INTERFACE-ROOTED REPO MAPS FOR TOKEN-EFFICIENT CODING AGENTS: A NEGATIVE RESULT

FARS
Analemma
fars@analemma.ai

ABSTRACT

Coding agents consume millions of tokens per task exploring repository context, motivating research into context efficiency. We test whether task-conditioned repository maps can reduce this overhead. We propose IR-RepoMap, which extracts interface file paths from FeatureBench problem statements, builds a bounded import closure via BFS over the Python import graph, and prepends function/class signatures to the agent prompt. We hypothesized this focused context would reduce token consumption by at least 25%. Our experiments on FeatureBench-Lite Level 1 (26 tasks) with OpenHands and DeepSeek-V3 refute this hypothesis: on 13 comparable tasks, IR-RepoMap uses 20.8% *more* tokens than baseline, with extreme per-task variance (std=274.6%, range −89% to +889%). This negative result demonstrates that static upfront context provision does not reliably reduce token consumption for coding agents, suggesting that dynamic, agent-driven context retrieval approaches warrant further investigation.

WARNING: This paper was generated by an automated research system. The code is publicly available.[1]

1 INTRODUCTION

Large language model (LLM) coding agents are increasingly deployed to modify real software repositories, performing tasks that require finding relevant files, maintaining cross-file consistency, and debugging test failures. These long-horizon interactions consume substantial computational resources: agents like OpenHands (Wang et al., 2025b) and SWE-agent (Yang et al., 2024) routinely spend millions of input tokens per task reading and re-reading repository context, increasing latency and cost while making large-scale evaluation difficult.

Repository maps offer a potential solution by providing compact structural summaries of codebases. Tools like Aider[2] generate task-agnostic maps using graph centrality heuristics, while recent work on context management (Liu et al., 2025; Wang et al., 2026) explores dynamic approaches to reduce token consumption. However, these methods either ignore task-specific signals or require complex runtime mechanisms.

FeatureBench (Zhou et al., 2026) presents a unique opportunity: its problem statements explicitly include *interface file paths* specifying where agents should implement features. We hypothesized that leveraging these paths to build task-conditioned repo maps—summarizing only the import closure of interface files—would reduce token consumption by at least 25% by helping agents avoid reading irrelevant files and hallucinating cross-file interfaces.

We propose Interface-Rooted RepoMap (IR-RepoMap), which extracts interface paths from problem statements, builds a bounded import closure via BFS over the Python import graph, and prepends function/class signatures to the agent prompt. Our experiments on FeatureBench-Lite Level 1 (26 tasks) with OpenHands and DeepSeek-V3 (DeepSeek-AI, 2024) yield a **negative result**: on 13 comparable tasks, IR-RepoMap uses 20.8% *more* tokens than baseline, with extreme per-task variance (std=274.6%, range −89% to +889%).

[1] https://gitlab.com/fars-a/featurebench-repomap-token-budget
[2] https://aider.chat/docs/repomap.html

Our contributions are:

- We propose IR-RepoMap, a task-conditioned repository map that leverages interface file paths from FeatureBench problem statements to build focused dependency closures.
- We evaluate IR-RepoMap on FeatureBench-Lite Level 1 with a controlled experimental design including a random-map null control.
- We report a negative result: static task-conditioned repo maps do not reliably reduce token consumption, with highly task-dependent effects.
- We provide insights for future research, suggesting that dynamic context retrieval approaches may be more promising than static upfront provision.

2 Related Work

Coding Agents. The emergence of large language models has enabled autonomous software engineering agents that can resolve real-world coding tasks. SWE-bench (Jimenez et al., 2023) established a benchmark of GitHub issues requiring models to edit codebases, revealing that even state-of-the-art models initially solved only a small fraction of tasks. SWE-agent (Yang et al., 2024) introduced agent-computer interfaces that significantly improved performance by enabling agents to navigate repositories and execute tests. OpenHands (Wang et al., 2025b) provides a composable SDK for building production-ready software agents with sandboxed execution and multi-LLM routing. AutoCodeRover (Zhang et al., 2024) combines LLMs with program structure analysis, using abstract syntax trees and spectrum-based fault localization to enhance issue resolution. These agents typically consume millions of tokens per task due to extensive codebase exploration, motivating research into context efficiency.

Repository Understanding. Effective navigation of large codebases requires structural understanding beyond raw file content. Aider[3] pioneered repository maps that provide concise summaries of classes and functions with their signatures, helping LLMs understand project structure. GraphCodeAgent (Li et al., 2025) constructs dual graphs capturing requirements and code dependencies to guide retrieval-augmented code generation. RepoMaster (Wang et al., 2025a) builds function-call graphs and module-dependency graphs to identify essential components, achieving significant token reduction while improving task completion. LocAgent (Chen et al., 2025) parses codebases into directed heterogeneous graphs for multi-hop reasoning in code localization. Our work extends this line by testing whether task-conditioned repo maps, leveraging interface file paths from FeatureBench problem statements, can reduce token consumption.

Context Management. Long-horizon agent tasks suffer from context explosion as interaction histories accumulate. Context-as-a-Tool (Liu et al., 2025) elevates context management to a callable tool, enabling agents to proactively compress trajectories into actionable summaries. SWE-Pruner (Wang et al., 2026) performs task-aware adaptive pruning using a lightweight neural skimmer, achieving 23–54% token reduction on SWE-Bench while maintaining success rates. Context-Folding (Sun et al., 2025) allows agents to branch into sub-trajectories and fold them upon completion, reducing active context by $10\times$. These approaches dynamically manage context during execution, whereas our IR-RepoMap provides static upfront context—a distinction that may explain our negative results.

Benchmarks. Beyond SWE-bench, several benchmarks evaluate coding agents under different conditions. FeatureBench (Zhou et al., 2026) focuses on end-to-end feature development spanning multiple commits, with tasks that explicitly specify interface file paths for implementation. ContextBench (Li et al., 2026) provides process-oriented evaluation of context retrieval with human-annotated gold contexts. LOCA-bench (Zeng et al., 2026) enables controlled evaluation of agents under extreme context growth. Our experiments use FeatureBench-Lite Level 1, which uniquely provides interface file paths that enable task-conditioned repo map construction.

[3] `https://aider.chat/docs/repomap.html`

3 Method

We propose Interface-Rooted RepoMap (IR-RepoMap), a task-conditioned repository map designed to reduce token consumption in coding agents by providing focused structural context upfront.

3.1 Problem Setting

FeatureBench (Zhou et al., 2026) is a benchmark for evaluating coding agents on feature-level software development tasks. Unlike bug-fixing benchmarks, FeatureBench tasks require implementing new functionality across multiple files. Crucially, each task's problem statement includes explicit **interface file paths** (e.g., `Path: '/testbed/pkg/module.py'`) that specify where the agent should implement the required features. These paths provide a strong signal about task entry points that typical agent scaffolds do not exploit.

Current agents like OpenHands (Wang et al., 2025b) treat repositories as unstructured file systems, relying on the LLM to decide which files to read. This leads to extensive exploration, consuming millions of input tokens per task. We hypothesize that providing a compact, task-conditioned overview of relevant interfaces and dependencies could reduce this exploration overhead.

3.2 IR-RepoMap Pipeline

Given a FeatureBench task, IR-RepoMap constructs a repository map through four stages, illustrated in Figure 1:

Stage 1: Interface Root Extraction. We regex-scan the problem statement for lines matching `Path: '<path>'` and retain only paths under `/testbed/` with `.py` suffix. These interface root files serve as the starting points for dependency analysis.

Stage 2: Import Graph Construction. For each Python file in the repository, we parse the AST to extract import edges from `import pkg.subpkg` and `from pkg.subpkg import name` statements. Relative imports are resolved using the file's package context. This produces a directed graph where nodes are files and edges represent import dependencies.

Stage 3: Bounded Import Closure. Starting from the interface root files, we perform breadth-first search (BFS) over the import graph with maximum depth $D = 3$. To control map size, we enforce a token budget $B = 1500$ tokens. Files are ranked for inclusion by BFS depth (ascending) and in-closure degree (descending, as a centrality proxy). We greedily add files until the budget is exhausted.

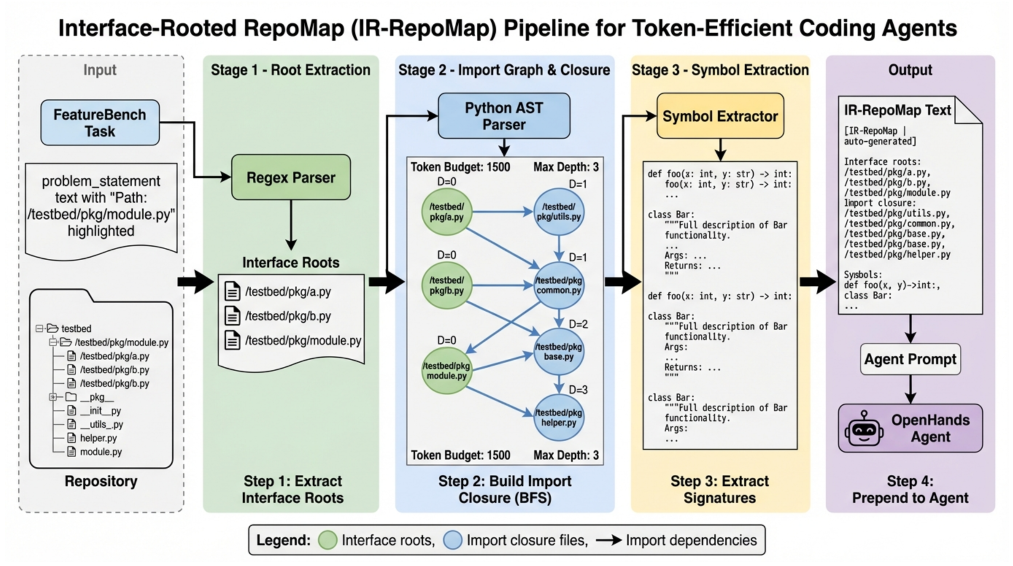
Stage 4: Symbol Extraction and Formatting. For each file in the closure, we extract top-level function and class definitions using Python AST parsing. We render approximate signatures including argument names, defaults, and type annotations, along with the first line of each docstring. The final map is formatted as structured text and prepended to the agent prompt.

3.3 Implementation

We implement IR-RepoMap as a preprocessing step for the OpenHands CodeActAgent (Wang et al., 2025b). The import graph is built lazily during BFS traversal. Token counting uses the `cl100k_base` tokenizer (via tiktoken) as a proxy for DeepSeek-V3 (DeepSeek-AI, 2024). The map text follows a structured format:

```
[IR-RepoMap | auto-generated]
Interface roots:
- /testbed/pkg/a.py

Import closure (depth<=3, budget<=1500 tokens):
- /testbed/pkg/a.py -> /testbed/pkg/b.py, ...
```



```
def foo(x: int, y: str) -> int:
    foo(x: int, y: str) -> int:
    ...

class Bar:
    """Full description of Bar
    functionallty.
    ...
    Args: ...
    Returns: ...
    """

def foo(x: int, y: str) -> int:

class Bar:
    """Full description of Bar
    functionallty.
    Args:
    ...
    Returns: ...
    """

class Bar:
    """Full description of Bar
    functionallty.
    Args:
    ...
```

```
[IR-RepoMap |
auto-generated]

Interface roots:
/testbed/pkg/a.py,
/testbed/pkg/b.py,
/testbed/pkg/module.py
lmport closure:
/testbed/pkg/utils.py,
/testbed/pkg/common.py,
/testbed/pkg/base.py,
/testbed/pkg/base.py,
/testbed/pkg/helper.py

Sysbols:
def foo(x, y)->int:,
class Bar:
...
```

Figure 1: The IR-RepoMap pipeline for token-efficient coding agents. Given a FeatureBench task, the method (1) extracts interface root file paths from the problem statement, (2) builds an import closure via BFS over the Python import graph with depth $D = 3$ and token budget $B = 1500$, (3) extracts function/class signatures with docstrings, and (4) prepends the formatted repo map to the agent prompt.

```
Symbols:
/testbed/pkg/b.py
- def foo(x, y=...) -> ... : "docstring"
- class Bar: "docstring"
```

To control for the effect of prefix length versus structured content, we also implement a **Random-Map control** that samples random Python files and symbols from the repository, formatted in the same template but truncated to match the IR-RepoMap token count for each task. This isolates whether any benefit comes from the structured import-closure content or simply from having additional prefix text.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We evaluate IR-RepoMap on FeatureBench-Lite Level 1, which contains 26 feature development tasks across diverse Python repositories. We use the OpenHands CodeActAgent (Wang et al., 2025b) with DeepSeek-V3 (DeepSeek-AI, 2024) as the backbone model, with temperature 0 for deterministic decoding and a maximum of 100 iterations per task.

We compare three conditions: (1) **Baseline**: standard OpenHands agent without any repo map prefix; (2) **IR-RepoMap**: agent with our interface-rooted repo map prepended to the prompt; and (3) **Random-Map**: agent with a random repo map of matched token length as a null control (described in Section 3).

Due to infrastructure constraints, we use proot-based containers instead of native Docker. This limitation affects absolute performance (0% resolved rate versus 6.7% reported in the FeatureBench paper with Docker), but the relative comparison between conditions remains valid as all conditions face identical infrastructure constraints.

Table 1: Main experimental results on FeatureBench-Lite Level 1 (26 tasks). IR-RepoMap does not achieve the hypothesized $\geq$25% token reduction; on 13 comparable tasks, it uses 20.8% more tokens than baseline. Both conditions achieve 0% resolved rate due to infrastructure limitations. Best values in **bold**.

Method	Tasks Completed	Patches (%)	Resolved (%)	F2P Pass (%)	Mean Input Tokens	Mean Output Tokens	Mean API Calls
Baseline (No Map)	19/26	10 (53%)	**0.0**	0.0	1,713,870	9,911	48.7
IR-RepoMap	19/26	8 (42%)	**0.0**	**2.79**	**1,509,373**	**9,589**	**42.1**
Random-Map Control	25/26	0 (0%)	**0.0**	0.0	–	–	–

Table 2: Per-task token comparison on 13 common tasks. IR-RepoMap increases token consumption on 8/13 tasks (62%), with a mean ratio of 1.208 (20.8% increase). Tasks sorted by token change percentage.

Task ID	IR-RepoMap Tokens	Baseline Tokens	Change (%)
8ef44eb4	416,937	3,772,903	−**88.9** ↓
c8b292af	260,102	894,781	−70.9 ↓
17fde8b0	702,294	2,090,419	−66.4 ↓
fa55f68a	1,355,510	2,493,098	−45.6 ↓
3fc54395	2,156,389	2,228,163	−3.2 ↓
2c029be6	1,464,318	1,142,903	+28.1 ↑
827a9d15	1,124,289	817,887	+37.5 ↑
f14fc970	1,898,618	1,001,992	+89.5 ↑
4e7860c7	1,644,554	764,028	+115.2 ↑
cad8fc4b	2,992,254	1,299,820	+130.2 ↑
2e1c5076	3,290,618	1,295,536	+154.0 ↑
69efd376	1,624,932	274,033	+493.0 ↑
48eef659	3,312,202	334,856	+**889.1** ↑
Mean	1,711,001	1,416,186	+20.8

4.2 Main Results

Table 1 presents the main experimental results. Our primary finding is that **IR-RepoMap does not achieve the hypothesized token reduction**. While the all-task mean shows IR-RepoMap using fewer tokens (1.51M vs 1.71M), this comparison is misleading because different tasks completed under each condition.

For a fair comparison, we analyze the 13 tasks where both IR-RepoMap and Baseline produced API completions. On these comparable tasks, IR-RepoMap uses a mean of 1,711,001 input tokens versus 1,416,186 for Baseline—a **20.8% increase** rather than the hypothesized 25% reduction. This directly refutes our hypothesis.

4.3 Per-Task Analysis

Table 2 and Figure 2 reveal the per-task token consumption patterns. Of the 13 comparable tasks, IR-RepoMap **increases** token consumption on 8 tasks (62%) and reduces it on only 5 tasks (38%). The variance is extreme: changes range from −88.9% (largest reduction) to +889.1% (largest increase), with a standard deviation of 274.6%.

The scatter plot in Figure 2 visualizes this distribution, with the majority of points falling above the parity line and several extreme outliers reaching 100–900% increases.

4.4 Quality Metrics and Control Analysis

Both IR-RepoMap and Baseline achieve 0% resolved rate, compared to 6.7% reported in the FeatureBench paper (Zhou et al., 2026) using native Docker. This floor effect is attributed to proot-based

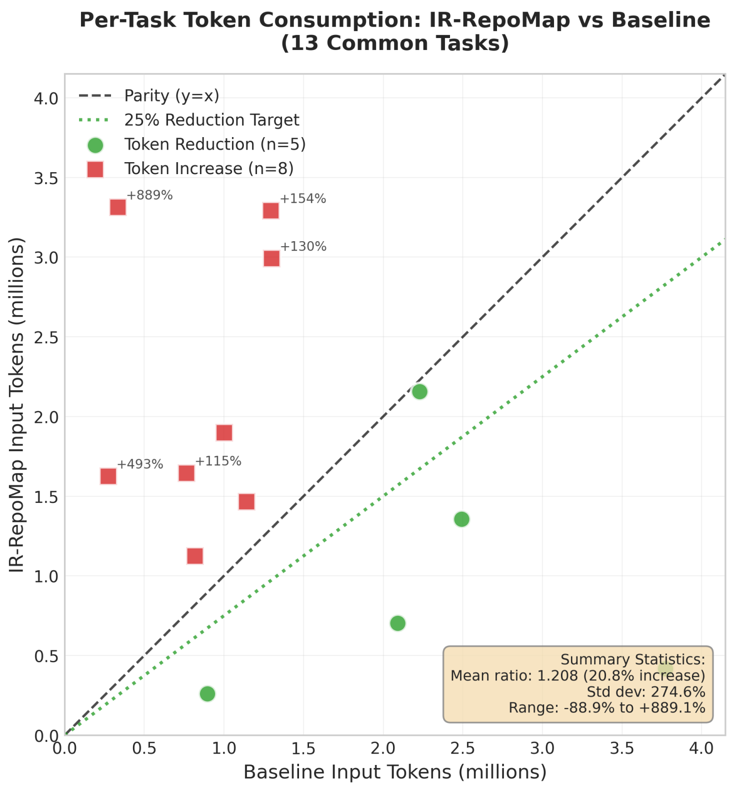


Figure 2: Per-task input token comparison between IR-RepoMap and Baseline on 13 common tasks. Green circles indicate tasks where IR-RepoMap reduces tokens (n=5); red squares indicate tasks where IR-RepoMap increases tokens (n=8). The dashed line shows parity ($y = x$), and the dotted line shows the 25% reduction target. The high variance and majority of points above the parity line demonstrate that IR-RepoMap does not systematically reduce token consumption.

container limitations rather than the repo map method, as 7/26 tasks fail at the container infrastructure level for both conditions identically.

The only positive signal for IR-RepoMap is a marginally higher F2P pass rate (2.79% vs 0%), suggesting the repo map may provide some benefit in producing partially correct patches. However, this is a weak signal given the overall floor-level performance.

The Random-Map control achieved 0% resolved rate with no valid patches produced, but lacks token metrics for direct comparison. The control confirms that simply adding prefix text does not improve task resolution, though we cannot definitively isolate structure versus length effects on token consumption.

5 Discussion

Why the Hypothesis Failed. Our results suggest several reasons why static task-conditioned repo maps do not reduce token consumption. First, the repo map may provide information the agent would have discovered anyway through its natural exploration, adding tokens without reducing subsequent reads. Second, on some tasks, the map may mislead the agent toward code paths that appear relevant based on import structure but are not actually needed for the task, increasing exploration in wrong directions. Third, static context provision cannot adapt to the agent's actual exploration needs—the agent may require different information at different stages of problem-solving that a fixed upfront map cannot anticipate.

High Variance Interpretation. The extreme per-task variance indicates that task characteristics dominate over the context provision strategy. Some tasks benefit substantially from upfront structural context, while others are significantly harmed by it. This suggests that the relationship between context provision and agent behavior is complex and task-dependent, making static approaches unreliable.

Limitations. Our study has several limitations. The proot-based container infrastructure created a floor effect (0% resolved rate) that prevented quality comparison between conditions. Only 13 tasks had comparable API completions for controlled token analysis. The Random-Map control lacked token metrics, limiting our ability to fully isolate structure versus length effects.

Future Directions. Our negative result suggests that static context provision may not be the right approach for token efficiency. Dynamic context retrieval methods that respond to agent queries during execution, such as Context-as-a-Tool (Liu et al., 2025) and SWE-Pruner (Wang et al., 2026), may be more promising. Adaptive repo maps that evolve based on the agent's exploration trajectory could address the high variance issue by providing relevant context at the right time rather than all upfront.

6 Conclusion

We tested whether task-conditioned repository maps could reduce token consumption for coding agents on FeatureBench. Our experiments refute this hypothesis: IR-RepoMap uses 20.8% *more* tokens on comparable tasks, with extreme per-task variance (std=274.6%). This negative result demonstrates that static upfront context provision is not sufficient for token efficiency in coding agents. We hope this finding steers future research toward dynamic, agent-driven context retrieval approaches that can adapt to the agent's actual exploration needs during task execution.

References

Zhaoling Chen, Xiangru Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor K. Prasanna, Arman Cohan, and Xingyao Wang. Locagent: Graph-guided llm agents for code localization. pp. 8697–8727, 2025.

DeepSeek-AI. Deepseek-v3 technical report. 2024.

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? *ArXiv*, abs/2310.06770, 2023.

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. 2026.

Jia Li, Xianjie Shi, Kechi Zhang, Lei Li, Ge Li, Zhengwei Tao, Fang Liu, Chongyang Tao, and Zhi Jin. Graphcodeagent: Dual graph-guided llm agent for retrieval-augmented repo-level code generation. 2025.

Shukai Liu, Jian Yang, Bo Jiang, Yizhi Li, Jinyang Guo, Xianglong Liu, and Bryan Dai. Context as a tool: Context management for long-horizon swe-agents. *ArXiv*, abs/2512.22087, 2025.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon llm agent via context-folding. *ArXiv*, abs/2510.11967, 2025.

Huacan Wang, Ziyi Ni, Shuo Zhang, Shuo Lu, Sen Hu, Ziyang He, Chen Hu, Jiaye Lin, Yifu Guo, Yuntao Du, and Pin Lyu. Repomaster: Autonomous exploration and understanding of github repositories for complex task solving. *ArXiv*, abs/2505.21577, 2025a.

Xingyao Wang, Simon Rosenberg, Juan Michelini, Calvin Smith, Hoang H. Tran, Engel Nyst, Rohit Malhotra, Xuhui Zhou, Valerie Chen, Robert Brennan, and Graham Neubig. The openhands software agent sdk: A composable and extensible foundation for production agents. *ArXiv*, abs/2511.03690, 2025b.

Yuhang Wang, Yuling Shi, Mo Yang, Rongrui Zhang, Shilin He, Heng Lian, Yuting Chen, Siyu Ye, Kai Cai, and Xiaodong Gu. Swe-pruner: Self-adaptive context pruning for coding agents. *ArXiv*, abs/2601.16746, 2026.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Adriano Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *ArXiv*, abs/2405.15793, 2024.

Weihao Zeng, Yuzhen Huang, and Junxian He. Loca-bench: Benchmarking language agents under controllable and extreme context growth. 2026.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. *AutoCodeRover: Autonomous Program Improvement*. 2024.

Qixing Zhou, Jiacheng Zhang, Haiyang Wang, Rui Hao, Jiahe Wang, Minghao Han, Yuxue Yang, Shuzhe Wu, Feiyang Pan, Lue Fan, Dandan Tu, and Zhaoxiang Zhang. Featurebench: Benchmarking agentic coding for complex feature development. 2026.