

Selecting Repository Context through Lexical Anchors and Auditable Graph Paths

Ethan Miller, Liam Parker, Noah Collins

Abstract

Repository retrieval determines which evidence a coding agent can inspect before it proposes an explanation, a test, or a patch. Graph expansion can recover structurally related code that a literal search misses, but it also introduces choices about entry points, relation types, confidence, and context budgets. This conceptual analysis develops an evaluation framework around LARGER's lexically anchored retrieval design. The framework distinguishes finding a relevant file, presenting a useful code span, and supporting a successful downstream action. It treats a graph path as a traceable selection rationale rather than proof that every retrieved neighbor matters. Proposed comparisons hold repository versions and resource budgets fixed, preserve the provenance of each selected item, and evaluate omission as well as excess context. Calibration and abstention are connected to explicitly defined retrieval events. The article contributes a research agenda and reporting specification; it does not reproduce LARGER's benchmarks, train a graph model, or measure a new coding-agent success rate.

Keywords: repository retrieval; lexical search; structural localization; coding agents; context budgets; evidence provenance.

1. Context selection precedes code generation

A coding agent's answer depends on the repository evidence it reaches. A well-formed patch can still be based on the wrong implementation, an obsolete test, or an interface whose name resembles the requested feature. The initial search therefore makes a substantive decision about the problem the agent will try to solve. When that decision is hidden behind a final generated answer, reviewers can struggle to distinguish a reasoning failure from an evidence-selection failure. Repository retrieval deserves evaluation as an observable component, with inputs and outputs that remain available for inspection after the downstream task is complete.

LARGER formulates repository localization through lexical matches, their alignment to structural anchors, and confidence-filtered local expansion within an existing search loop (Hu et al., 2026). That design motivates the central question here: how should a study determine whether added structural context improves the evidence available to an agent? The present discussion uses the method's retrieval principle rather than claiming an independent reproduction of its reported gains. Its key distinction is between a mechanism for selecting context and evidence that this context supports a particular action. The latter depends on the task and requires its own outcome definition.

Literal search supplies a useful entry point because a query can retain distinctive identifiers, error messages, or fragments of a user report. However, the relation between a query and a repository object is not always one-to-one. A name can be shared across modules; an error message can be constructed dynamically; and the relevant test can refer to behavior through a fixture. The graph can help connect these pieces, but only after the initial match has been interpreted. A mistaken anchor can turn a locally coherent neighborhood into evidence for the wrong problem. Expansion should therefore expose the anchor decision rather than bury it inside a retrieval score.

2. Three units of success

File localization asks whether a selected set contains a relevant file. Span selection asks whether the material shown from that file includes the definitions, conditions, and surrounding behavior needed for the task. Action support asks whether an agent can use the selected evidence to produce an acceptable explanation, test, or change. These outcomes are related without being equivalent. A correct file can contribute an irrelevant excerpt, and an apparently incomplete file list can still be sufficient for a narrowly specified question. A benchmark should state which outcome it treats as primary before interpreting improvements in the others.

Lexical ranking methods supply a transparent baseline for the first stage. BM25 formalizes relevance signals based on term occurrence and document statistics (Robertson and Zaragoza, 2009). A repository application must still define a document, tokenization, and treatment of identifiers. Ranking entire files is different from ranking functions or snippets. Splitting compound identifiers can improve one query while increasing ambiguity in another. These choices should be shared across comparison systems when the intended question concerns the added value of graph expansion. Otherwise an apparent structural advantage can partly reflect a stronger lexical preprocessing pipeline.

Heterogeneous retrieval evaluation offers a reason to avoid relying on a single convenient task collection (Thakur et al., 2021). Repository questions differ in how much they expose the needed identifier, whether they concern an implementation or its callers, and whether a relevant file even exists. Transfer among these conditions is an empirical question. A useful study would distinguish exact-name queries, behavior-only descriptions, failure traces, and questions spanning multiple components. It would preserve their frequencies rather than construct an aggregate whose composition changes between systems. A higher average score can conceal a serious regression in a category important to the intended workflow.

3. A graph path explains a selection decision

Typed structural relationships can make a retrieval result easier to inspect. An import edge, a call relationship, a type connection, and a code-to-test association do not carry the same implication. A caller may reveal usage assumptions, whereas a test may reveal an intended edge case. Neither relationship guarantees that the neighbor is relevant to the current question. The selection record should retain the relation type, direction, source location, and repository version. A reader can then assess the reason for expansion without treating graph connectivity as a universal relevance label.

Inductive graph representation learning demonstrates how graph-based methods can construct representations for previously unseen nodes using local information (Hamilton et al., 2017). That literature supplies a useful conceptual contrast between memorizing a fixed graph and applying a representation procedure to new objects. It does not imply that LARGER uses GraphSAGE, nor that a repository edge has the same semantics as an edge in another application. The broader lesson for evaluation is to specify what information is available when the graph changes. An approach that requires rebuilding a complete index faces different update costs from one that can incorporate a small local modification.

Graph incompleteness also needs a visible meaning. Static analysis can miss relationships generated through reflection, dynamic loading, configuration, or build steps. A missing edge may therefore indicate an unobserved relationship rather than a genuine absence. A retrieval system should avoid converting this distinction into unwarranted certainty that no relevant dependency exists. An informative test collection would include cases with known incomplete structural coverage and record whether the system falls back

to another search. These are proposed tests. No parser coverage or dynamic-dependency experiment was conducted for this article.

Version identity is particularly important during active editing. A retrieved span can be current while its graph neighborhood was constructed from an earlier checkout. In that situation, the path that justifies its selection may no longer describe the visible code. The evidence record should identify the commit and any local modifications used for both lexical indexing and structural extraction. If those views differ, the system should expose the difference. A reproducible lookup is more than the ability to repeat a query string: it requires the same searchable objects and the same relationships among them.

4. Useful context under a declared budget

A graph can increase recall by returning more neighbors, but a coding agent has finite attention, context capacity, and time. Excess material can compete with decisive details. A fair comparison therefore needs a budget expressed in quantities relevant to use, such as code tokens shown, tool interactions, elapsed time, and index-maintenance cost. Equal numbers of returned files are insufficient when file lengths differ substantially. The analysis should also distinguish the cost of preparing a reusable repository index from the marginal cost of answering one query. Their relative importance changes with the number of tasks performed on the same checkout.

Retrieval-augmented generation connects selected external evidence to a language model's output (Lewis et al., 2020). In a code repository, the selected material can include executable definitions and tests with strong internal dependencies. The evaluation should preserve which spans were actually supplied to the generator and which were merely found by a search tool. A system can score well on an offline ranked list while presenting a different truncated context to the agent. Reporting both stages allows a reviewer to locate whether evidence was lost through ranking, expansion, deduplication, or final assembly.

An informative ablation would compare lexical results alone, fixed structural expansion, and confidence-filtered expansion under matched presentation budgets. A further comparison could retain the same candidates while changing only their ordering. These contrasts separate candidate discovery from context organization. They should use the same base agent and permit the same additional searches. If one condition receives more tool calls after an unhelpful first retrieval, the resulting performance difference concerns the complete interaction policy. That can be a worthwhile outcome, provided it is not described as the effect of the initial ranked list alone.

The budget should include human checking where that work is part of the workflow. A result whose graph paths are easy to follow may reduce review effort even if its file recall changes little. Conversely, a large neighborhood with plausible but weak paths may impose more inspection than a short lexical list. Proposed review studies could measure time to identify a sufficient evidence set and the frequency of unnoticed omissions. Such a study would require a defined reviewer task and consistent stopping rule. The present framework does not assign invented time savings to structural retrieval.

5. Confidence requires a defined event

A confidence score can describe anchor alignment, the reliability of a graph edge, the relevance of a candidate, or the adequacy of an entire context set. These events have different labels. Calibration research emphasizes evaluating whether reported confidence corresponds to the frequency of a specified correctness event (Guo et al., 2017). A retrieval score should not be called calibrated merely because it orders candidates sensibly. Before drawing a calibration curve, investigators must decide what a positive case means and how it is adjudicated. An entire neighborhood's adequacy may depend on relationships among items that individual candidate labels do not capture.

The negative cases deserve equal attention. A query may refer to an absent feature, an outdated revision, or information outside the repository. Returning the nearest structural neighborhood can conceal that the requested evidence is unavailable. Selective classification offers a framework for examining systems that can decline to make a decision (Geifman and El-Yaniv, 2017). In retrieval, the analogous action could request clarification, broaden the search, or report that sufficient support was not found. Its evaluation should include the resulting coverage and workload. Automatically treating every empty answer as a safe success would reward a system that never retrieves anything.

Confidence thresholds also interact with repository characteristics. A threshold that works for a densely connected codebase can remove necessary neighbors from a sparse or partially parsed repository. Testing this sensitivity requires retaining the threshold-selection procedure and evaluating it on held-out repositories or tasks. Comparing many thresholds on the final evaluation set and reporting only the best one changes the evidentiary role of that set. A fixed policy can be less impressive numerically while providing a clearer estimate of transfer. Both tuned and fixed settings can be reported if their roles remain explicit.

6. Benchmark outcomes and engineering obligations

SWE-bench makes repository-level issue resolution an observable task involving real issue descriptions and code changes (Jimenez et al., 2024). It provides a useful setting for asking how localization contributes to a larger workflow. A reference patch is nevertheless an observed solution, not a complete list of every file that might be useful to inspect or every acceptable repair. A retrieval evaluation built from changed-file labels should explain that limitation. Finding all reference files can be an informative diagnostic, but it does not prove that the retrieved evidence is sufficient or that alternative paths are irrelevant.

Tests introduce another boundary. A generated patch can pass a particular test set while violating an unstated requirement, or fail because of an execution-environment issue unrelated to retrieval. A complete study should retain environment setup, test outcomes, and failure categories alongside the context trace. It should avoid attributing every downstream failure to the retriever or every success to a graph edge. Carefully matched comparisons can estimate the effect of a retrieval policy on task outcomes; they do not automatically explain the internal causal contribution of each selected span.

Machine learning systems accumulate dependencies beyond their central model (Sculley et al., 2015). Repository retrieval has a similar lifecycle: parsers, build conventions, generated files, index formats, and tool wrappers can change independently. A graph that is cheap to query can still be expensive to maintain accurately. Reporting update frequency, invalidation rules, and unsupported languages helps make those costs visible. A deployment-oriented evaluation would measure recovery after common repository changes, not only retrieval on a frozen clean checkout. That extension remains a proposed engineering study rather than an experiment performed here.

7. A compact evidence record

The reporting record proposed here contains a versioned query, repository snapshot, lexical candidates, selected anchors, typed expansion paths, filtered candidates, and final presented spans. It also records budgets, fallback searches, and the downstream task outcome. The FAIR principles motivate identifiers and metadata that make scientific objects findable and reusable (Wilkinson et al., 2016). Applying that idea to retrieval traces can make a failure inspectable without requiring a reader to reconstruct an entire interactive session from prose. Reusability still depends on access permissions and on preserving the actual object versions referenced by those identifiers.

The conceptual contribution is to organize these records around three questions: what was selected, why it was selected, and what the selection enabled. LARGER's lexical-to-structural transition is a concrete point at which those questions can be asked. A strong evaluation would test whether structural expansion supplies useful missing evidence under a realistic budget while preserving a clear route back to the code. It would also report when the graph is incomplete or the system cannot establish adequate support. This approach gives repository retrieval a measurable role in coding-agent reliability without confusing a persuasive path through a graph with a verified software solution.

References

- Hu, Yuntong, et al. "LARGER: Lexically Anchored Repository Graph Exploration and Retrieval." arXiv preprint arXiv:2605.16352 (2026).
- Robertson, Stephen, and Hugo Zaragoza. "The Probabilistic Relevance Framework: BM25 and Beyond." *Foundations and Trends in Information Retrieval*, vol. 3, no. 4, 2009, pp. 333-389. <https://doi.org/10.1561/15000000019>.
- Thakur, Nandan, et al. "BEIR: A Heterogeneous Benchmark for Zero-Shot Evaluation of Information Retrieval Models." *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 5370-5382. <https://arxiv.org/abs/2104.08663>.
- Hamilton, William L., Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs." *Advances in Neural Information Processing Systems*, vol. 30, 2017. <https://arxiv.org/abs/1706.02216>.
- Lewis, Patrick, et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 9459-9474. <https://arxiv.org/abs/2005.11401>.
- Guo, Chuan, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. "On Calibration of Modern Neural Networks." *Proceedings of the 34th International Conference on Machine Learning*, PMLR, vol. 70, 2017, pp. 1321-1330. <https://proceedings.mlr.press/v70/guo17a.html>.
- Geifman, Yonatan, and Ran El-Yaniv. "Selective Classification for Deep Neural Networks." *Advances in Neural Information Processing Systems*, vol. 30, 2017. <https://arxiv.org/abs/1705.08500>.
- Sculley, D., et al. "Hidden Technical Debt in Machine Learning Systems." *Advances in Neural Information Processing Systems*, vol. 28, 2015. <https://papers.nips.cc/paper/2015/hash/86df7dcfd896fcdf2674f757a2463eba-Abstract.html>
- Wilkinson, Mark D., et al. "The FAIR Guiding Principles for Scientific Data Management and Stewardship." *Scientific Data*, vol. 3, 2016, article 160018. <https://doi.org/10.1038/sdata.2016.18>.
- Jimenez, Carlos E., et al. "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" *International Conference on Learning Representations*, 2024. <https://arxiv.org/abs/2310.06770>.