The Context Graph Protocol: Typed, Budgeted, Provenance-Carrying Context Retrieval for LLM Agents

Mac Anderson[†] and the Context Graph Protocol contributors

[†]Oxagen · mac@oxagen.sh · Specification and reference implementation: github.com/macanderson/context-graph-protocol

ABSTRACT

Large-language-model agents assemble their working context from retrieval pipelines that return opaque strings: nothing states where a passage came from, what it costs against the context window, whether it is still true, or how to cite it. We present the Context Graph Protocol (CGP), an open wire protocol that replaces the blob with a typed unit of exchange — the *frame* — carrying kind, relevance, canonical token cost, provenance, temporal validity, and a citation label. Frames relate to one another through labelled edges, making context a graph a host can traverse by anchor proximity. The protocol defines an eight-envelope NDJSON binding over stdio and streamable HTTP, an explicit capability handshake, a deterministic budget-accounting rule that renders budget honesty machine-checkable, a consent model for data egress, and three content representations that let large context travel by hash without sacrificing verifiability. Conformance is falsifiable by construction: a seven-check suite is validated against fourteen single-fault misbehaviour modes, and four independent implementations — Rust, TypeScript, Python, and Go — pass the same oracle. We argue that deterministic, canonical-order composition converts provider prompt caches from an accident into a contract, with a worked example showing a ~7× reduction in context tokens over a twenty-turn session.

Keywords: context engineering · LLM agents · retrieval protocols · provenance · token budgets · conformance testing · prompt caching

1 Introduction: the blob-pipe problem

An agent's quality is bounded by the quality of its context, and context windows degrade as they fill. Liu et al. showed that models reliably use information at the edges of long contexts while missing it in the middle [1]; Hong et al. measured systematic performance decay — "context rot" — as input length grows even on trivially simple tasks [2]. The engineering response has been retrieval: fetch less, fetch better. But the interfaces through which agents receive retrieved context have not kept pace with what retrieval now feeds them.

Today's dominant interface is what we call the *blob-pipe*: a pipeline concatenates whatever it found into a string and hands it to the model. The blob answers none of the questions an auditor — human or machine — would ask. Where did this passage come from? What does it cost against the window? Was it true at the time the question concerns? Who consented to its leaving the machine? How should the model cite it? When an agent misbehaves, the blob leaves nothing to inspect; Kapoor et al. argue that exactly this kind of unaccountable scaffolding is why agent benchmark results routinely overstate real-world reliability [5].

The Context Graph Protocol (CGP) is a wire protocol for the retrieval interface itself. Its central commitment: **the unit of exchange is a frame, never a blob**. A frame states what it is, where it came from, what it costs, when it was true, and how to cite it. A host asks providers for frames relevant to a goal, under a token budget; providers return frames carrying their own origin, their honest cost, and a human-readable citation label; the host composes them into a prompt it can trust as *evidence*, not accept on faith.

CGP is deliberately narrow. It does not specify tool invocation — that is the Model Context Protocol's scope [7], and CGP will not absorb it. MCP connects tools; CGP connects context. An agent needing both composes them: CGP frames feed the prompt, MCP tools do the work.

2 Design principles

- **Evidence, not instruction.** Frame content is untrusted data. A conforming host delimits it as quoted material, never as instructions — the principle that separates an email body from its headers, made part of the protocol rather than left to host discretion.
- **Every claim checkable.** A guarantee whose violation is indistinguishable from legitimate behaviour cannot be checked, so capabilities are negotiated explicitly at handshake, never inferred by observation, and every normative rule is paired where possible with a machine check.
- **Honest gaps, declared.** The specification enumerates the rules its conformance suite cannot check (host-side consent gating, content-byte matching beyond digest grammar). A suite that quietly omitted them would be the self-attestation the project rejects.
- **Additive bias.** A new optional field is a minor change; a removed or renamed field requires a new major family. Normative wire shapes are separated from informative reference-host behaviour.

3 The frame model

A frame is a structured record of relevance, cost, provenance, and validity. Seven kinds cover the shapes agent context takes in practice: `snippet`, `symbol`, `fact`, `doc`, `memory`, `episode`, and `graph`. Five fields are required on every frame — `id`, `kind`, `title`, `score` (relevance in [0,1]), and `token_cost` — with content, provenance, temporal bounds, citation, embedding fingerprint, and relations layered on optionally.

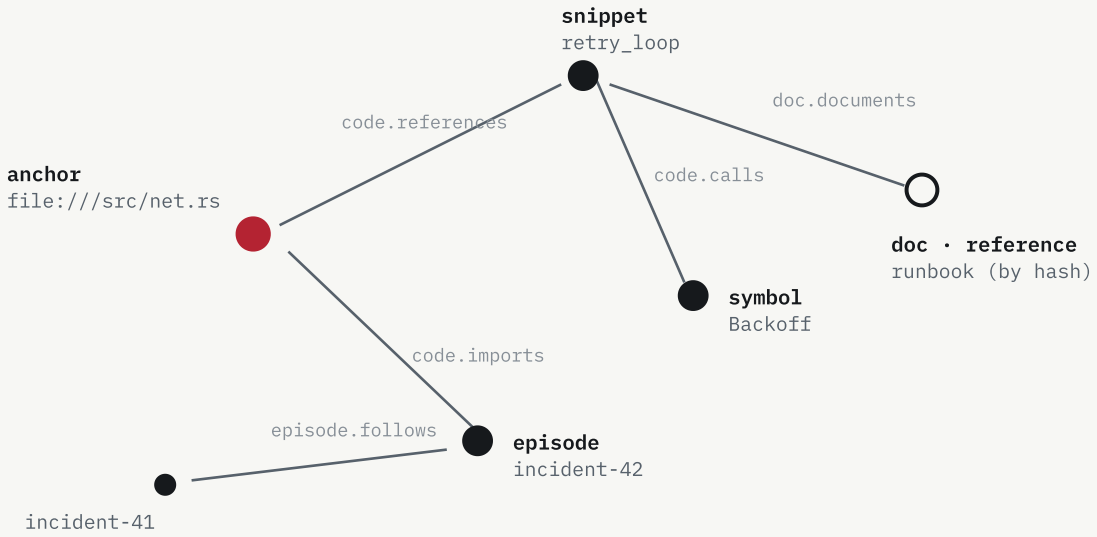

Figure 1 — A context graph. A graph frame is a node with its labelled edges; the host names anchors, and graph-capable providers boost frames within a few relation hops. The open node is a *reference* frame: no inline content, still verifiable by hash.

3.1 The graph is carried in relations

CGP is named for the graph, and the graph is carried in `relations`: a graph frame is a node with its labelled edges, not an ad-hoc serialization format. Each relation is a directional edge `{ rel, target_uri, display_name? }`. Six names are published — `code.calls`, `code.imports`, `code.defines`, `code.references`, `doc.documents`, `episode.follows` — and the vocabulary is open: providers namespace their own, and hosts must not reject unknown values. Queries meet the graph through *anchors*, focal URIs such as open files or mentioned symbols; graph-capable providers should boost frames within a small number of relation hops of an anchor. This gives CGP a portable analogue of structure-aware retrieval results from GraphRAG [3] and repository maps [4] without prescribing any particular index.

3.2 Content-optional representations

A frame carries content in one of three representations. `full` inlines the content (and, for wire compatibility, omits the `representation` field). `compact` inlines a transformed rendering — say, a signature-only view — alongside the canonical hash of its source and a `transform` identity. `reference` carries no inline content at all: an opaque `content_ref` resolver handle plus a canonical hash; it is never encoded as an empty string. Fidelity is explicit (`exact | normalized | summarized | omitted`), and hosts negotiate via ordered `representation_preferences`. The consequence: large context can travel by name and still be verified byte-for-byte, and a host that already holds the bytes pays near-zero tokens to re-establish trust in them.

3.3 Identity and time

The stable identity of a frame's exact bytes is the triple `(provider_id, frame_id, content_digest)` — the spine shared by deterministic composition, usage reports, and verification. A frame may omit its digest, but is then unverifiable and must be re-queried rather than reused. Frames are bi-temporal (`valid_from`/`valid_to` bound truth; `recorded_at` marks capture), and a query's `as_of` pins retrieval to an instant — the property that lets an agent ask what was true *then*, a need MemGPT-style memory systems surface constantly [6].

4 The wire protocol

The semantic layer — frames, queries, capabilities — is defined independently of transport. The current binding is NDJSON: every message is a single JSON envelope internally tagged by `type`; over stdio, exactly one envelope per line; over streamable HTTP, one envelope per request and response. CGP is **not** JSON-RPC — there is no `jsonrpc` member and no `method`/`params` split; the lifecycle is informed by MCP, but the framing is its own.

TYPE	DIRECTION	PAYLOAD
`handshake`	host → provider	protocol_version
`handshake_ack`	provider → host	protocol_version, provider, capabilities
`query`	host → provider	query (ContextQuery), optional id
`frames`	provider → host	result (ContextQueryResult), optional id
`verify`	host → provider	request (VerifyRequest) — capability-gated
`verified`	provider → host	response (VerifyResponse)
`shutdown`	host → provider	—
`error`	provider → host	message, optional id, optional code

Table 1 — The eight envelope types of `contextgraph/1.0-draft`.

The host opens with `handshake`; the provider replies with its version, identity, and capabilities; no query payload moves before the exchange completes. Versioning is by major family — `contextgraph/1.0-draft` and `contextgraph/1.0` interoperate; `contextgraph/2.0` does not — which lets the draft suffix drop at freeze without a flag day. Requests may carry an opaque correlation `id` that capability-declaring providers must echo verbatim; an envelope without an id is a notification, the shape a future push extension needs. Six error codes form an open vocabulary (`bad_request`, `unsupported_kind`, `budget_unsatisfiable`, `unavailable`, `shutting_down`, `internal`); unknown codes degrade to `internal`. Robustness is contractual: a provider must survive a malformed line and tear down cleanly on `shutdown`.

5 Budget honesty

The context window is the scarcest resource an agent manages, so CGP makes cost accounting a first-class, checkable protocol obligation rather than a courtesy. A *budget token* is an accounting unit, not a tokenizer:

```
token_cost = ceil( utf8_byte_length(content) / 4 )        (B3 — exact, no tolerance band)
```

Because the rule is deterministic in every language, honesty is machine-checkable: summed `token_cost` must not exceed the query's `max_tokens` (B1); frame count must not exceed `max_frames` (B4); and a host must drop, with a loud report, the frames of any provider that violates the budget (B2). The reference host classifies such providers as budget liars during fan-out and isolates them from well-behaved peers.

A host must not treat one budget token as one model token. Four bytes per token tracks English prose; dense source code runs ~3–3.5 bytes per model token and CJK text ~3, so hosts map real budgets through a safety factor (the reference host suggests 1.35: 10,000 model tokens of headroom becomes a 7,407 budget-token query). The count covers `content` only — never titles, citation labels, or host chrome. For representation-bearing frames, `token_cost` prices the inline rendering while `canonical_token_cost` (with a named `tokenizer_ref`) prices the canonical source.

6 Consent and data flow

Every provider declares its data flow at handshake: `{ reads, writes, egress, egress_scopes[] }`, with four base scope classes (`local-only`, `org-tenant`, `third-party-index`, `third-party-model`) plus namespaced custom scopes. The host-side obligations are strict: never auto-enable an egress provider; never transmit a query payload before consent is recorded; and treat every non-loopback HTTP provider as egress regardless of what it claims. Scope truthfulness is conformance-checked — a provider that declares `local-only` while flagging egress is caught by the suite, and consent records retain the human-readable scope that was actually granted.

7 Deterministic composition and context reuse

Retrieval feeds a prompt, and modern inference pricing makes the prompt's *stability* an economic property: providers discount cached prefix reads by an order of magnitude. CGP's reference host therefore composes frames in canonical `FrameId` order — independent of arrival order, ignoring score and cost ties — inside explicit `<frame>` fences as quoted material. Identical evidence yields byte-identical context blocks, turning the prompt cache from an accident into a contract.

The worked example in the specification's context-reuse note: a twenty-turn session composing eight frames (~4,000 tokens) per turn pays ~80,000 context tokens through a blob-pipe that reorders freely. Under canonical-order composition with cache reads priced at 0.1×, the same session pays ~11,600 effective tokens — a ~7× reduction on the context portion — with no change to what the model sees semantically.

Verification closes the loop: a host re-establishes trust in cached frames via `verify`, whose verdicts (`valid`, `stale`, `gone`, `unknown`) partition reuse candidates totally, defaulting to deny.

8 Conformance as falsification

"CGP conformant" means green on the conformance suite for the declared capability set — a checkable claim, not a self-attestation. Seven checks cover the handshake, consent-scope truthfulness, frame validity, verify honesty, budget honesty, clean shutdown, and malformed-input tolerance; checks skip, visibly, when a capability is undeclared.

The suite must also demonstrate that it can catch a cheat. The bundled reference provider ships fourteen `--misbehave` modes, each engineered to break exactly one guarantee — lying about costs, under-reporting them, flooding past `max_frames`, malforming digests and timestamps, dropping correlation ids, rubber-stamping or hollowing verification, lying about egress scope, crashing on garbage — and continuous integration asserts every mode is caught by at least one check. The mode list is discovered from the binary's own help output, so adding a misbehaviour without a catching check turns CI red. A suite that only ever passes proves nothing.

Cross-language byte-exactness rests on golden fixtures: five JSON files plus a manifest (fixture profile 1.1.0), each carrying a source object, its digest-profile normalization, the exact RFC 8785 canonical text [8], and the SHA-256 of those bytes; the manifest pins versions and digests every file, so fixtures cannot drift silently. Four independent implementations — the Rust reference and the TypeScript, Python, and Go SDKs — pass the same oracle, each pointed at the identical Rust suite through a language-neutral runner. That shared oracle is the point: every additional green implementation is independent evidence that the specification, not any one codebase, defines the protocol.

9 Related work

MCP [7] standardizes how agents invoke tools and receive resources; CGP standardizes how retrieved context is typed, budgeted, and accounted. The protocols are complementary by construction: MCP has no budget-honesty contract, egress consent gate, provenance chain, or conformance suite because those are outside its scope, not deficiencies in it. **Long-context degradation** — positional loss [1] and context rot [2] — motivates budgeted, selective retrieval rather than window-stuffing. **Structure-aware retrieval** — GraphRAG's entity graphs [3] and Aider's tree-sitter repository maps [4] — demonstrates the value of the graph shape CGP makes portable. **MemGPT** [6] treats context as a managed memory hierarchy, exactly the kind of host that benefits from frames with honest costs and temporal bounds. **Kapoor et al.** [5] call for agent evaluations that account for cost and reproducibility; CGP's usage reports and deterministic composition are protocol-level answers to that call.

10 Status and governance

The protocol is published as `contextgraph/1.0-draft` with the specification, JSON Schema (Draft 2020-12), three Rust crates (`contextgraph-types`, `contextgraph-host`, `contextgraph-conformance` with the `contextgraph-inspect` prober), three SDKs, wire-transcript examples, and roughly thirty-seven stable requirement anchors (H1–H4, C1–C6, B1–B4, F1–F5, D1–D4, V1–V4, …) that conformance evidence cites. Normative changes require an issue, a specification and changelog update, and a *witness*: a conformance check or wire example proving the change is real and enforced. The freeze criterion for dropping the draft suffix includes at least two independent implementations passing conformance — a bar the TypeScript, Python, and Go SDKs already clear. Specification and code are dual-licensed MIT OR Apache-2.0.

References

1. N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, P. Liang. *Lost in the Middle: How Language Models Use Long Contexts.* TACL 2024. arXiv:2307.03172.
2. K. Hong, A. Troynikov, J. Huber. *Context Rot: How Increasing Input Tokens Impacts LLM Performance.* Chroma Research, 2025.
3. D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, J. Larson. *From Local to Global: A Graph RAG Approach to Query-Focused Summarization.* Microsoft Research, 2024. arXiv:2404.16130.
4. P. Gauthier. *Building a Better Repository Map with Tree-sitter.* Aider project, 2023.
5. S. Kapoor, B. Stroebl, Z. S. Siegel, N. Nadgir, A. Narayanan. *AI Agents That Matter.* TMLR 2025. arXiv:2407.01502.
6. C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, J. E. Gonzalez. *MemGPT: Towards LLMs as Operating Systems.* UC Berkeley, 2023. arXiv:2310.08560.
7. Anthropic et al. *Model Context Protocol.* Specification, modelcontextprotocol.io, 2024–2026.
8. A. Rundgren, B. Jordan, S. Erdtman. *JSON Canonicalization Scheme (JCS).* RFC 8785, IETF, 2020.
