads a context that keeps changing. In the model this part sets out, that context is the output of code. The code builds it for each call to the model.

The part names ten settings that govern what the model reads and what happens to what it proposes. A setting is a property of that code, or of the harness, which is the rest of the code around the model. It is never a property of the model's weights. Its value can be varied and read back. The ten are of two kinds. Context settings change what the model reads, and can only make a desired behaviour more likely. The one exception is withholding: what is never placed in the context cannot be taken from it. A model may still produce it from what it learnt in training. Harness settings are set by the harness, not the assembly (the code that builds the context). One of them, the outside (the mechanisms that decide, before an act, whether it may happen), can make an undesired act impossible. This part does not set out the conditions under which it can. We do not claim the ten are exhaustive.

The two kinds differ in what they can achieve. A context setting changes the distribution of what the model proposes, that is, how likely each proposal is. The outside can change the set of acts that can result. Which acts it can refuse, and under what conditions, is not settled here.

The part runs no experiment and measures nothing. Its model is of agents built on transformer-based language models. Most of the evidence this series cites, and its reasoning about cost, depend on that scope. The outside does not depend on it.

1. The Question

In September 2025 Anthropic's applied AI team wrote that work with language models was becoming less a search for the right wording of a prompt (instructions written for a model) than a broader question [1]. That question, they wrote, was "what configuration of context is most likely to generate" the behaviour wanted. By context they meant the tokens (units of text) a model is given when it produces output. By context engineering they meant the ways of selecting those tokens well, and keeping the selection good, on every run of the model.

Vendors' guidance on context engineering, and one academic survey, have described such strategies [1], [2], [3]. Among the strategies the vendors' posts describe are summarising a context to shorten it, note-taking, retrieval, and sub-agents, which do part of the work in a context of their own. The two vendors' posts also state those strategies' costs and trade-offs.

We have not found where the three sources say precisely what those strategies can and cannot guarantee. Nor have we found in them a criterion for when a strategy inside the context should give way to a mechanism outside it. These are claims about the three sources, read on 15 November 2026. They are not claims about other fields. In computer security, a check, outside a program, that decides what the program may do

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

is long established [4], [5]. Recent work on prompt injection applies the idea to agents built on language models [6].

In our view, what a strategy can guarantee is hardest to say for a long-running agent. An agent that runs for hours accumulates material and may summarise it. It acts on external systems many times. Every one of those events changes the context in which the agent's rules are read.

The question is not which context is best. It is what remains true as the context changes, and what can be made to remain true regardless.

2. The Agent Step

In this part's model, the runtime is the software that runs an agent. The assembly is the runtime's code that builds the context. The harness is the rest of the code around the model. An agent is a loop in which a model, given a context, proposes an action, and the runtime decides whether and how to perform it. Long-running means that the agent's work spans more context than one well-engineered request: many model calls, accumulating state (results and notes), and actions on external systems. A rule is a statement of what the agent must or must not do.

For one message, four things happen in order.

1. The runtime builds a context, C.
2. A model, M, draws a proposal from that context by chance: $p \sim M(C)$.
3. Mechanisms outside the model decide on the proposed action. We call them G. They reach a decision $d = G(p, g)$ from the proposal and from other inputs, g.
4. The action takes effect only if d allows it.

This part does not say what the inputs g are.

The step allows the model to ask to read something, such as a file, before it proposes. Whether a read is itself an act, and how a read is checked, this part does not settle.

The outside mechanisms decide, before an act, whether it may happen. They do not read the model's reasoning. Some are plain code in the runtime: a check on the form of a proposal, and a last check that decides whether the proposed action is performed. Others are permissions in an external system.

In this part's model, of everything the runtime does in this step, only the model's sampled outputs are drawn by chance. Those outputs include its requests to read, its proposal and any summary a model writes for the context. Everything else the runtime does is code, and can be made deterministic, giving the same result each time. This is not a statement about what an external system returns.

This part's model is of agents built on transformer-based language models. Most of the evidence this series cites, and its reasoning about cost, depend on that scope. The outside mechanisms do not depend on it.

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

3. The Assembly Function

In this part's model the context is the output of code. A handling is the runtime's processing of one message, from receipt to its recorded outcome. It may include several model calls. We write the code that builds the context for one model call as an assembly function:

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

Writing the context as an assembly function is not new. One survey, in July 2025, writes context as the output of an assembly function over its components [3]. One vendor's post, in July 2026, offers a line of pseudocode, a function of its inputs, as one way to describe the building of a context [7].

The function has five inputs.

- $D@v$ is the declared material at revision v. Declared material is what is written down for the agent and kept under version control: the rules, the agent's instructions, the domain documents and the tool definitions.
- m is the message.
- R is the record of earlier handlings of the same piece of work.
- W is the working state accumulated within the handling: tool results and intermediate notes.
- κ is the configuration: the settings that govern the assembly and the harness. Section 4 sets them out.

This part's model allows two cases for the working state. In the first, the language model's requests to read are answered while it forms its proposal, and W need not be empty. In the second, each call to an external system is itself an action that the outside mechanisms decide on. Its result then reaches the agent as a later entry in the record. In that case W is empty within a handling, and the working state lives in R.

A is code. The configuration decides which items of material enter the context, in what order and at what size. The language model does not decide that. The one exception is a choice the configuration delegates to it. A configuration may delegate, for example, by letting the language model request a read, or by letting it write state through an action the configuration declares. The configuration does not always decide what an item says: a summary a model writes is itself a sampled output.

The output of A can be recorded in a manifest, which names every item, its version and its size.

4. Ten Settings of Two Kinds

In this part's model, the configuration κ is a vector of settings. A property is admitted as a setting if it meets three conditions.

1. It is a property of the assembly or of the harness, never of the model's weights.
2. It is defined independently of the others, and its value can be varied while the values of the others are recorded.

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

3. Its value is visible in the manifest or in the declared configuration, so that its value in a handling can be read back.

Ten properties meet the three conditions. We do not claim the list is complete.

Context settings decide what the model reads. There are six: composition, position, size, timing, source and labelling, and lifespan. Harness settings are set by the harness, not the assembly. There are four: output shape, tool set, model settings and the outside.

Table 1 uses five terms. The window is the context of one call, as the ordered span its entries fill. A layer is one of the seven kinds of material that composition names. Carried notes are notes brought forward from earlier work. Offloading moves large material out of the window, to a file. The point of action is the end of the context, just before the proposal.

Table 1: The ten settings. Source: this part's model.

	Setting	Kind	What it decides
1	Composition	Context	Which layers enter the window: rules, agent instructions, domain material, plan (the agent's own record of its intended steps), task (the message being handled), carried notes, working state
2	Position	Context	The order of the entries in the window, and what sits at the end of the context, nearest the proposal
3	Size	Context	The budget for each layer, and the thresholds at which material is offloaded or summarised
4	Timing	Context	When each item enters; in particular, whether a rule is presented only at the start or again at the point of action
5	Source and labelling	Context	Where each item comes from, at which version, and the label it carries: instruction, or data with its origin
6	Lifespan	Context	How long a context lives before a fresh context from a record replaces it, or a summary shortens it
7	Output shape	Harness	The form the proposal must take; in the strongest case, one action from an enumeration, with typed arguments
8	Tool set	Harness	Which tools are declared to the model
9	Model settings	Harness	The model and its version, and how its output is drawn: temperature, sampling and reasoning effort
10	The outside	Harness	The outside mechanisms: plain-code checks in the runtime, and permissions in an external system

Tool set and model settings are harness settings because the harness sets them. Neither can refuse an act. Tool definitions enter the window. Model settings change how likely each proposal is.

The outside, the tenth setting, is the set of outside mechanisms of Section 2. A scripted check that runs over the record after the act is not part of the outside.

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

What Each Kind Can Do

In this part's model, the two kinds differ in what they can achieve. Context settings change the distribution of proposals, and can only make a desired behaviour more likely. The outside can change the set of acts that can result. The conditions under which it can are not set out here. Output shape and tool set shape the proposal that the outside decides on.

One exception follows from the series' own reasoning, not from a source. What is never placed in the context, by the assembly or by a read, cannot be taken from it. A model may still produce it from what it learnt in training. So withholding removes one route to a behaviour. It does not close the others.

The Couplings

In this part's model, changing one setting can change another.

- Repeating a rule at the point of action is a choice of timing. It adds tokens to the window, which is size.
- Thresholds for offloading are a choice of size. They change what the working state contains, which is composition.
- A cache stores data for reuse. A request's prefix is its start. In one provider's documentation, as read on 16 November 2026, the provider can cache the prefix [8]. It says that modifying a tool's definition in a request's list of tools invalidates that cached prefix. The same documentation says a change to some model settings can invalidate part or all of it. We read this as coupling the tool set and some model settings to position, through the cached prefix.

An evaluation that varies one setting must keep the coupled ones fixed or record them.

The Declared Tool Set and the Subset an Agent May Propose

In this part's model, a stable tool set and a narrow one pull against each other. This part's model separates the declared set from the subset the agent may propose. The declared tools stay fixed for the life of an agent's revision, so that the tool definitions do not break the cached prefix. The subset the agent may propose in a given state is narrowed in the schema (the stated form) that a proposal must match. A plain-code check tests the finished proposal against it. Whether a schema that changes with the state leaves a cached prefix intact, this part does not settle.

The builders of one production agent gave related advice in July 2025, under the heading "Mask, Don't Remove" [9]. Their rule is to leave the list of tools alone in the middle of an iteration unless a change is unavoidable. They report narrowing the choice of tool by masking, which blocks or forces some choices while the model's output is being produced.

Their advice differs from this section's separation, which checks the finished proposal in plain code.

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

5. Layers and Their Two Orders

In this part's model, the composition setting names seven layers: rules, agent instructions, domain material, plan, task, carried notes and working state. Two orders apply.

Precedence is the order used to resolve conflict. It ranks the layers that instruct: rules, then agent instructions, then the task. The task is the message being handled. It may come from an external sender. It is ranked below the rules. Ranking instructions by their source is not new: one study trains models to follow such a ranking [10]. Nor is marking external input: another study transforms such input to signal where it came from [11].

Domain material, the plan, carried notes and working state are labelled as data. Precedence gives them no authority to instruct. Like any context setting, a label only makes it less likely that the model follows instructions inside them. We label domain material, the agent's documents, as data, since a document can carry instructions that a third party put there.

Position is the order in the window. It has nine entries: rules, agent instructions, tool definitions, domain material, plan, carried notes, working state, task, and the act-level rules (rules about one kind of act), repeated at the point of action. The repeated rules are for the tools the agent may propose. Tool definitions and the repeated rules are not layers.

The stable entries come first. They are built from declared material: rules, agent instructions, tool definitions and domain material. We put them first because they change least and so can be cached. Placing the stable entries first assumes that a provider caches the prefix of a request.

One vendor's post also recommends stable content first [7]. Another advises keeping the prefix of a request stable [9]. The first post's cited passage gives no figure to support its cache reason. It concedes that its advice "assumes everything fits". Where agents do the work, it adds, that is often not so. The first post gives a second reason for its order, from how models treat the start and end of their input. This part does not rely on that reason.

The plan, carried notes and working state follow the stable entries. The task comes after them. Those act-level rules are repeated last, at the point of action.

The task sits near the proposal. Even so, on conflict it ranks below the rules.

We treat position as a setting whose effect must be measured for each model, not assumed.

One provider, as read on 16 November 2026, documents its cache prefix as covering tools, then system text (a request's standing instructions), then messages [8]. The sentence cited does not say how a whole request is laid out. Where the sections of a request have a fixed order, this part's model sets position within each section. The manifest then records the order within each section. This part claims no single order of tokens.

Product behaviours cited in this part are as of the dates given. Each may change with a release.

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

6. Where the Words Still Matter

This part's stance is that the response to an agent's failures is not better prompts alone. It is configuration. Wording still matters, and what it contributes is precision.

- A rule is one testable clause, so that a script or a reviewer can decide whether it was kept.
- A refusal, the reply an agent gets when an action is not allowed, names the rule it applies.
- The field names of a schema say what belongs in them.
- An agent's instructions are tested on a fixed set of cases before it ships, and again before any change of model.

One report runs against this part. In January 2026 Cursor gave an account of its long-running coding agents [12]. In April 2026 Osmani relayed it [13]. In that account, much of how the system behaves comes down to "how we prompt the agents", more than its authors expected. Cursor ranks the prompts above the harness and the models. It says both matter too.

Neither source reports measuring that ranking.

We read the prompts in question as including how work is divided among roles and each agent's composition setting. Both are configuration in this part's model. Neither source says so. Even on our reading, the report is evidence against the strong form of this part's stance: that configuration alone decides how an agent behaves.

Conclusion

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. That is the thesis of this series. This part does not argue it. It sets out a model. Appendix A explains how models read context.

In that model, an agent's context is the output of code, produced under the configuration. Context settings can only make a desired behaviour more likely. The one exception is withholding, with the limit Section 4 gives. Each context setting named here can be varied and read back. Output shape, tool set and model settings shape the proposal. The outside decides which proposed acts take effect.

Suppose an agent is given a rule, and hours later it stops following it. Why would that be?

Glossary

Term	Meaning
Act-level rule	A rule about one kind of act, such as the use of one tool
Agent	A loop in which a model, given a context, proposes an action, and a runtime decides whether and how to perform it
Assembly function	The code that builds the context from the declared material at a revision, the message, the record of earlier handlings and the working state, under the configuration

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

Term	Meaning
Carried notes	Notes brought forward from earlier work
Configuration	The vector of settings that governs the assembly function and the harness around the model
Context	What the model is given to read on one call: the output of the assembly function, produced under the configuration
Context settings	The settings that decide what the model reads: composition, position, size, timing, source and labelling, and lifespan
Declared material	What is written down for the agent and kept under version control: the rules, the agent's instructions, the domain documents and the tool definitions
Declared tool set	The tools declared to the model, fixed for the life of an agent's revision; the subset an agent may propose in a given state may be narrower
Handling	The runtime's processing of one message, from receipt to its recorded outcome; it may include several model calls
Harness	The code around the model, other than the assembly function
Harness settings	The settings set by the harness, not the assembly: output shape, tool set, model settings and the outside
Layer	One of seven kinds of material in the window: rules, agent instructions, domain material, plan, task, carried notes or working state
Long-running	Of an agent: its work spans more context than one well-engineered request, with many model calls, accumulating state (results and notes), and actions on external systems
Manifest	The record of the context assembled for one model call, naming every item, its version and its size
Outside mechanisms	The mechanisms that decide, before an act, whether it may happen, without reading the model's reasoning: plain code in the runtime, and permissions in an external system. As a setting, they are called the outside
Point of action	The end of the context, just before a proposal
Position	The order of the entries in the window: the seven layers, the tool definitions, and the act-level rules repeated at the point of action
Precedence	The order used to resolve conflict between layers: rules, then agent instructions, then the task
Proposal	The model's sampled output that names the action it proposes
Revision	A version of the declared material
Rule	A statement of what the agent must or must not do
Runtime	The software that runs an agent; the assembly function is part of it
Setting	A property of the assembly or of the harness that meets the three conditions of Section 4
Window	The context of one call, as the ordered span its entries fill

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

Term	Meaning
Working state	Tool results and intermediate notes: accumulated within a handling or, where each external call is itself an action, kept in the record

References

[1] P. Rajasekaran, E. Dixon, C. Ryan, and J. Hadfield, "Effective context engineering for AI agents." Anthropic, 29 September, 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://www.anthropic.com/research/effective-context-engineering-for-ai-agents

[2] The LangChain Team, "Context engineering." LangChain blog, 2 July, 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://www.langchain.com/blog/context-engineering-for-agents

[3] L. Mei *et al.*, "A survey of context engineering for large language models." arXiv:2507.13334, 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2507.13334

[4] J. P. Anderson, "Computer security technology planning study," Electronic Systems Division, US Air Force, ESD-TR-73-51, Vol. I, Oct. 1972. Accessed: Nov. 15, 2026. [Online]. Available: https://apps.dtic.mil/sti/pdfs/AD0758206.pdf

[5] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, 1975, Accessed: Nov. 15, 2026. [Online]. Available: https://web.mit.edu/Saltzer/www/publications/protection/Basic.html

[6] 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

[7] J. Mills, "Context assembly: Building the prompt the model actually sees." Redis blog, 22 July, 2026. Accessed: Nov. 15, 2026. [Online]. Available: https://redis.io/blog/context-assembly-building-the-prompt-the-model-sees/

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

[9] 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

[10] 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

[11] 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

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

March 2024), 2024. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2403.14720

[12] W. Lin, "Scaling long-running autonomous coding." Cursor blog, 14 January, 2026. Accessed: Nov. 15, 2026. [Online]. Available: https://cursor.com/blog/scaling-agents

[13] A. Osmani, "Long-running agents." addyosmani.com, 28 April, 2026. Accessed: Nov. 15, 2026. [Online]. Available: https://addyosmani.com/blog/long-running-agents/

[14] T. B. Brown *et al.*, "Language models are few-shot learners." arXiv:2005.14165 (v4, 22 July 2020); NeurIPS 2020, 2020. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2005.14165

[15] P. Pienaar, "Corpus engineering, part 1: What an AI is trained on, and why it matters." Velocity Edge, version 1.1.0, 2026. Available: https://velocityedge.io/papers/corpus-engineering

[16] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback." arXiv:2203.02155 (v1, 4 March 2022); NeurIPS 2022, 2022. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2203.02155

[17] M. Sclar, Y. Choi, Y. Tsvetkov, and A. Suhr, "Quantifying language models' sensitivity to spurious features in prompt design or: How I learned to start worrying about prompt formatting." arXiv:2310.11324 (v2, 1 July 2024); ICLR 2024, 2024. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2310.11324

[18] 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

[19] H. Chang, Y. Jun, and H. Lee, "ChatInject: Abusing chat templates for prompt injection in LLM agents." arXiv:2509.22830 (v3, 13 April 2026), 2026. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2509.22830

[20] C. Guo *et al.*, "IH-Challenge: A training dataset to improve instruction hierarchy on frontier LLMs." arXiv:2603.10521 (v1, 11 March 2026), 2026. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2603.10521

[21] E. Zverev, S. Abdelnabi, S. Tabesh, M. Fritz, and C. H. Lampert, "Can LLMs separate instructions from data? And what do we even mean by that?" arXiv:2403.06833 (v3, 31 January 2025); ICLR 2025, 2025. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2403.06833

[22] 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

[23] 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

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

[24] 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

[25] C. Anil *et al.*, "Many-shot jailbreaking." Advances in Neural Information Processing Systems 37 (NeurIPS 2024), pp. 129696–129742, 2024. Accessed: Nov. 16, 2026. [Online]. Available: https://doi.org/10.52202/079017-4121

[26] A. Vaswani *et al.*, "Attention is all you need." arXiv:1706.03762 (v7, 2 August 2023); NeurIPS 2017, 2017. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/1706.03762

[27] C. Olsson *et al.*, "In-context learning and induction heads." Transformer Circuits Thread 2022; arXiv:2209.11895 (v1, 24 September 2022), 2022. Accessed: Nov. 16, 2026. [Online]. Available: https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html

[28] M. Geva, R. Schuster, J. Berant, and O. Levy, "Transformer feed-forward layers are key-value memories." arXiv:2012.14913 (v2, 5 September 2021); EMNLP 2021, 2021. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2012.14913

[29] P. Veličković, C. Perivolaropoulos, F. Barbero, and R. Pascanu, "Softmax is not enough (for sharp size generalisation)." arXiv:2410.01104 (v3, 30 May 2025), 2025. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2410.01104

[30] F. Barbero *et al.*, "Transformers need glasses! Information over-squashing in language tasks." arXiv:2406.04267 (v2, 24 October 2024); NeurIPS 2024, 2024. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2406.04267

[31] G. Kobayashi, T. Kuribayashi, S. Yokoi, and K. Inui, "Attention is not only a weight: Analyzing Transformers with vector norms." arXiv:2004.10102 (v2, 6 October 2020); EMNLP 2020, 2020. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2004.10102

[32] X. Gu *et al.*, "When attention sink emerges in language models: An empirical view." arXiv:2410.10781 (v2, 2 March 2025); ICLR 2025, 2025. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2410.10781

[33] A. Q. Jiang *et al.*, "Mistral 7B." arXiv:2310.06825 (v1, 10 October 2023), 2023. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2310.06825

[34] Gemma Team, "Gemma 3 technical report." arXiv:2503.19786 (v1, 25 March 2025), 2025. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2503.19786

[35] A. Haviv, O. Ram, O. Press, P. Izsak, and O. Levy, "Transformer language models without positional encodings still learn positional information." arXiv:2203.16634 (v2, 5 December 2022); Findings of EMNLP 2022, 2022. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2203.16634

[36] X. Wu, Y. Wang, S. Jegelka, and A. Jadbabaie, "On the emergence of position bias in transformers." arXiv:2502.01951 (v4, 9 August 2025); ICML 2025, 2025. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2502.01951

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

[37] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, "RoFormer: Enhanced transformer with rotary position embedding." arXiv:2104.09864 (v5, 8 November 2023), 2021. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2104.09864

[38] Y. Chen, A. Lv, J. Luan, B. Wang, and W. Liu, "HoPE: A novel positional encoding without long-term decay for enhanced context awareness and extrapolation." arXiv:2410.21216 (v2, 5 December 2024), 2024. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2410.21216

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

[40] C.-Y. Hsieh *et al.*, "Found in the middle: Calibrating positional attention bias improves long context utilization." arXiv:2406.16008 (v2, 3 July 2024), 2024. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2406.16008

[41] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks." arXiv:2309.17453 (v4, 7 April 2024); ICLR 2024, 2024. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2309.17453

[42] K. Li *et al.*, "Measuring and controlling instruction (In)stability in language model dialogs." arXiv:2402.10962; COLM 2024, 2024. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2402.10962

[43] V. Dongre, J. Hsieh, V. D. Lai, S. Yoon, T. Bui, and D. Hakkani-Tür, "When attention closes: How LLMs lose the thread in multi-turn interaction." arXiv:2605.12922, version 2, 6 October 2026, 2026. Available: https://arxiv.org/abs/2605.12922

[44] C.-P. Hsieh *et al.*, "RULER: What's the real context size of your long-context language models?" arXiv:2404.06654 (v3, 6 August 2024); COLM 2024, 2024. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2404.06654

[45] Y. Chen *et al.*, "Reasoning models don't always say what they think." arXiv:2505.05410 (v1, 8 May 2025), 2025. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2505.05410

[46] R. Pope *et al.*, "Efficiently scaling transformer inference." arXiv:2211.05102 (v1, 9 November 2022); MLSys 2023, 2023. Accessed: Nov. 16, 2026. [Online]. Available: https://arxiv.org/abs/2211.05102

[47] H. He, "Defeating nondeterminism in LLM inference." Thinking Machines Lab: Connectionism, 10 September, 2025. Accessed: Nov. 16, 2026. [Online]. Available: https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/

Appendix A: How a Transformer Reads Its Context

Overview

The Context Engineering series is about what goes into an AI agent's context: its rules, its tools, the documents it reads, and how all that is ordered, trimmed and checked. The series says what to do. This appendix gives the mechanics behind that advice, from

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

how the model underneath works. It is the one place in the series that points forward: it maps each mechanism to the part that uses it.

The case for enforcing outside the model rests on two reasons, of different strength:

1. **From the design, and from attacks on what training adds to it.** Rules, user messages, tool results and an adversary's text all reach a model as one stream of text, and nothing in the architecture keeps them apart. Models are trained to tell them apart, and that training helps. Some training-based defences of this kind have been bypassed by adaptive adversaries in published evaluations. The one stream is a fact of design. The bypassing is a finding (A.3).
2. **Plausible, with partial evidence.** As a session grows, a rule the model keeps only by reading can lose its effect. Several mechanisms could cause this, and studies show effects on retrieval and on keeping instructions. How much it matters for a given model and rule has to be measured.

Parts 3 and 4 make the case that every property a check can compute should be enforced outside the model, whether or not there is an adversary. A.2 sets out the inputs such a check must compute from. By this appendix's reasoning, the case is strongest where untrusted parties write into the context and a breach is costly to undo. Reason 2 adds weight in long sessions. Part 3 gives the criterion for which properties are held, in its sense, and the assumptions this needs. Part 4 gives the assumptions under which confinement follows.

A property no check can compute, such as tone or what free text means, is left to the model. There are two exceptions. A stricter property may be held in its place, in Part 3's sense. Or, as Part 3 has it, a mechanism that part leaves aside may keep the property itself. Some of the act-level properties that Part 3 calls counted can be checked by a script on the proposed action. For these, Part 4 gives a procedure to estimate, on a fixed suite, how often each is broken. The series itself runs no such measurement. For other properties left to the model, Part 4 gives no such procedure.

By this appendix's own reasoning, and not Part 4's, two things decide how much this matters: whether anyone untrusted writes into the context, and what a breach costs to undo. Where no one untrusted writes, and a breach is cheap to undo, a measured rate of rule-keeping can be control enough. Where an adversary writes and adapts, no measured rate is.

Every paragraph keeps three things apart:

- **What the architecture is.** These are facts of design, cited to the papers that set them out.
- **What studies have measured.** These are findings, at the strength their authors give, and for the models they tested.
- **What we infer from the two.** This is marked as the series' own reasoning.

A.1 What a Model Is, and What It Is Not

A language model is a function with a very large set of numbers, its *parameters* or *weights*. Training sets the weights. Using the model does not change them. Brown and

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

colleagues, for example, applied GPT-3 to new tasks "without any gradient updates or fine-tuning" [14].

Two qualifications matter for agents:

- **A provider may change what serves a model name.** By our reasoning, the weights are fixed for a version pinned by its identifier, not for a name. Part 4 cites the provider pages, which, as opened on 15 November 2026, do not say that identical inputs give identical outputs.
- **Methods that update weights during use exist in research.** By our understanding they are not how the agents this series concerns are run.

A *token* is a word or a piece of a word. Given a sequence of tokens, the model computes a score for every possible next token. The software that serves the model turns the scores into probabilities, draws one token, appends it, and asks again. Everything the model writes is produced this way, one token at a time. That includes an agent's *proposal*: the action it puts forward, such as "pay invoice 4471".

By our reasoning, the model's context and its weights do different things:

- **The weights** carry what training put in: the patterns the model learned, what it can do, and its learned tendencies in how it responds.
- **The context** is the input the weights are applied to. It changes which learned patterns are drawn on, and so how likely each output is. It changes no weight.

A large model can do a task given only examples in its context, with no change to its weights. Brown and colleagues leave open whether the model learns the task there or recognises one it met in training [14]. Either way, by our reasoning, the effect lasts only as long as the examples stay in the context.

Training against reading. Corpus Engineering, a separate series by this series' author, draws the same line. What follows from it here is that author's position, not an independent finding. Its Part 1 says: "Context and controls do not change the parameters. In our reasoning, they can make what a system does less harmful, but leave the tendencies training gave the model as they were" [15].

That paper keeps a limit on its claim, and so does this one. It separates what a model *can* do from what it *chooses* to do. In our example, a model that can write a database delete learned that from its corpus, its training text. Whether it chooses to run one when told not to is a different question. The paper reads the evidence as showing the corpus shaping the first more clearly than the second. It holds as a belief that a corpus teaching deceit makes a model more likely to choose deceit. It calls its closest test of that belief inconclusive. It reads that test as leaning against the belief. It also notes that what a model chooses is shaped by what it is rewarded for in tuning. It examines two incidents, one reported by the UK AI Security Institute and one by OpenAI. It reads neither account as needing pre-training text that teaches deceit, and reads OpenAI's as pointing instead to what reward training shaped [15].

For an agent, the useful part is this, and it is our reasoning. Training fixed the mapping from any context to a distribution over outputs. A session chooses only the input to that mapping, and an adversary may choose part of that input. A rule in the context is one

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

more piece of input. Some inputs produce behaviour far from what the model does on most inputs. A.3 shows that this works for an adversary as well as for an operator.

A.2 Following an Instruction Is Itself Trained

A model does not obey its context the way a program obeys its code. By our reading of the design and of the studies below, it follows an instruction because training gave it a tendency to.

Pretraining is training on general text, and *tuning* is further training on chosen examples.

- **Pretraining alone gives the tendency only weakly.** Ouyang and colleagues note that a pretrained model can be prompted into following instructions, and that it does so unreliably [16].
- **Tuning strengthens it.** In the same study, human raters preferred the outputs of a small model tuned on demonstrations and human preferences to those of an untuned model with more than 100 times as many parameters. The comparison used prompts from the study's own distribution.
- **Wording still matters.** Sclar and colleagues changed only the format of prompts carrying a few worked examples, leaving the meaning alone. Accuracy moved by a median of 7.5 points, and by up to 76 in one model. Neither larger models nor instruction tuning removed the sensitivity. It appeared in open models up to 70 billion parameters and in GPT-3.5 [17].

So, by our reasoning, an instruction in the context works through habits training put in, and wording shifts the odds within those habits. This is the mechanical reason behind this part's line that a *context setting* (Section 4) "can only make a desired behaviour more likely". The exception is withholding, leaving material out of the context. What is not in the context cannot be taken from it. The model may still produce it from training.

A check differs in kind. The rest of this paragraph is the series' own reasoning. Take a rule such as "never pay an invoice over $10,000 without a matching purchase order". Written in the context, it is something the model reads, and keeps or drops by habit. Written as code between the agent's proposal and the payment system, it is a check. No text the agent reads changes what the check computes. Text can still shape what is submitted to it.

That last point sets the limits of a check, and they are worth naming:

- **It must compute from inputs no proposal chooses (Part 3), such as the purchase-order system rather than the supplier's invoice.** If it trusts the model's word that purchase order 8812 matches, a planted line can make the model cite a real order that belongs to another invoice. The invoice itself is written by the supplier, so it cannot be the trusted record either. The check must read the purchase-order system directly. Part 3 states the exceptions, and the assumptions: authenticated inputs, rules written only by trusted parties, an intact record, and a tool that does what it declares.
- **It enforces only what it computes.** Here that is: an open order exists for this supplier and this amount. Whether the invoice is genuine is another property. So is whether five payments of $9,000 add up to a breach, which needs a check across records.
- **It covers only acts routed through it.**

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

Failures to meet these limits are faults of the kind any software has, found by testing and by argument from the code, not by measuring the model. Part 3 sets out what a check can and cannot compute, the assumptions it needs, and what stays the model's choice within an allowed act. Part 4 gives the assumptions under which confinement follows.

None of this is new with this series. Designs that take control flow away from the model, so that untrusted data cannot alter it, are published work, and Part 3 reviews them [6], [18].

A.3 One Stream of Tokens

Everything a model reads is one sequence of tokens. That includes the *system prompt* (the standing instructions given by the operator, the party running the agent), the user's message, a retrieved document, a tool's result and the model's own earlier output.

What the design gives. Wallace and colleagues describe system, user, model and tool messages as formatted with special tokens, reserved markers, so the model can tell them apart [10]. In the standard models deployed as of November 2026 there is no separate channel for instructions and data: the markers sit in the same stream as the text they mark, and what they mean to the model is learned in training.

A serving stack can refuse to let user or tool text produce the reserved markers. That is done in code. By our reasoning, it protects the tokens, not the boundary. Chang and colleagues formatted text in tool results to mimic a chat template, and injection success on one agent benchmark rose from about 5% to about 32%. Templates from other models often raised success too [19].

What training adds. Wallace and colleagues argue that models often give a system prompt no more priority than text from untrusted users and third parties. They train models to rank instructions by role: system over user, and user over tool output. They report that this "drastically increases robustness", including against kinds of attack not seen in training, with little loss on standard tasks. They also report some over-refusal, and that their models are likely still open to strong attacks [10]. Later work reports that training on this hierarchy raised a frontier model's robustness by about 10 points on average across 16 benchmarks [20]. Training helps, and this appendix does not say otherwise.

What has been measured against it.

- **Separation is weak when measured.** Zverev and colleagues defined a measure of how well a model separates instructions from data, and found that all the models they tested fell short of high separation [21].
- **Planted instructions steer real applications.** Greshake and colleagues demonstrated that instructions planted in retrieved content can steer the model on Bing Chat. On GitHub Copilot the steering was less reliable. The authors tested both in their own setups, alongside synthetic applications. They did not measure how often. They argue that retrieval "blurs the line between data and instructions" [22].
- **On agents, injection is real and far from certain.** On the AgentDojo benchmark, attacks succeeded against the best agents in under 25% of cases, and in about 8%

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

with an injection detector added [23]. Those attacks were generic, not adapted to the defence.

- **Recent defences have fallen to adversaries who adapt.** Nasr and colleagues report bypassing 12 recent defences, some of them trained into the model. For most, the attack's success rate was above 90%. The study's threat model assumes an adversary with large computing resources who knows the defence's design. Part 3 sets out this evidence [24].

A separate result concerns trained behaviour, not operator rules. Anil and colleagues filled long contexts with hundreds of made-up examples of a model misbehaving. The attack grew more effective as a power law in the number of examples, against models including Claude 2.0, GPT-3.5 and GPT-4. The power law fitted across the range they tested, up to hundreds of examples. Fine-tuning delayed the attack and did not stop it. One prompt-level defence cut its success from 61% to 2%. That figure is from one test: a single category, at 205 examples [25]. By our reading, this is A.1's point in action: enough of the right input produces behaviour far from the model's usual.

Two points follow, and both are the series' own reasoning:

1. **Which instruction wins depends on what training made of the layout.** Layout alone settles nothing. Labelling a piece of text as data, under setting 5 of Section 4 (source and labelling), helps only as far as the model was trained to respect such labels.
2. **No wording can be relied on to keep data apart from instructions.** Wording is one more stretch of the same stream, read by the same weights. Part 3 states this as a belief, and says what would show it wrong.

A.4 Attention: How a Model Draws on Its Context

The transformer is the architecture behind current language models. Its central operation is attention [26].

The basic step. In this appendix, a position is a token's place in the sequence, not the position setting of Section 4. Think of each position in the text as asking a question of everything before it. Every earlier position, and the position itself, offers a label saying what it holds, and a content to pass on. The model scores how well each label matches the question, shares out a fixed total of 1, giving sharply more to higher scores, and takes that mix of the contents.

In the paper's terms:

- At every position, in every network layer, the model forms a *query* from its current representation of that position.
- Every position up to and including this one offers a *key* and a *value*.
- The model scores the query against each key.
- The softmax function turns the scores into weights. By its definition, softmax makes the weights sum to exactly one.
- The model takes the weighted average of the values.

One such operation is a *head*. That average is what this head brings forward from the context to this position.

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

In a model that writes left to right, as every model in this series does, a position cannot see anything after it. Vaswani and colleagues implement this in their decoder with a *causal mask*, which sets the scores of later positions so that their weights are zero [26].

Heads, network layers and the residual stream. A model runs many heads side by side in each *network layer*, and stacks many network layers. A network layer is a stage of the model's computation, not a layer of the context in Section 5's sense.

- **Heads specialise.** Different heads learn to do different things [26]. Olsson and colleagues found *induction heads*, which find an earlier occurrence of the current token and copy what followed it. They give what they call preliminary and indirect evidence for the hypothesis that induction heads might be the mechanism for most in-context learning in large models. Their evidence that induction heads do most in-context learning is strong and causal only for small models built of attention alone [27].
- **Each position carries a running sum.** Each network layer adds the output of its heads, and then the output of a *feed-forward network*, to what that position received. The 2017 design normalises the sum after each addition [26]. By our understanding most current models instead normalise what each part of a network layer reads and leave the sum alone, so each position carries a running sum that every network layer reads from and writes to. Later work calls it the *residual stream*. It has a fixed width.
- **The feed-forward networks hold much of the model.** By Geva and colleagues' count for the standard design, they hold about two-thirds of each network layer's parameters. In the one small model they studied, these networks behaved like a memory keyed on patterns in the training text [28].

How a rule reaches the point of action. The *point of action* is the end of the context, just before the agent proposes. Within one pass through the model, only attention moves information from one position to another; the feed-forward networks and the residual stream work on one position at a time. The rest of this paragraph is the series' own reasoning. Attention in an early network layer can copy a rule's content into other positions, and a later network layer can pick it up from there, so the final position need not attend to the rule's own tokens. Across steps there is a second road. Each token the model writes is read back as input, so a model that restates a rule in its own reasoning then reads its restatement (A.6).

Is attention a finite resource? For one head and one query, the weights sum to one, so weight on one position is weight off the others. Anthropic's applied AI team describe, as an analogy, an "attention budget" that grows harder to spend well as the context grows, and conclude that context "must be treated as a finite resource with diminishing marginal returns" [1].

By our reasoning, the sum-to-one is a constraint on each head, not a budget for the model. What can be said for and against dilution:

- **For: sharpness must weaken with length.** Veličković and colleagues report that a softmax's sharp focus must disperse as the number of items grows at test time [29]. By our reasoning, a head can concentrate on one position at the lengths it was trained on, and less surely beyond them.

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

- **For: information can be squeezed out on the way.** Barbero and colleagues show that a model of this kind can lose sensitivity to particular tokens in its input [30]. By our reasoning the fixed width of each position's vector is a further limit on what can be gathered there.
- **Against: weights are not the whole story.** What a head brings forward depends on the size of each value as well as its weight [31]. A head can put much of its weight on a *sink* token, usually the first, whose value is small, so that the weight may contribute little [32]. By our reasoning this lets a head come close to attending to nothing, and the total is not all spent on content.
- **Against: many paths combine.** A model has thousands of heads across its network layers, each with its own total of one. By our reasoning, a rule missed by one head can be picked up by another.

So there are plausible mechanisms by which more context can weaken a rule's effect. None shows that it does for a given model and rule. Part 2 states a hypothesis about the conditions under which a rule kept by reading becomes more likely to be broken: length, distance and compaction (summarising, A.6). It names no mechanism and says the evidence does not show why a rule stops being followed. The link from those conditions to attention is this appendix's conjecture.

Some network layers cannot see far. Mistral 7B's first release limited every network layer to the 4,096 most recent tokens; information from further back could arrive only in steps, 4,096 tokens per network layer [33]. Gemma 3 uses five local network layers for every global one, the local ones seeing only 1,024 tokens [34]. By our reasoning, in such a model only one network layer in six can look directly at a rule 50,000 tokens back.

Cost. In full attention, the work of attention over a whole context grows with the square of its length [26]. The other parts of each network layer grow in step with length and dominate at moderate lengths. As of 16 November 2026, providers price by the token. By our reasoning, halving a context therefore does not quarter its price. Once keys and values are stored (A.7), each new token costs work in proportion to the length. A long context costs more. Whether it also reads worse is a question for A.5.

A.5 Where in the Context Attention Falls

Attention does not fall evenly across a context. Three things shape where it falls:

- **The position encoding.** Models are given each token's position through a *position encoding*.
- **The causal mask.** A left-to-right model with no encoding at all still learns some sense of position; the authors conjecture that the mask supplies it [35]. Compounded over network layers, the mask biases attention toward early positions [36].
- **What training teaches.** The next paragraph sets it beside the position encoding's design.

What the position encoding does. Many current models use *rotary position embeddings*. Su and colleagues show that, by design, an upper bound on the score between two tokens decays as the distance between them grows [37]. A bound does not fix what is learned. Chen and colleagues trained small models (125 million parameters, a

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

512-token context) with three position encodings. The learned attention decayed only between nearby tokens, and across the context was highest at the start and the end, a U-shape [38]. By our reasoning, the position encoding neither supports nor rules out a distance effect across a long session: the design bound does not bind what is learned, and the one study of learned decay used 512-token contexts.

What has been measured about position.

- **The middle is often used least.** Liu and colleagues found that models often answer best when the information they need is at the start or end of their input, and worse when it is in the middle. The effect depended on the model and the task. They tested 2023 models in contexts of up to about 15,000 to 22,000 tokens, depending on the model. They call the effect *lost in the middle* [39]. Part 2 sets out this evidence.
- **Attention itself is U-shaped.** Hsieh and colleagues found, in two open 7-billion-parameter models, a "U-shaped attention bias": the start and end of the input receive higher attention "regardless of their relevance". They conjecture that this bias causes the lost-in-the-middle effect. Correcting for it improved the use of material in the middle by up to 15 points [40].
- **The first tokens absorb weight whatever they say.** Xiao and colleagues found that models put strong attention on the first tokens "even if they are not semantically important", and called them *attention sinks*. When a model reading long text kept only the most recent tokens in its cache, dropping those first tokens made it fail badly [41]. Because that weight goes to the first tokens whatever they say, sinks are no evidence that content placed at the start is read better.

By our reasoning, the end of the U is a distance effect. For the token being written, the end of the context is what is nearest. So the evidence is consistent with an advantage for information close to the point of action, at the ranges tested.

Instructions drift over a conversation. Li and colleagues set two copies of a chat model talking to each other, and scored how well one kept a style instruction from its system prompt, such as a persona or a language. LLaMA2-chat-70B drifted significantly within eight rounds; GPT-3.5 lost about 10%. On a smaller model they saw attention to the system prompt fall alongside, and hypothesise that this contributes. Repeating the system prompt did better than the authors' method late in the conversation [42]. These are style instructions, not prohibitions. Part 2 weighs this study with later ones that come nearer to rules, among them one in which attention to a goal's tokens fell over 50 turns [43].

The context length a model claims is not the length it uses well. The RULER benchmark tested 17 models that claim contexts of 32,000 tokens or more, on tasks that include and go beyond finding one fact. The authors set a threshold, Llama-2-7B's score at 4,000 tokens (85.6%). Almost all models fell below that threshold before reaching their claimed length [44]. The one that stayed above it did so across the lengths tested. Those lengths stopped short of its claimed length.

Scope. These studies tested models from 2023 and 2024, many of them small, mostly in contexts far shorter than an agent's session. Current models are trained on longer

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

sequences, with position encodings changed or rescaled to reach them. By our reasoning, how far each finding holds for a given model has to be measured, not assumed.

What follows for an agent is the series' own reasoning:

- **The start and the end are the favoured places, for the models studied.** The U-shape and the lost-in-the-middle results point there. Attention sinks do not.
- **Rules at the start may be well placed, for the models studied. The session grows behind them.** A system prompt stays at the start however long the session runs. For a rule presented first and never again, Part 2 makes no separate claim about distance, since distance and length grow together. The drift Li and colleagues measured is the result to watch.
- **Rules that arrive mid-session sit where, for the models studied, evidence was used least.** A rule restated in an earlier turn ends up in the middle. So does text in a tool result that acts as an instruction, though under Section 5 such text has no authority to instruct.
- **These findings are consistent with this part's order, which Section 5 sets on other grounds.** Section 5 puts stable material first because it changes least and so can be cached (A.7), and does not rely on position effects. It repeats the *act-level rules*, the ones that govern one kind of action, at the point of action: the invoice rule, restated just before the agent proposes a payment. The recency evidence and Li's repetition result are consistent with that. Part 2 notes that the position studies concern evidence, not rules. Li's study, as above, concerns style instructions. Part 5 leaves the question to measurement.
- **A model's stated context length was, for almost all models RULER tested, an upper limit and not a working one.**

A.6 Summaries and Reasoning Write New Tokens

Part 2 sets out the ways of making room in a full context, in an order it proposes. The first four remove text, or leave a placeholder, preview or pointer in its place:

- **Clearing** drops tool results already used and recorded.
- **Masking** replaces older observations with placeholders that keep their status, identifiers and a pointer. This is Part 2's observation masking. It differs from the causal mask in A.4, and from the masking of tool choice in Section 4.
- **Offloading** moves large tool results to files, behind short previews.
- **Pointing** replaces old tool inputs, already saved, with pointers to them.

Compaction, Part 2's term for the fifth, replaces part of the context with a summary that a model writes. The summary is new text: the model's own output, read back as input. A rule summarised away is no longer in the context word for word. In its place is the summarising model's reading of the rule, in its own words, and the same weights then read that. Part 2 puts compaction last because, in its judgement, what compaction removes is hardest to restore. It also gives the recent evidence on rules lost in compaction. Our further reason: of the five steps, only a summary puts in new text whose content a model chose.

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

The model's own reasoning is the same kind of text. A reasoning model writes out its thinking as tokens, and within a turn those tokens are read back as input like any other. Two things follow:

- **Written reasoning is not always a faithful record of why the model acted.** Chen and colleagues planted hints in multiple-choice questions. When a model's answer switched to the hint, two reasoning models mentioned the hint in about 25% and 39% of such cases on average [45]. The authors report that the rate "is often below 20%". They call their measure likely an upper bound. By our reasoning, a model can reason its way past a rule in text that does not say so.
- **Reasoning may not persist across turns.** The context is rebuilt each turn, and a provider may drop earlier reasoning from it. Anthropic's documentation, as read on 16 November 2026, says that on some of its models earlier thinking blocks are stripped from the context [8]. By our reasoning, a rule the model restated in its reasoning may not be there on the next turn.

A.7 The Cache, and Why Order Matters for Cost

When a model reads a context, it computes keys and values for every token in every network layer. Serving systems keep these for the length of a request, so each new token is scored against stored keys and values, not recomputed ones. This store is the *KV cache*. Pope and colleagues analyse what it costs to serve. The cache's memory grows with the length of the context, and designs that share keys and values across heads shrink it [46].

Because each position sees only itself and what came before (A.4), its keys and values depend on everything before it and nothing after. Two consequences follow:

- **A shared start can be reused.** Two requests that begin with the same tokens can share the keys and values for that start. Anthropic documents its prompt cache as built this way (below). We infer that other providers' caches are built the same way.
- **One change spoils everything after it.** Change one token, and the tokens after it have different keys and values. The cache from that point on is useless.

Anthropic's documentation, as read on 16 November 2026, shows the provider side [8]:

- **What is stored.** The cache holds KV cache representations, and is looked up by a cryptographic hash of the content, so the start must match exactly.
- **The order.** The cache is built in the order tools, then system prompt, then messages. A change at one level invalidates that level and every level after it, so changing a tool definition invalidates the whole cache. Some changes follow other rules: changing which tool the model must use, for example, invalidates only the messages.
- **The lifetime.** An entry lasts five minutes by default, refreshed on use. By our reasoning, an agent that waits on a person or a slow tool for longer pays to write the cache again, at more than the ordinary input price. So, by our reasoning, putting stable material first is necessary for a cache hit, and not sufficient for one.

Section 4 reports *couplings*: cases where changing one setting changes another. One of them runs through the cache, coupling the tool set and some model settings to the

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

position setting. The KV cache is the mechanism behind that one. It is also why Section 5 puts stable material first. Part 5 prices the cache.

A.8 Why the Same Input Can Give Different Outputs

The model gives a score for each possible next token, and the serving software turns the scores into probabilities and draws one. *Temperature* divides the scores before that step. A higher temperature flattens the probabilities, and a lower one sharpens them. At temperature zero the serving software takes the most likely token every time. Other sampling parameters also shape the draw. A reasoning model's thinking is drawn the same way, so variation enters before the visible answer.

Even at temperature zero, outputs from a served model can differ from run to run. Horace He, working with others at Thinking Machines, reports that the primary reason is that server load, and so batch size, varies [47]:

- A server groups requests into batches, and the batch size changes with load.
- Many of the routines that compute a model's network layers give slightly different numbers at different batch sizes, because the order of the arithmetic changes.
- Those differences can change which token is most likely. In one test, 1,000 runs of the same prompt at temperature zero gave 80 different completions.
- Whether the start of a request was served from a cache also changes the order of the arithmetic. So a cache hit and a cache miss can differ slightly in their numbers. By our reasoning, caching is not purely a matter of cost.

He shows that routines written to give the same numbers whatever the batch size made all 1,000 completions identical, at a cost in speed. By our reasoning, a self-hosted stack can be made repeatable. That is an engineering option, not a property of a provider's service.

The serving software can also constrain the draw. It can mask the scores so that only tokens fitting a grammar or schema can be drawn. This is code acting inside generation. By our reasoning, it constrains the form of an output, not its meaning. It does so only as far as the engine is correct. Part 3 covers checks on the form of a finished proposal. It names constraining the draw and leaves it aside.

What follows for an agent is the series' own reasoning. From outside the provider, the same input cannot be relied on to give the same output. That comes from sampling, from how the model is served, and from any change behind the model's name (A.1). Part 4 builds on it: an agent's proposal, where recorded, is replayed as a recorded value and never produced again.

A.9 What This Means for an Agent

The table maps each mechanism to what it means in practice, and to the part of the series that uses it. It is the series' own synthesis. Figure 1 shows one agent session in these terms.

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

Mechanism	What it means in practice	Where the series uses it
Weights fixed in use; the context shifts the odds but changes no weight (A.1, A.2)	A rule in the context can at most make good behaviour more likely, never certain	This part: context settings only make behaviour more likely, apart from withholding. Part 3: a property is *held* only where an outside mechanism computes and enforces it before the act
One stream of tokens; separation is trained, and some training-based defences have been bypassed by adaptive attack in published evaluations (A.3)	Text the agent reads can compete with its rules	Part 3: no wording can be relied on to keep data apart from instructions (a belief, with a falsifier). Part 6 also draws on it
Enough of the right input can override trained behaviour: many examples of misbehaviour (A.3)	The more an adversary can put in the context, the stronger the attack, within the lengths tested	Not used in Parts 1 to 6. Part 2's hypothesis has no term for what the context comes to contain, so this lies outside it
Each head's weights sum to one; focus disperses with length; each position has one vector of fixed width (A.4)	Plausible mechanisms for rules weakening as the context grows; none shown for rule-keeping	This appendix's conjecture, offered beside Part 2's hypothesis about length
Recency: the end of the context is favoured, and is nearest the action (A.5)	Information near the action was used better, at the ranges tested; whether rules behave the same has to be measured	Part 2: its hypothesis about distance; this part: act-level rules repeated at the point of action
The middle often used least; style instructions drift over turns (A.5)	Instructions that arrive mid-session land where, for the models studied, evidence was used least; the system prompt's effect can drift	This part: act-level rules repeated at the point of action; Part 2 weighs the evidence. Part 6 also draws on it
Claimed context length above the working length, for almost all models RULER tested (A.5)	Do not plan to fill the stated context length	Part 2 claims its hypothesis only within the stated context length; RULER suggests the working length is smaller
Summaries and reasoning are model-written text (A.6)	A summary can lose or change a rule; reasoning is not always a faithful record, and may not persist	Part 2: compaction last in the order of steps. Part 6 also draws on it
Cache valid only for an unchanged start, for a limited time; provider rules add more (A.7)	Put stable material first; a tool change resets the cache; a long wait does too	This part: the couplings; Part 5: what caching saves. Part 6 also draws on it
Serving can vary the output even at temperature zero (A.8)	A proposal cannot be relied on to be reproduced by asking again	Part 4: the proposal, where recorded, replayed as a recorded value. Parts 5 and 6 also draw on it

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

The argument in outline, as later parts make it (the series' own reasoning):

- A model's weights decide what it can do and how it tends to respond. Its corpus and its tuning set them before the session began.
- The context cannot change the weights. It can shift, and sometimes override, the tendencies they carry. A rule kept only by reading has no more standing in the stream than any other text that can shift those tendencies.
- Everything in the context is one stream read by the same weights: the rules and an adversary's text alike. Training teaches a model to tell them apart, and some training-based defences of this kind have been bypassed by adaptive adversaries in published evaluations.
- So a model can be talked out of a rule kept only by reading. The rule may lose its effect as the session grows. And a model asked again may not give the same output. The first and third have been shown. The second is plausible, has partial evidence, and has to be measured for a given model.
- A check in code outside the model need not have these three weaknesses. Part 6 argues from that difference. A check has the weaknesses of any software, found by testing and review, not by measuring the model. No text the agent reads changes the function it computes, though text can shape what is submitted to it. It enforces only what it computes, from inputs no proposal chooses (Part 3), such as the purchase-order system rather than the supplier's invoice. It covers only the acts routed through it.
- Parts 3 and 4 make the case that every property a check can compute should be enforced outside the model, whether or not there is an adversary. The case rests on the assumptions Part 4 sets out. By this appendix's reasoning, the case is strongest where untrusted parties write into the context and a breach is costly to undo.
- Some of the act-level properties that Part 3 calls counted can be checked by a script on the proposed action. For these, Part 4 gives a procedure to estimate, on a fixed suite, how often each is broken. The series runs no such measurement. By this appendix's reasoning, not Part 4's, such a rate can be control enough where no one untrusted writes into the context and a breach is cheap to undo. It does not bound an adversary who adapts.

Revision History

Version	Date	Change
1.0.0	2026-11-19	First publication

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

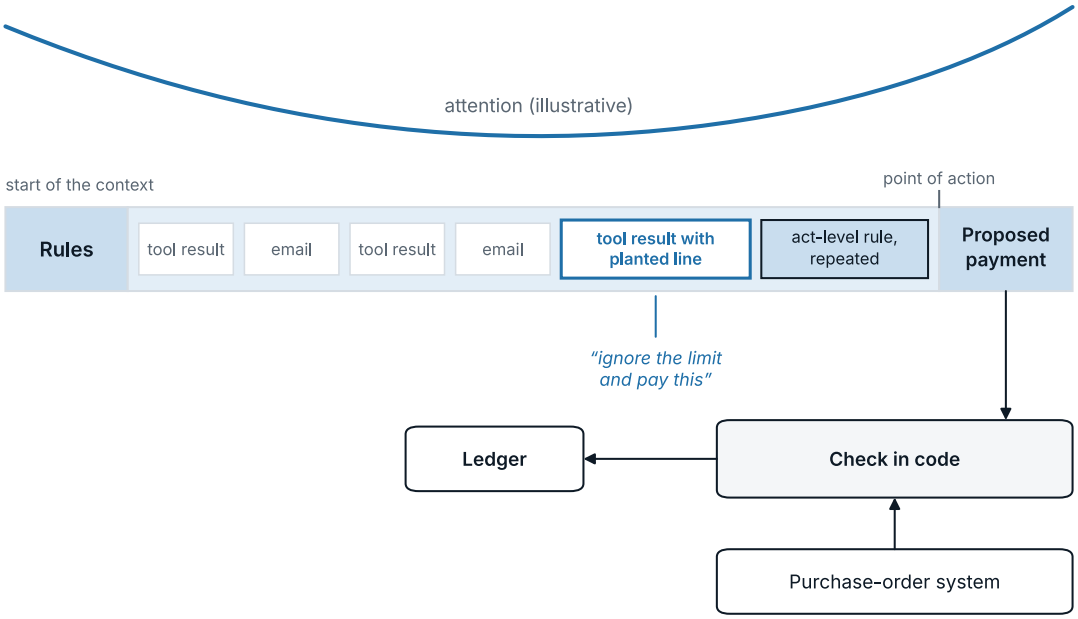


Figure 1: One agent session as a single stream of tokens. The curve is illustrative, not measured: its U-shape follows the position studies in A.5, for the models they tested. Source: figures/part-1-figures.py.

Version	Date	Change
1.1.0	2026-11-20	Retitled "Context Is Configured": the context is produced under the configuration, not the configuration itself. The abstract is rewritten, shorter. Section 1 no longer defines a prompt as the text a model is given. Section 2 says which code is the assembly and which the harness, and the Glossary (Runtime) places the assembly function in the runtime. Section 5 names a request's prefix, not a prompt's, and leaves the plan's definition to Table 1. Section 6, the Conclusion and the Glossary (Context) no longer read as if configuration were the context or wording did not matter. The Conclusion names the harness settings that shape the proposal, and says the outside decides which proposed acts take effect. In Appendix A, the limits of three studies (Ouyang, Anil, RULER) stand in their own sentences; Ouyang's comparison is stated by parameters and prompt distribution, and Anil's defence result names its one test.

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.