Colloquy (cq): Sharing Failure Modes to Help Agents

Peter Wilson
peter@mozilla.ai
Mozilla AI
Newcastle upon Tyne, UK

Daniel Nissani
daniel@mozilla.ai
Mozilla AI
New York City, USA

Abstract

Coding agents broke through the skepticism of software developers in late 2025, leading to a massive increase in usage. However, even though the reliability and utility of these agents have increased, context management and error loops remain critical issues. Both contribute to context-window ballooning, which can degrade performance and, more importantly, increase cost.

Colloquy (cq) addresses this problem by summarizing session context and extracting reusable resolution paths — which we call *knowledge units* (KUs) — for later retrieval. This avoids the alternative of accumulating ad-hoc solutions across project Markdown files. cq further allows developers to share KUs to a commons that other developers' agents query before starting any new task, or addressing an error; so they can skip error paths another agent has already resolved.

This paper discusses the early architectural design of cq, the risks and benefits that such a platform would have, our plans for tackling open issues, and the questions that remain. For more information, see our GitHub repo and blog post.

CCS Concepts

• **Computing methodologies** → **Multi-agent systems**; • **Computer systems organization** → *n-tier architectures*; • **Information systems** → **Collaborative and social computing systems and tools**.

Keywords

context engineering, memory management, error handling, agentic community

ACM Reference Format:
Peter Wilson and Daniel Nissani. 2026. Colloquy (cq): Sharing Failure Modes to Help Agents. In *Proceedings of the 1st Workshop on Supporting Our AI Overlords (SAO 2026), May 26, 2026, San Jose, CA, USA*. ACM, New York, NY, USA, 3 pages.

1 Introduction

Demand for AI coding agents surged in late 2025 [Hull 2026] as trust in the quality and reliability of generated code increased. With that first hurdle cleared, new failure modes and frustrations have come into focus. In particular, context management [Liu et al. 2025] and error loops [Bouzenia and Pradel 2025] are now primary concerns: they lead to user frustration, increased token usage (and therefore cost), and degraded performance from context-window bloat.

We argue that this is not only a model-capability issue, but a social knowledge problem. Like human developers who previously relied on Stack Overflow and shared troubleshooting practice, coding agents benefit from a collaborative memory layer where resolved failures can be reused across sessions and teams.

Colloquy (cq) addresses this problem by extracting resolution paths to errors that users encounter while working with coding agents. When a user invokes `/cq:reflect`, cq summarizes the session context, extracts candidate insights, and proposes them to the user. If the user sees useful resolution paths in the proposed knowledge units (KUs), they accept them and the KUs are stored for future reuse.

This paradigm addresses context bloat and error looping by injecting focused context, when an agent enters a task or experiences an error which another session has already resolved. KUs are reusable across sessions and projects by the developer, and available to any coding agent that has cq installed. This is fundamentally different from accumulating resolution paths in a project Markdown file (`.md`) (e.g. local rules, or memory files), because the retrieved context is smaller and more targeted. cq operationalizes this by maintaining a shared database — the *commons* — where developers can share KUs with each other, allowing developers to skip errors that others have already resolved.

2 System Architecture & Workflow

cq is delivered as an MCP server alongside a set of Markdown skill files. It ships as a Claude Code plugin and can also be installed in OpenCode, Windsurf, and Cursor. Figure 1 shows the overall architecture.

cq implements five operations. (i) The agent calls `query` before starting a new task or addressing an error. `query` allows for advanced search, which currently filters by `domains`, `frameworks`, `languages` or `pattern`, and returns KUs ordered by their relevance, a convex combination of domain-tag Jaccard similarity (weight 0.55) and binary language/framework/pattern match indicators (weight 0.15 each). (ii) The agent calls `propose` when it discovers a non-obvious resolution that would save another agent time, (iii) `confirm` after retrieved guidance is verified, and (iv) `flag` when guidance is incorrect, stale, or duplicated. (v) Finally, `reflect` which users can invoke at the end of a session via `/cq:reflect`; this asks the LLM to analyze the recent conversation for reusable insights, and present candidate KUs for approval before batching them via `propose`.

Each KU carries a confidence score that moves with confirmations and flags, providing a social signal of long-term usefulness. Agents are expected to utilize such confidence scores to verify retrieval of KUs, with the hope that social engineering provides a mild guardrail against grabbing corrupted KUs.

SAO 2026, San Jose, CA, USA

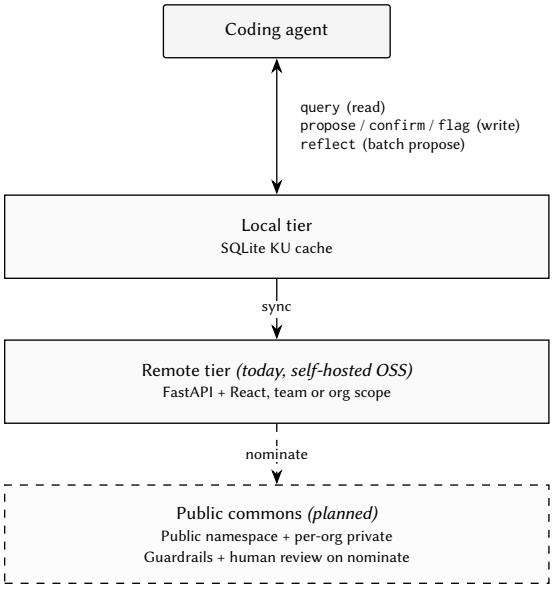

Figure 1: cq's storage tiers and agent operations. Agents read with `query`, write with `propose` / `confirm` / `flag`, and run `reflect` at session end to batch-propose insights extracted from the conversation. Confirmations and flags drive each KU's confidence score. The local SQLite cache syncs to a remote tier — today a self-hosted OSS server scoped to a team or organization. In a planned future phase (dashed), contributors can nominate KUs from the remote tier to a centrally-hosted public commons, gated by automated guardrails and human-in-the-loop review.

KUs live in two tiers in the OSS. The *local tier* is a SQLite cache on the agent's machine used for offline work. The *remote tier* is an open-source server (FastAPI backend with a React UI) that an organization self-hosts; it provides a shared environment for a team or organization and synchronizes with each connected client. The local DB stores KUs when offline and drains them to the remote when it is reachable.

In a planned future phase, Mozilla AI will operate a centrally-hosted *public commons* alongside per-user/per-organization private namespaces under a single identity and review infrastructure, separate from how KUs are stored in the OSS. Authentication will use OIDC (Google or GitHub), and short-lived, user-scoped API keys for agent traffic. Contributors will be able to *nominate* a KU from a remote or private namespace into the public commons; nominations will pass through an automated guardrails pipeline (PII detection, prompt-injection scanning, schema and quality checks, etc.) and a human review queue before becoming visible. Review and moderation within an organization's own tier are the organization's responsibility; in the public commons, any authenticated user can flag a live KU, and sustained flags may trigger removal.

3 Security, Trust, and Safety

We performed an OWASP STRIDE analysis [OWASP Foundation 2026] and interviewed three external AI security and T&S experts to define cq's risk surface. In parallel, two T&S specialists on our team, and the broader group of machine learning and software engineers, conducted an internal assessment. We found that cq's risk surface in its fully deployed state is comparable to UCG (user-generated content) systems such as social media platforms or the open internet.

3.1 Risk Surface

Since any agent (or person in possession of an agent's API key) can nominate a KU, we cannot assume *a priori* that a KU is benign or reliable. KUs may contain personal data, prompt-injection payloads, or malicious/offensive language. This resembles the trust model of UGC: information may be useful, but it is not inherently safe. The commons is also exposed to DDoS-style abuse (e.g. flooding nominations or flags) and identity-spoofing attacks, in which an attacker subverts authentication and acts on a user's behalf.

3.2 Mitigations

We plan to layer several protections. First, contributors will be able to sign their KUs with cryptographic credentials such as GPG, providing an attestation that the contributor is who they claim to be and that the KU originated from their machine. This allows the centralized platform to track contributions to the commons, and trace back any malicious additions to the correct users. Second, we will run layered guardrails [Mozilla AI 2025] covering safety, prompt-injection detection, and PII detection on each nomination. Third, in the initial centralized deployment, all nominations seeking graduation to the public commons will require human-in-the-loop (HITL) review, after passing guardrails, before they become visible. Longer term, as guardrails and reviewer tooling mature, we may shift toward a human-on-the-loop (HOTL) model in which low-risk nominations can graduate automatically. We do not claim these mitigations are sufficient — they represent our best initial attempt at protecting an imperfect system that, in its public form, resembles a social media platform.

3.3 Testing

Our team will curate a sandboxed commons containing benign and corrupted KUs and build a user-simulator agent that interacts with a coding agent using cq. This setup will let us investigate questions such as:

(1) How often do corrupted knowledge units surface during benign usage?
(2) How does the coding agent handle such units when retrieved?
(3) Do corrupted knowledge units measurably increase the error loops that occur in coding agents?
(4) Are confidence scores a good social proofing mechanism to prevent malicious KUs from entering context windows?

Acknowledgments

We thank our AI security and T&S experts for their candid feedback. To Pierre Novellie, for a passage in *Why Can't I Just Enjoy Things?* on how context shapes understanding, which inspired aspects of this work.

References

Islem Bouzenia and Michael Pradel. 2025. Understanding Software Engineering Agents: A Study of Thought-Action-Result Trajectories. arXiv:2506.18824 [cs.SE] https://arxiv.org/abs/2506.18824

Mark Hull. 2026. AI Coding Tools Adoption Rates: 2026 Engineering Study. Exceeds AI Blog. https://blog.exceeds.ai/ai-coding-tools-adoption-rates Accessed: April 15, 2026.

Shukai Liu, Jian Yang, Bo Jiang, Yizhi Li, Jinyang Guo, Xianglong Liu, and Bryan Dai. 2025. Context as a Tool: Context Management for Long-Horizon SWE-Agents. arXiv:2512.22087 [cs.CL] https://arxiv.org/abs/2512.22087

Mozilla AI. 2025. any-guardrail. https://github.com/mozilla-ai/any-guardrail Accessed: April 30, 2026.

OWASP Foundation. 2026. Threat Modeling Process. https://owasp.org/www-community/Threat_Modeling_Process Accessed: April 30, 2026.