// ARTICLE · PUBLISHED 30 OCTOBER 2026

Context Windows and Code Graphs: The Error Past the Cut

What happens when a reviewing model is handed more than it can hold, and what a deterministic graph of the code does about it

Jason Newell · MAX Research Collective · Article ART-AGT-032 · Published 30 October 2026
Licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0), https://creativecommons.org/licenses/by/4.0/
Original: https://www.maxresearchcollective.com/articles/context-windows-and-code-graphs.html
Cite as: Newell, J. (2026). Context Windows and Code Graphs: The Error Past the Cut. MAX Research Collective, ART-AGT-032. https://www.maxresearchcollective.com/articles/context-windows-and-code-graphs.html

A defect at its own boundary

The local reviewer reported a syntax error. It named a line, described an unclosed construct, and sounded entirely confident. Nothing was wrong at that line. The line was the last one the model had been given, and the fault it found was the edge of its own input.

That's a truncation phantom, and there's a mechanism behind it rather than a mystery. A diff — the set of lines a change adds or removes — was capped at 60,000 characters before it went into the prompt. Past the cap the text stopped mid-construct. A model can't see past a cut. What it can see is a fragment that looks broken, so it reports the fragment, and the check went red on a defect that existed only inside the prompt.

Two more failures were running at the same time, and all three reached an operator as the same complaint: too big. All three trace to a window nobody chose and a prompt nobody sliced, and a deterministic graph of the code fixes the half that a bigger window can't.

#	FAILURE	MECHANISM
1	Truncation phantoms	The model cannot see past the cut, so it reports the cut itself as a defect: a syntax error at its own boundary
2	No code around the change	A diff alone carries no callers, no callees, no tests. A small model then scores well by paraphrasing the diff's own comments, and four findings came back that were each a comment explaining the bug they had found
3	An ungoverned window	No options were sent at all, so the model ran at its native 256,000-token window and paid a memory allocation sized for it while reading a prompt of about 20,000

Table 1. Three failures that all surface as the same operator complaint.

Failure 2 produced the most plausible output of the three, which is why it deserves a second look. A reviewer handed a diff and nothing else sees the change and nothing the change touches. Nobody calls into it. Nothing calls out of it. No test is visible anywhere. So the model does what a student does with an unfamiliar passage: it paraphrases. Four confident findings came back, and every one restated a comment sitting in the diff — including the comment that described the bug.

Nobody chose the window

Failure 3 is the counter-intuitive one.

The window was never too small; nobody picked it. A context window is the number of tokens a model can attend to in one request (prompt and answer together), and a token runs roughly three quarters of an English word. Left unspecified, the server reached for the model's native ceiling of 256,000.

What that costs stays hidden until you know where the memory goes. A transformer keeps a key-value cache for every token in its window, and the allocation follows the window size rather than how much of it you fill. So a prompt of about 20,000 tokens was paying for 256,000. The tax landed on every review. It showed up as latency, and the timeouts it fed are what eventually got the lane disarmed altogether.



Figure 1. What the window was allocating against what the prompt actually used, before and after the window became a stated number.

An ungoverned setting fails in both directions. Nothing announced the 256,000. Had a server configuration lowered the default instead, the prompt would have been truncated in silence and the check would have gone green. A number nobody states is a number nobody can check.

Two halves of a fix

Govern the window. The request now carries an explicit 65,536, with the arithmetic written down beside it: roughly 10,000 tokens of system text and repository brief, leaving roughly 55,000 for the diff and the answer. That number prints in the header of every review report, so it's a visible lever rather than a server-side mystery. On an older client that rejects the option, it's the first rung shed in the existing degradation ladder — and the review still runs.

Stop sending one big document. This is the half that needed a code graph, and it's the half that changed what the reviewer could actually say.

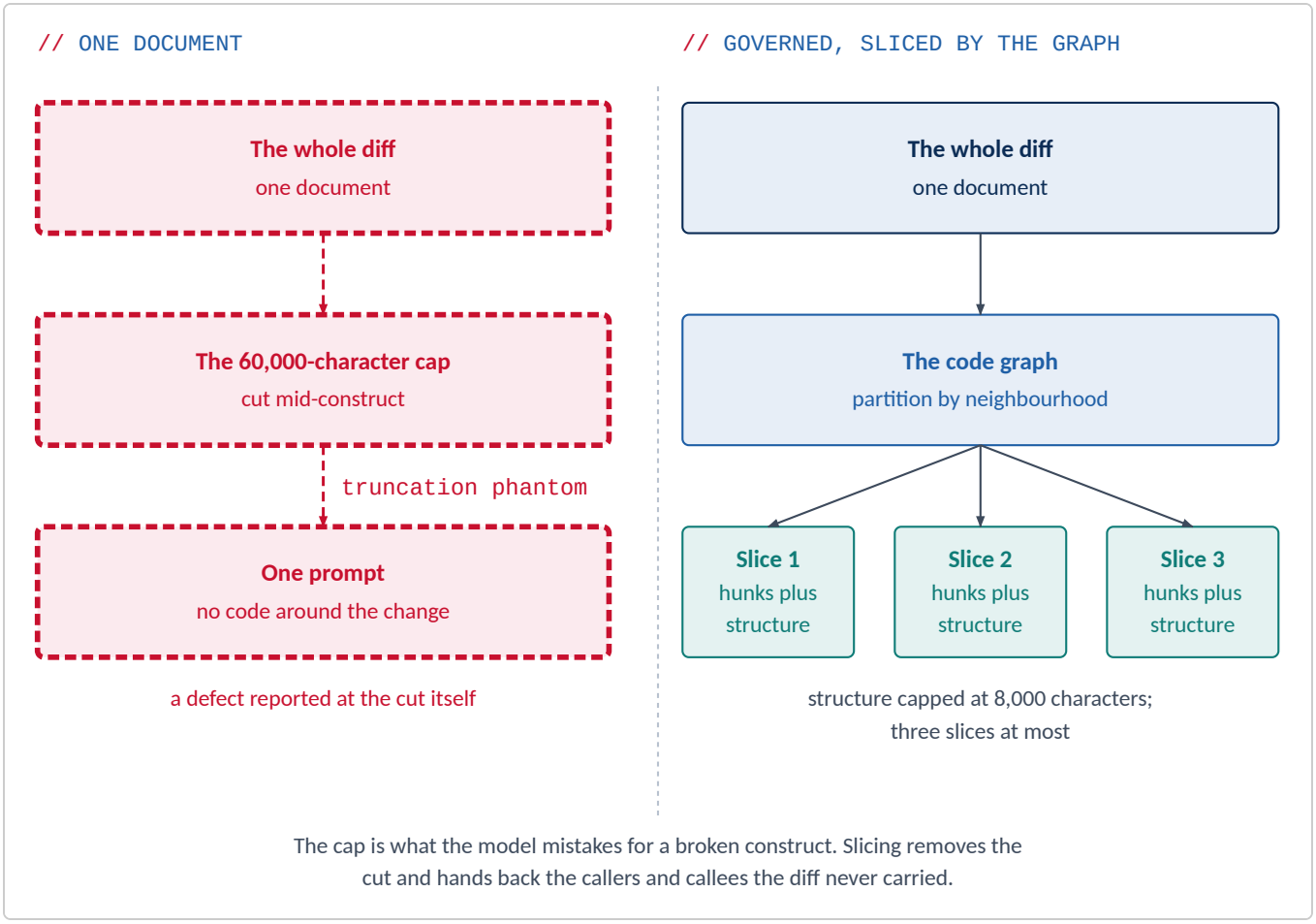


Figure 2. One monolithic prompt against a governed, graph-sliced one.

A code graph is the output of static analysis: every symbol in the tree as a node, and every call, import, inheritance and reference as a typed edge between them. The library that builds it is pure Python over an incremental parser. No model sits in the path; nothing touches a network; the same commit in gives the same graph out. Determinism is what makes everything downstream checkable instead of probabilistic.

With a graph in hand the diff stops being one document. It partitions into at most three slices, grouped by which neighbourhood of the code each hunk touches. Every slice carries its own hunks plus a structural block: the enclosing symbols, one hop of callers, one hop of callees, each edge marked as read from the syntax tree or

inferred by the producer. That block is hard-capped at 8,000 characters — structure can never crowd out the diff it competes with.

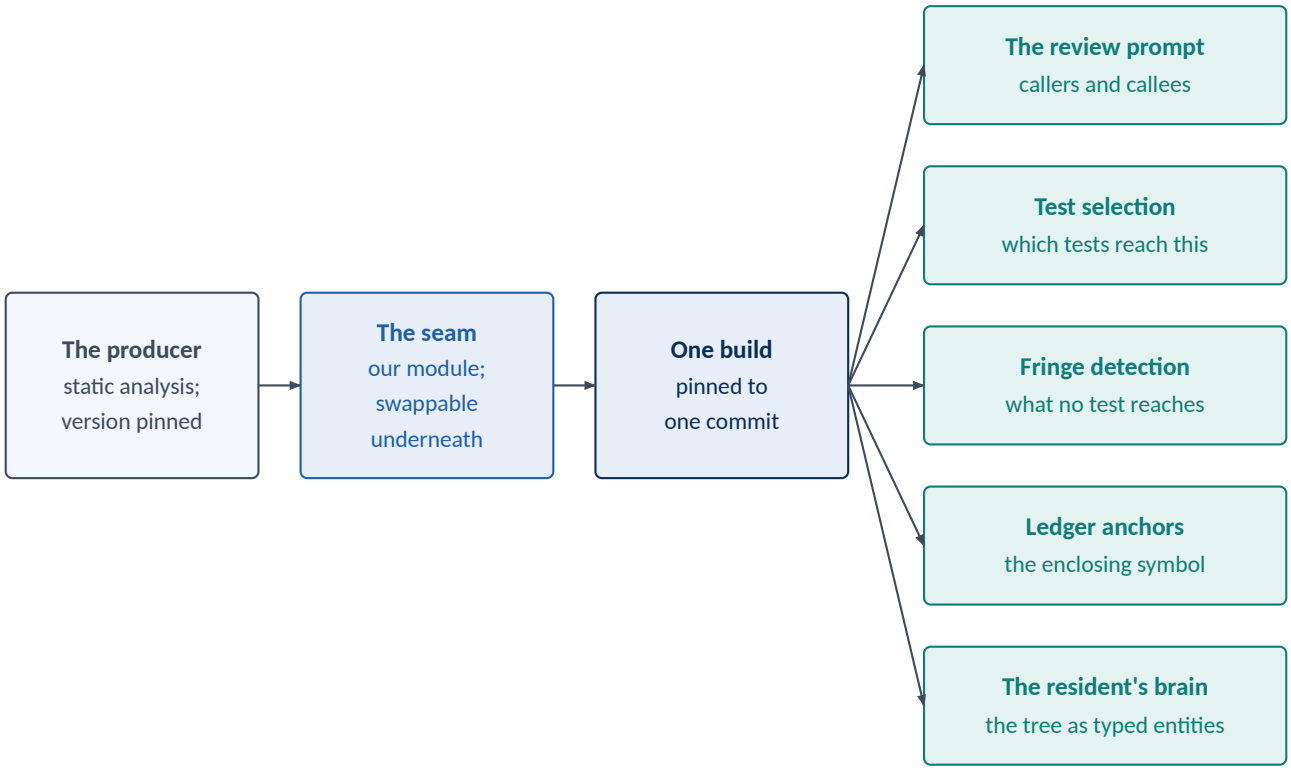
Four things follow mechanically, and only the last of them is about quality. No slice reaches the 60,000-character cap, so the phantom has nowhere left to form. Smaller prompts generate faster. A deadline now costs one slice rather than the whole review, because the slices are independent and a partial answer is still an answer. The model also sees the code around the change at last.

One graph, five consumers

Building the graph is the expensive part — the one that takes real seconds. Once it exists, everything else is a query.

CONSUMER	WHAT THE QUERY ANSWERS	EFFECT
The review prompt	What calls this, and what does it call	Closes both the phantom and the paraphrase
Test selection	Which tests can reach these changes	A change stops paying for the whole test pyramid
Fringe detection	Which changed symbols no test reaches	Runs before the model, and a graph query cannot hallucinate
Ledger anchors	The same enclosing symbol, in place of line proximity	Retired a class where a deleted file's hunks confirmed an unrelated file's imports
The resident's knowledge	The tree as typed entities	The system holds its own codebase the way it holds any other subject

Table 2. Five consumers off one build, and what each one asks the graph.



Building the graph costs seconds. Every consumer is a query against the build, and none of them imports the producer directly.

Figure 3. One producer behind a seam of our own, and the five consumers that query one build.

The fourth row is the quiet one. A ledger of past findings has to decide whether a new finding matches an old one, and line proximity is the obvious way to do it. It's also wrong the moment a file moves. Anchoring on the enclosing symbol retired a whole class of false matches, including one where a deleted file's hunks were confirming an unrelated file's imports.

Freshness is commit-pinning rather than a schedule. The graph is a derived artifact of exactly one commit, and it carries that commit wherever it travels; every consumer asks for the one it is working on. A mismatch degrades in the open. The reviewer notes that structural context is stale and leaves it out, and test selection falls back to the full suite. New tests arrive as new nodes in the very commit that adds them, so there's no separate index to fall behind — and that missing index is what makes development speed the fuel here rather than the threat.

Why a deterministic tool

”Which tests reach this symbol” is a graph walk. Asking a 27-billion-parameter model to guess it is how you get a confident wrong answer; asking a parser gets the same answer twice.

The measurements come off this tree rather than off a vendor page. A cold build of the full backend takes 73.4 seconds across 1,337 files; a warm one takes 20.2, a little under a third of that. The result holds 31,807 nodes and 65,156 edges. Yet the decisive probe was narrower than any of those numbers: one module, asked for the tests that reach it, returned exactly its four test files.

Two design choices sit behind that. The producer lives behind a seam of our own, since the library is pre-1.0 and moving fast; the version is pinned and no consumer imports it directly, because swapping the producer must not touch a consumer. Over-selection is the second. Where the graph widens through a shared fixture it hands back too many tests, never too few. A suite that runs long is a cost; a suite that misses the changed code is a hole.

What it doesn't solve

LIMIT	THE HONEST STATEMENT
Precision near hubs	One module selects 20 test files, because it widens through a shared fixture. Safe, and not precise.
Dynamic dispatch is invisible	Static analysis cannot see a call made through a string or bound late. Hence the split between edges read from the syntax tree and edges the producer inferred, labelled as such wherever they reach a prompt.
Configuration is outside the graph	The build configuration has no extractor, so it is absent entirely and reaches the prompt through a separate brief.
The window is still finite	65,536 is a choice rather than an escape from bounded input. A large enough slice still truncates.
Slices share one clock	The overshoot budget is charged once globally rather than per slice, so one slow slice exhausts the clock and the later ones are skipped rather than silently dropped.
Test selection is not gating yet	It ships observe-only, printing what it would have selected. The flip waits on measurements, under a measure-before-you-flip discipline.
The producer is pre-1.0	One file produced zero nodes and was named in a warning. Honest absence, and still absence.

Table 3. What the graph does not fix, stated before anybody has to discover it.

The fence has to be in the code

The review prompt carries the repository's full diff. That single fact makes the choice of model host an egress decision rather than a configuration preference, because one mistyped variable would ship every diff to a third party, quietly, behind a green check.

So the reviewer refuses any host that isn't loopback — an address that can't leave the machine. The refusal lives in code rather than in a convention somebody documented, and it sits on the probe path and the chat path both, since the check that matters is the one on the path the data actually takes. Hosts are parsed rather than substring-matched. An address like http://127.0.0.1.evil.com/ contains the loopback address and is entirely remote.

Two more choices follow from treating the lane as a guard rather than as an oracle. A field recording whether the review ran is load-bearing, since a dog that is asleep must never be mistaken for a dog that saw nothing, and an exit code of zero has to mean the review happened. And the lane stays inert until somebody arms it. With no local model available it would otherwise report that it hadn't reviewed, forever, and a permanently red check is one people learn to scroll past. Absent is more honest than red.

The cost ladder, as the code states it

The roster of lanes is a typed table in the reviewer's own source, and the tier field encodes cost-of-catch directly.

TIER	LANE	KIND	METERED IN
1	The interactive session	Reviewer	The turn itself, before any push
2	The local reviewer	Reviewer	Nothing, free at the margin and never throttled
3	Three hosted services	Reviewer	Vendor quota, per push
4	Continuous integration	Gate	Build minutes, the most expensive place to learn something

Table 4. The lanes in run order. A reviewer produces findings; a gate produces only pass or fail.

Every defect caught at tier 1 or tier 2 is a tier-3 request never made, and a tier-4 minute never burned. Tier 2 costs nothing to run, so it can afford several focused passes where a metered lane runs once and gets rationed. The per-slice fan-out is exactly what that freedom buys.

The argument for keeping a lane on the box rests on the lane being there, whatever the local model's quality. On one evening in August all three hosted services went unavailable at once, for three unrelated reasons, on a day with six pushes; later that evening the build service stopped creating runs at all. Underneath the availability floor sits a compatibility one. A vendor ships its reviewer as a container action, and the build service runs those on Linux only, so on a Windows workstation that lane can't run whatever the configuration says.

What a fleet does to all of this

The same discipline runs across several repositories, and exactly one artifact is byte-identical between them: a shared core block inside the standards file, authored in one repository and carried verbatim by every sibling. Each downstream repository records the block's digest and fails on a local edit.

That proves there was no local edit, and it proves nothing else. The copies may still have parted company, because the block and its digest live together and a sync updates both. Only the authoring repository can see more than one tree at once, so its comparison gives an explicit three-way verdict in which "cannot compare" reads as unknown rather than as in sync.

That distinction stopped being theoretical one morning in August. A sibling sat 155 lines behind, with every check inside it green. Not one local edit had been made. Upstream had added 155 lines of rules and nobody had told the sibling, and what found it in the end was somebody cloning the two trees and diffing them by hand.

Why it matters

A reviewer reading past its own input, a window nobody chose and a copy drifting behind a green board all fail the same way: quietly, with a confident result. The cost

lands on whoever trusts the check, and the fix each time was a number or a fence written down where a person can see it.

A false clean is worse than no check at all. No check is a known gap, and somebody eventually closes a known gap. A false clean is the state that lets drift accumulate behind a green board for exactly as long as nobody happens to look.

Glossary

TERM	DEFINITION
Commit-pinning	Stamping a derived artifact with the commit it was built from, and refusing to use it against any other. The alternative, a refresh schedule, can serve stale data with nothing showing.
Context window	The maximum number of tokens a model can attend to in one request, prompt and answer together.
Diff	The set of lines a change adds, removes or alters. A hunk is one contiguous changed region with its surrounding lines.
Egress	Data leaving the machine. The review prompt carries the full diff, which makes the choice of model host an egress decision.
Extracted and inferred	A producer's confidence on an edge. Extracted was read directly from the syntax tree; inferred is the producer's guess, covering dynamic dispatch and indirect calls, and it's labelled wherever it reaches a prompt.
Fringe	The boundary of a change's blast radius where no test reaches the changed symbol. A graph query finds it before the model runs.
Gate	A check returning pass or fail over a whole change. A reviewer returns findings and can therefore teach; a gate can't.
Honest degrade	Failing to a named and visibly reduced state rather than to a plausible wrong answer. Noting that structural context is stale and omitting it is a degrade; reading a stale graph quietly isn't.
Key-value cache	The tensors a transformer holds for every token in its window. The memory is allocated from the window size rather than from how much of it you use, which is why an ungoverned window taxes a small prompt.
Lane	One reviewer or gate that can speak on a change, carrying a tier that encodes what a run costs.

TERM	DEFINITION
Ledger	The on-box store of every finding every lane has made against a repository, with whether each one held up.
Loopback	An address that can't leave the machine. Any other host for the model is remote, however it is spelled.
Seam	A module of our own wrapping a third-party producer, so the producer can be swapped without touching a consumer.
Self-hosted runner	A build agent on your own hardware rather than the vendor's. It's what lets a build job reach a local model at all.
Slice	One partition of a diff, grouped by the part of the code graph it touches, reviewed as its own small prompt.
Static analysis	Reading source code without running it. Here it produces a typed graph of symbols and the edges between them.
Token	The unit a model reads, roughly three quarters of an English word. Code runs denser than prose.
Truncation phantom	A defect a model reports at the boundary of its own input, where the text was cut mid-construct and looks broken.

AGENTIC AI ART-AGT-032

// ABOUT THE AUTHOR

Jason Newell leads organizations and people in technology, and is a researcher, deep technologist and builder — from a CRAY to agentic AI. He founded MAX Research Collective to build AI for regulated industries that runs where its owner has control, with healthcare as its strongest focus. About Jason → Connect on LinkedIn →

Newell, J. (2026). MAX Research Collective, ART-AGT-032. CC BY 4.0