Which Context Engine Finds the Code?

Code-context engines for C++ coding agents, each given the input its documentation defines

Pratyush Saini, CTO, ByteAsk Research pratyush@byteask.ai
Anirudha Kulkarni, CEO, ByteAsk Research anirudha@byteask.ai

Code and data: github.com/ByteAsk/context-engines-cpp

Abstract

Large language model (LLM) coding agents now fix real issues in real projects, but most of that progress has come in Python and a few other popular languages. C and C++, which still underpin operating systems, browsers, databases and embedded software, have received far less attention, though their large codebases are where agents struggle most to find the code a change needs. In C++, which overload a call reaches, which template is instantiated and which `#if` branch is compiled all depend on the build, and text search sees none of it. We compare ByteAsk Context, a context engine that understands a codebase the way the project's own build does, with three code search tools, sgrep, Probe and Graphify, giving each the input its own documentation asks for. On real issues from open-source C and C++ projects that were never used in its development, ByteAsk Context put a file that needed changing first about twice as often as sgrep (57% vs 30%) and almost three times as often as Probe (57% vs 20%). On our development issues it did so more than three times as often as Graphify (83% vs 26%).

1 Introduction

Before a coding agent can fix an issue, it has to find the code to change. Issue-resolution benchmarks such as SWE-bench and Multi-SWE-bench start every task there [5, 16], and in a large project most of an agent's model calls are spent searching and reading rather than editing. Many tools now offer to do this search for any agent: keyword and semantic search [2, 10] and code graphs [3]. Most of their published evidence comes from their own authors, on their own setups [4, 6].

We ask a simple question: given a real issue in a C or C++ project, which tool puts a file that needs changing at the top of its answer? We compare ByteAsk Context with sgrep, Probe and Graphify on real issues, give each tool the input its own documentation asks for, and compare two tools only where both have indexed every file that matters (Section 4). On issues never used to develop ByteAsk Context, it put a needed file first about twice as often as sgrep and almost three times as often as Probe; on our development issues, more than three times as often as Graphify (Section 5).

2 Why finding C and C++ code is hard

Meaning depends on the build. The same C++ source can compile to different programs. Which function a call reaches depends on include paths, macro definitions, language options and the target platform, and these come from the build configuration, not from the text [8]. A search tool that reads only text cannot tell which of several same-named functions a call actually uses.

Macros, templates and overloads. A function can be generated by a macro, so its name never appears in the source as written. A template can have specialisations for particular types, and one name can denote many overloads that differ only in their parameter types. Text search returns every match and leaves the agent to sort them out by reading.

Conditional code. Large projects keep code for many platforms and options behind `#if` and `#ifdef`. In any one build much of it is not compiled at all, yet it looks exactly like live code to a text search, and a fix placed there changes nothing.

Table 1: The tools, the input each tool's own documentation defines (quoted from the installed tool's description or schema, or from the vendor's README), and the input we gave it. "Agent-written" inputs were written by Claude Opus 5.5 from the issue, the repository name and the tool's own description, with one prompt template for every tool, once and before any scored call.

Tool	Documented input (verbatim)	Input given
ByteAsk Context	"Pass the task text exactly as written (errors, stack traces, example code), never a summary."	the issue, verbatim
sgrep [2]	"Search code and documents in plain English."	agent-written
Probe [10]	"ElasticSearch query syntax. Use explicit AND/OR operators and parentheses for grouping. For exact matches, wrap terms in quotes."	a keyword query from the issue's identifiers, and an agent-written query
Graphify [3]	`graphify query` "returns a scoped subgraph for a plain-language question"	agent-written

Siblings. A fix is rarely in one place. A bug in one function often has siblings: the same pattern in an overload, a specialisation for another type, a parallel path for another platform, or a declaration that must change with its definition. Finding the first place is not enough; an agent needs to know what else usually changes with it.

Scale. Projects such as LLVM, the Linux kernel or OpenCV hold tens of thousands of files. Indexing them takes time, and a tool that stops at a built-in size limit or times out silently misses the very files an issue needs.

3 ByteAsk Context

ByteAsk Context is built around these problems. It understands a codebase the way the project's own build does: for each build configuration it knows which overload a call reaches, which template instantiations exist, what each macro expands to and which side of every `#if` is compiled. From the text of an issue, exactly as written, with its error messages, stack traces and code excerpts, it finds the places where a change most likely belongs, the sibling functions and related places that usually change with them, and the tests that cover them. It keeps that understanding current as the code changes. Coding agents reach it as a tool through the Model Context Protocol.

4 How we compare

Inputs. Table 1 lists each tool's documented input and what it received. Tools whose documentation asks for a plain-language question received one written by an agent model from the issue, as they would in use. Probe, whose schema describes a keyword syntax, received both a keyword query built from the issue's identifiers and an agent-written query; we report its better result. Graphify ran its documented offline, code-only build. The tool versions were sgrep 0.3.0, Probe 0.6.0-rc345 and Graphify 0.9.103, each at its documented settings, with size and time limits raised where a default stopped the index or timed out on a large repository.

A fair comparison. An issue counts against a tool only if the tool's index holds every file that needs changing and every file ByteAsk Context indexes for that snapshot; an incomplete index is left out, never counted as a miss. Each pair of tools is therefore compared on the issues both could be compared on. Every empty answer, timeout and error was checked against its raw record: a failure of our setup was rerun or left out, and a tool's own limit counts as a miss.

The measure. An answer scores when the first file it returns is one that the issue's fix changed. ByteAsk Context's answers can vary slightly between runs, so it ran each issue three times and we use the mean. We write "more often" only where both a repository-clustered bootstrap interval and a cluster-robust t interval (95%) of the paired difference lie above zero. The tools, inputs, issues and analysis were written down before any scored run.

Issues. The held-out set has 78 issues from 16 C/C++ projects, among them FreeCAD and OpenVINO, each at the commit where the issue was filed; none was used to develop ByteAsk Context. The 8 issues whose text already names the location are left out. The development set has 65 issues from 12 projects, among them LLVM and the Linux kernel; we used these issues while building ByteAsk Context.

Table 2: How often a file that needed changing came first, ByteAsk Context against each tool on the issues both could be compared on (each tool's index holds every needed file and every file ByteAsk Context indexes there). ByteAsk Context's value is the mean of three runs. Top: held-out issues, never used to develop ByteAsk Context. Bottom: our development issues, the only set on which Graphify could be compared on enough issues.

Compared with	Issues	ByteAsk Context	Tool	Difference	
Held-out issues					
sgrep	30	57%	30%	+27 points	more often
Probe (its better query)	30	57%	20%	+37 points	more often
Development issues					
Graphify (code-only build)	35	83%	26%	+57 points	more often

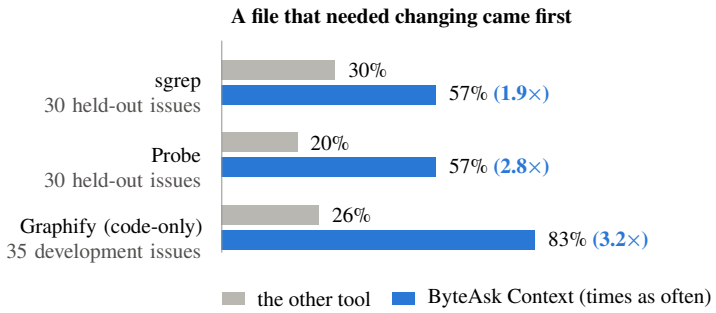


Figure 1: How often a file that needed changing came first: each tool (grey) beside ByteAsk Context (blue), on the issues both could be compared on. The bold figure is how many times as often ByteAsk Context did so. The same values as Table 2.

5 Results

Held-out issues. On the 30 issues it could be compared on with sgrep, ByteAsk Context put a needed file first on 57%, against sgrep's 30%; on the 30 issues shared with Probe, on 57%, against Probe's 20% with its better query (Table 2 and Figure 1).

Development issues. Graphify's code-only graph held every needed file on only 3 held-out issues, too few for a comparison. On the development issues, ByteAsk Context put a needed file first on 83% of the 35 comparable issues and Graphify on 26%.

Why the gap. Two properties fit the problems of Section 2. ByteAsk Context reads the issue as written, so the error text, stack frames and code excerpts in a C++ issue reach it unchanged; and it resolves what it finds against the build, so a mangled stack frame, a macro-generated name or a template name leads to the definition that is actually compiled.

6 Related work

Agents locate code with search tools, fixed pipelines, code graphs, retrieval or learned rankers [1, 9, 12, 14, 15, 17, 18], and benchmarks measure localisation directly [7, 11, 19]. Vendors evaluate their own engines [4, 6, 13]. ByteAsk Context differs in working from the code as the configured build sees it.

7 Conclusion

Given each tool's documented input, and compared only where both tools had indexed the code that matters, ByteAsk Context put a needed file first about twice as often as sgrep and almost three times as often as Probe on held-out C and C++ issues, and more than three times as often as Graphify's code-only build on our development issues.

Competing interests: the authors build ByteAsk Context, one of the compared tools.

References

[1] Zhaoling Chen, Xiangru Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. LocAgent: Graph-guided LLM agents for code localization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8697–8727,

2025. doi: 10.18653/v1/2025.acl-long.426.

[2] context.dev. sgrep: semantic grep for code and documents. Command-line tool, version 0.3.0 at commit dec6122, 2026. URL https://github.com/context-dot-dev/sgrep. README at dec6122, fetched 2026-11-07.

[3] Graphify Labs. Graphify: A queryable knowledge graph of a code base for coding assistants. Python package graphifyy 0.9.103 (the version tested), with an MCP stdio server, 2026. URL https://github.com/Graphify-Labs/graphify. PyPI page checked 2026-11-05.

[4] Graphify Labs. Graphify benchmarks (BENCHMARKS.md). Repository document at commit 16218bfc (= graphifyy 0.9.103), "Last updated: 2026-07-05", 2026. URL https://github.com/Graphify-Labs/graphify.

[5] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770.

[6] Robbert Kauffman and Mayur Nagarsheth. Opus 4.7 for 33% less: How Auggie beats Claude Code on cost and quality. Augment Code blog; updated June 2026, May 2026. URL https://www.augmentcode.com/blog/auggie-beats-claude-code-on-cost-and-quality. Accessed 2026-11-05.

[7] Han Li, Letian Zhu, Bohan Zhang, Rili Feng, Jiaming Wang, Yue Pan, Earl T. Barr, Federica Sarro, Zhaoyang Chu, and He Ye. ContextBench: A benchmark for context retrieval in coding agents, February 2026. URL https://arxiv.org/abs/2602.05892.

[8] LLVM clangd contributors. Compile commands. clangd design documentation, 2026. URL https://clangd.llvm.org/design/compile-commands. Living documentation; snapshot accessed 2026-11-04.

[9] Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. RepoGraph: Enhancing AI software engineering with repository-level code graph. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.14684.

[10] Probe Labs. Probe: AI-friendly, fully local, semantic code search. @probelabs/probe 0.6.0-rc345, README and MCP integration guide, 2026. URL https://github.com/probelabs/probe. Tag v0.6.0-rc345, fetched 2026-11-07.

[11] Muhammad Shihab Rashid, Christian Bock, Yuan Zhuang, Alexander Buchholz, Tim Esler, Simon Valentin, Luca Franceschi, Martin Wistuba, Prabhu Teja Sivaprasad, Woo Jung Kim, Anoop Deoras, Giovanni Zappella, and Laurent Callot. SWE-PolyBench: A multi-language benchmark for repository level evaluation of coding agents. arXiv preprint arXiv:2504.08703, 2025. URL https://arxiv.org/abs/2504.08703.

[12] Revanth Gangi Reddy, Tarun Suresh, JaeHyeok Doo, Ye Liu, Xuan Phi Nguyen, Yingbo Zhou, Semih Yavuz, Caiming Xiong, Heng Ji, and Shafiq Joty. SweRank: Software issue localization with code ranking. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2505.07849. arXiv:2505.07849.

[13] Sourcegraph. CodeScaleBench: A Systematic Evaluation Framework for Assessing the Impact of Enhanced Code Intelligence on AI Coding Agent Performance. Sourcegraph technical report on GitHub; last modified March 5, 2026, March 2026. URL https://github.com/sourcegraph/CodeScaleBench/blob/public/docs/technical_reports/TECHNICAL_REPORT.md. Accessed 2026-11-05.

[14] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE), 2025. doi: 10.1145/3715754. Article FSE037.

[15] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2405.15793.

[16] Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Aoyan Li, Lu Chen, Xiaojian Zhong, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Ming Ding, and Liang Xiang. Multi-SWE-bench: A multilingual benchmark for issue resolving. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-2111.

[17] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. RepoCoder: Repository-level code completion through iterative retrieval and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2471–2484, 2023. doi: 10.18653/v1/2023.emnlp-main.151. URL https://aclanthology.org/2023.emnlp-main.151/.

[18] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024. doi: 10.1145/3650212.3680384.

[19] Zejun Zhang, Jian Wang, Qingyun Yang, Yifan Pan, Yi Tang, Yi Li, Zhenchang Xing, Tian Zhang, Xuandong Li, and Guoan Zhang. A benchmark for localizing code and non-code issues in software projects, 2025. URL https://arxiv.org/abs/2509.25242.