One Size Does Not Fit All:
Revisiting Code Context Engineering for Repository-Level Code Generation

Yichen Li, Qiye Lin, Yun Peng, Zhihan Jiang, Jinyang Liu,
Chaozheng Wang, Yintong Huo, Cuiyun Gao

Chinese University of Hong Kong / Harbin Institute of Technology / Singapore Management University

FSE 2026

香港中文大學
The Chinese University of Hong Kong

Chinese University of Hong Kong

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Harbin Institute of Technology

Singapore Management University

Read the paper!

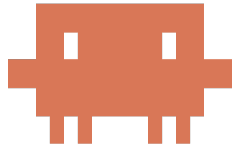
Background & Motivation: Coding Agents

Coding agents are no longer only completing code.

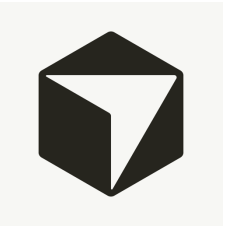
Now they work inside repositories.

They read files, call tools, edit code, run tests, and revise the answer.

Codex



Claude Code



Cursor

Before	complete the current line
Now	finish repository tasks

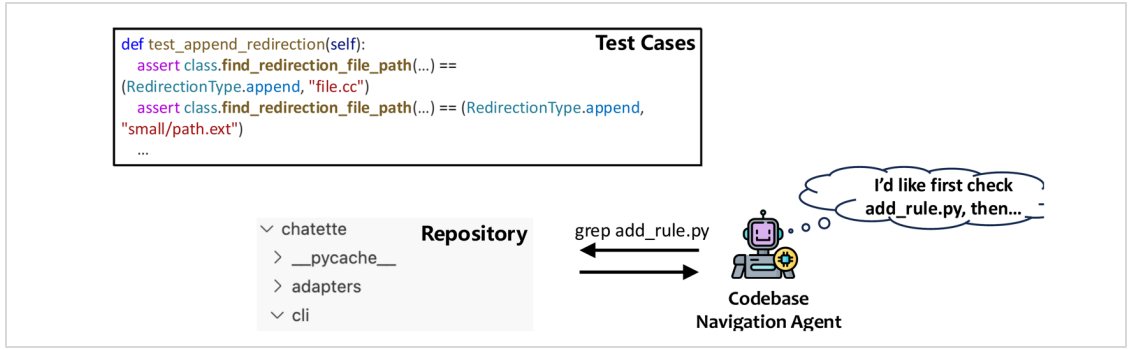
When the task grows to a repository, choosing the context becomes part of the work.

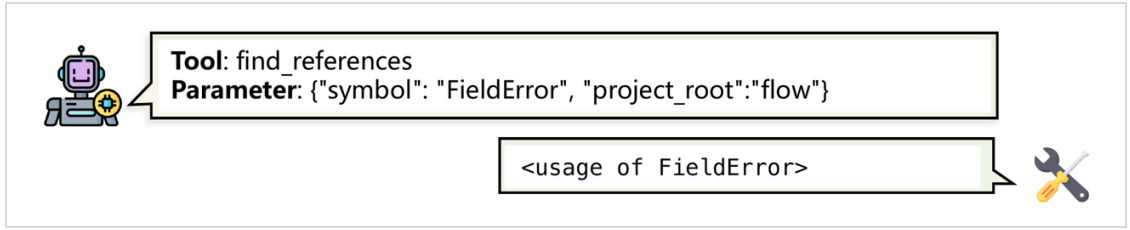
But What Should the Agent Read?

An issue may point to the file to edit, but not to the files that explain the edit.

I know what to edit.
But what else should I read?

The agent still has to find definitions, usages, tests, and sibling code before generation.





Task prompt what to change

Repo context what makes the patch correct

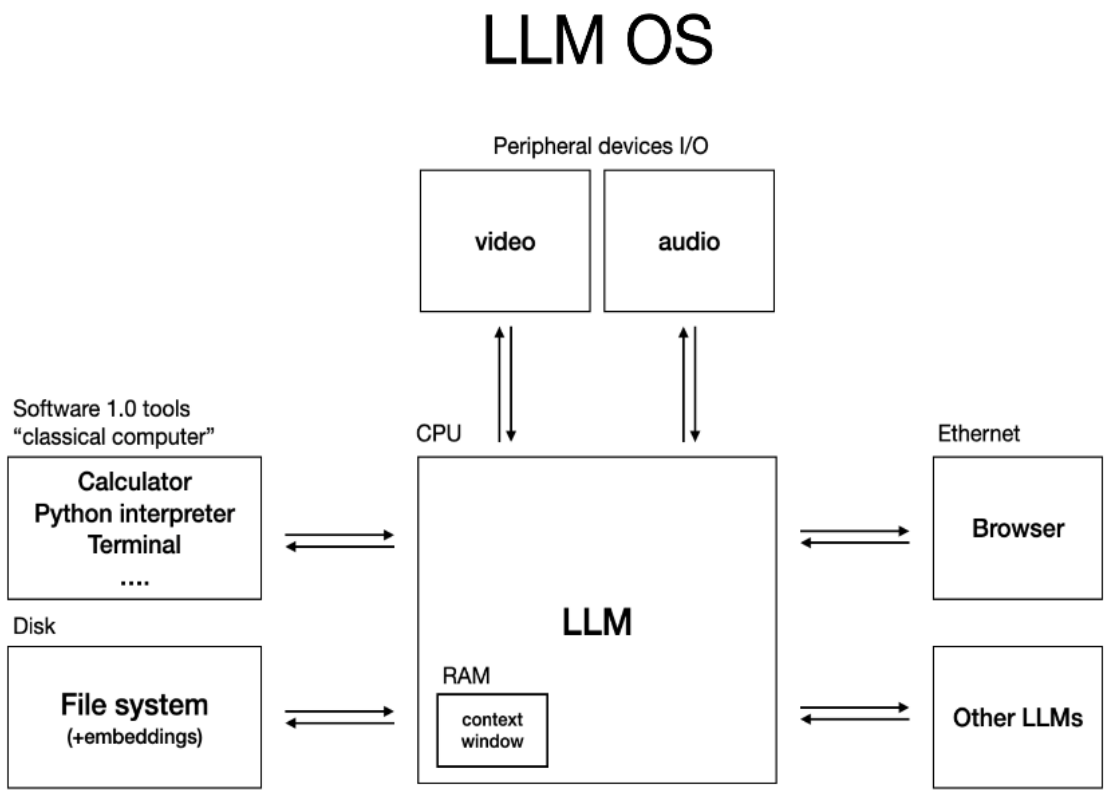
From Context Engineering to Code Context Engineering

"Context engineering is the delicate art of filling the context window with just the right information for the next step."

— Andrej Karpathy, June 2025

For coding agents, this question becomes empirical:

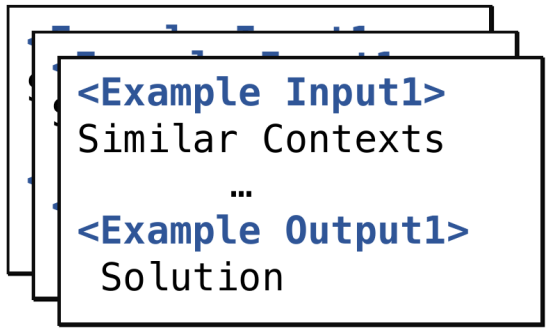
- What code does the agent decide to read?
- How much of it is actually useful?
- Does better context change the patch?



Three Paradigms of Code Context Engineering

Similarity-based ICL

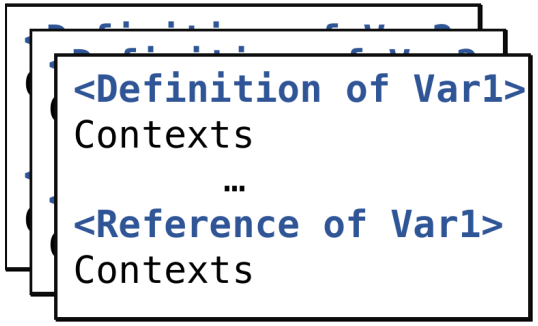
Learn from Example



- A simple baseline
- Similarity is not dependency
- Often brings extra noise

Static Analysis

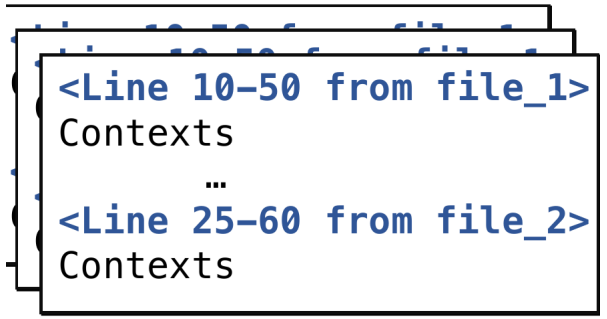
Learn from Dependency



- Most stable passive method
- Follows program structure
- Low cost and predictable

Navigation Agent

Learn by Exploration



- Higher gain on hard tasks
- Depends on tools and model
- Can recover usage evidence

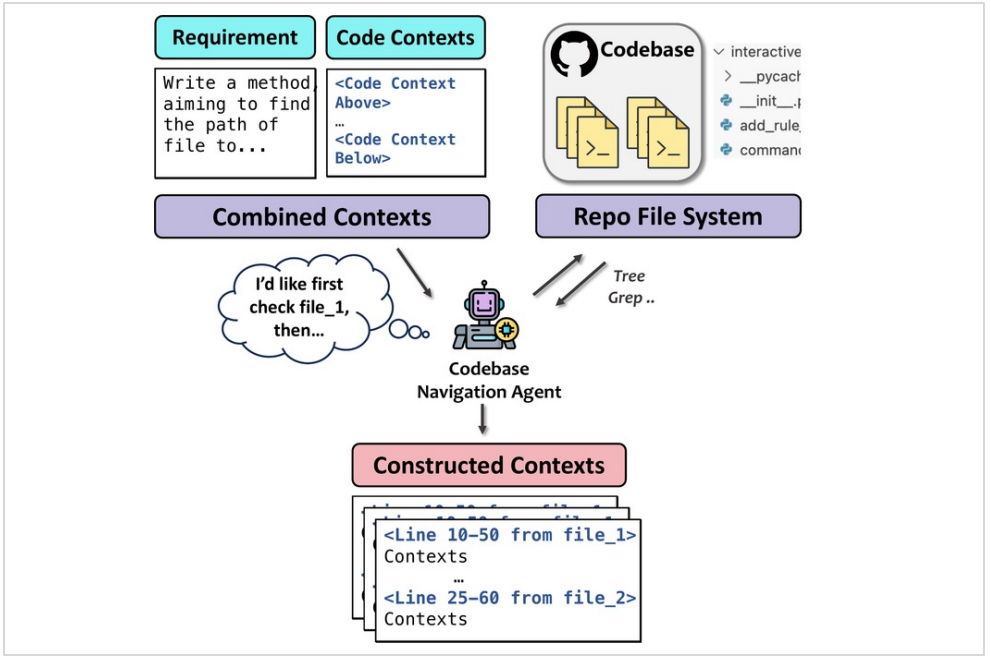
The choice depends on what kind of context the task needs.

What Was Still Unclear?

Many coding systems combine retrieval, tools, prompts, and model choice.

We had many methods, but no clean comparison.

Most evaluations look only at final pass rate. They do not show whether the method found the code evidence it needed.



So this paper studies the context step itself.

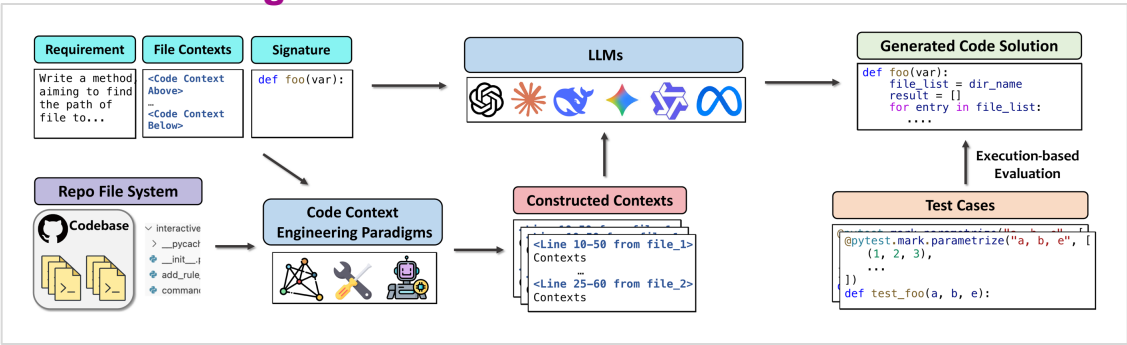
Study Design: Same Task, Different Context

We keep the task and generator fixed, and only change what the model gets to read.

Evaluation scope

Task corpus	CoderEval DevEval RepoClassBench
Methods	Methods from similarity retrieval, static analysis, and navigation
Models	Modern LLMs; generator held fixed within each comparison
Measures	Effectiveness, efficiency, cost, and context quality

What changes across runs



Each method gives the model a different view of the same repository.

If performance changes, the context method is the reason.

Our Contribution: A Code Context Engineering Evaluation Framework

We evaluate the context acquisition process, not just the final patch.

Outcome

Does the patch pass the tests?

Pass@1 remains the end-to-end signal for code generation.

It tells us whether the final answer works.

Cost

How expensive was the context?

Rounds, latency, calls, and tokens determine whether a method fits the workflow.

Exploration is useful only when its cost is justified.

Context evidence

What useful code arrived?

DCR checks whether dependencies were collected; composition checks what kind of evidence they provided.

Coverage and evidence type have to be read together.

This lets us compare context strategies by correctness, cost, and evidence.

RQ1

Effectiveness

Which paradigm works best, and when?

RQ1: Function-Level Generation

Table 6. *Pass@1* results for different context acquisition methods on the DevEval, CoderEval, and RepoClass-Bench datasets (RQ1). The *Avg-Improve* column shows the averaged relative improvement over the *Base* performance. The greener the cell, the better the performance on that dataset.

Dataset	Method	L-3.3	Q-3	QC-3	G-O	DS-R1	GPT-4o	C-S-4	G-P-2.5	Avg-Improve
DevEval (Infilling)	Base	54.78	61.74	66.09	58.26	65.22	59.13	66.09	67.83	—
	ICL_{BM25}	56.52	63.48	70.43	62.61	65.22	62.61	68.7	67.83	3.73%↑
	$ICL_{UniXcoder}$	57.39	64.35	69.57	60.0	64.35	61.74	66.96	71.3	3.34%↑
	$ICL_{CoCoSoDa}$	55.65	66.09	66.96	63.48	64.35	60.87	71.3	68.7	3.71%↑
	RepoCoder	57.39	65.22	70.43	65.22	64.35	61.74	69.57	68.7	4.87%↑
	SA_{Method}	60.0	66.96	69.57	67.83	70.43	60.0	70.43	69.57	7.28%↑
	SA_{File}	62.61	69.57	70.43	69.57	71.3	61.74	73.04	71.3	10.29%↑
	A3-CodGen	61.74	67.83	70.43	68.7	70.43	60.87	70.43	69.57	8.19%↑
	Nav_{Raw}	55.65	65.22	67.83	65.22	66.96	64.35	73.04	73.04	6.98%↑
	Nav_{Plus}	56.52	68.7	71.3	67.83	68.7	66.96	73.91	73.91	9.77%↑
	SWE-Agent	54.78	67.83	68.7	66.96	67.83	65.22	73.91	73.04	7.82%↑
CoderEval (Infilling)	Base	41.74	47.83	46.96	48.26	44.35	42.61	49.56	52.17	-
	ICL_{BM25}	40.43	48.7	49.13	51.74	49.57	46.96	52.61	56.09	5.77%↑
	$ICL_{UniXcoder}$	43.04	48.26	52.61	53.04	50.87	45.65	53.91	57.83	8.43%↑
	$ICL_{CoCoSoDa}$	42.17	48.26	52.61	53.48	50.87	46.96	51.74	56.52	7.80%↑
	RepoCoder	40.87	50.0	50.87	52.17	50.87	47.39	53.91	56.96	7.85%↑
	SA_{Method}	52.17	54.35	56.52	54.78	53.91	49.13	56.96	60.43	17.51%↑
	SA_{File}	53.91	55.22	56.09	55.22	54.35	52.17	58.26	58.7	19.19%↑
	A3-CodGen	53.48	54.78	57.39	55.22	53.91	50.0	57.83	60.87	18.95%↑
	Nav_{Raw}	35.22	52.61	53.91	50.87	50.0	57.83	63.04	60.87	14.17%↑
	Nav_{Plus}	47.39	54.35	53.91	55.22	52.6	56.09	64.78	62.61	17.17%↑
	SWE-Agent	38.26	53.48	53.91	53.04	51.3	56.96	63.91	61.74	15.61%↑
RepoClassBench (Class-level)	Base	24.47	29.79	30.85	26.60	31.91	27.66	30.85	32.98	—
	ICL_{BM25}	23.40	30.85	32.98	27.66	35.11	29.79	35.11	38.30	7.22%↑
	$ICL_{UniXcoder}$	23.40	29.79	32.98	26.60	31.91	27.66	30.85	38.30	2.34%↑
	$ICL_{CoCoSoDa}$	23.40	29.79	32.98	26.60	30.85	28.72	38.30	36.17	4.61%↑
	RepoCoder	-	-	-	-	-	-	-	-	-
	SA_{Method}	-	-	-	-	-	-	-	-	-
	SA_{File}	26.60	32.98	35.11	29.79	35.11	30.85	37.23	37.23	12.54%↑
	A3-CodGen	-	-	-	-	-	-	-	-	-
	Nav_{Raw}	19.15	35.11	31.91	28.72	37.23	34.04	43.62	42.55	14.72%↑
	Nav_{Plus}	21.28	39.36	37.23	31.91	40.43	39.36	45.74	43.62	26.16%↑
	SWE-Agent	20.21	31.91	35.11	29.79	36.17	37.23	44.68	42.55	17.17%↑

‡ RepoCoder, SA_{Method}, and A3-CodGen are not applicable to class-level tasks (see text).

Function-level tasks

Static analysis is the reliable default.

On DevEval and CoderEval, dependency-aware context gives consistent gains across models.

Red boxes mark the SA variants in Table 6.

RQ1: Class-Level Generation

Table 6. *Pass@1* results for different context acquisition methods on the DevEval, CoderEval, and RepoClass-Bench datasets (RQ1). The *Avg-Improve* column shows the averaged relative improvement over the *Base* performance. The greener the cell, the better the performance on that dataset.

Dataset	Method	L-3.3	Q-3	QC-3	G-O	DS-R1	GPT-4o	C-S-4	G-P-2.5	Avg-Improve
DevEval (Infilling)	Base	54.78	61.74	66.09	58.26	65.22	59.13	66.09	67.83	—
	ICL_{BM25}	56.52	63.48	70.43	62.61	65.22	62.61	68.7	67.83	3.73%↑
	$ICL_{UniXcoder}$	57.39	64.35	69.57	60.0	64.35	61.74	66.96	71.3	3.34%↑
	$ICL_{CoCoSoDa}$	55.65	66.09	66.96	63.48	64.35	60.87	71.3	68.7	3.71%↑
	RepoCoder	57.39	65.22	70.43	63.48	64.35	64.35	69.57	68.7	4.87%↑
	SA_{Method}	60.0	66.96	69.57	67.83	70.43	60.0	70.43	69.57	7.28%↑
	SA_{File}	62.61	69.57	70.43	69.57	71.3	61.74	73.04	71.3	10.29%↑
	A3-CodGen	61.74	67.83	70.43	68.7	70.43	60.87	70.43	69.57	8.19%↑
	Nav_{Raw}	55.65	65.22	67.83	65.22	66.96	64.35	73.04	73.04	6.98%↑
	Nav_{Plus}	56.52	68.7	71.3	67.83	68.7	66.96	73.91	73.91	9.77%↑
	SWE-Agent	54.78	67.83	68.7	66.96	67.83	65.22	73.91	73.04	7.82%↑
CoderEval (Infilling)	Base	41.74	47.83	46.96	48.26	44.35	42.61	49.56	52.17	-
	ICL_{BM25}	40.43	48.7	49.13	51.74	49.57	46.96	52.61	56.09	5.77%↑
	$ICL_{UniXcoder}$	43.04	48.26	52.61	53.04	50.87	45.65	53.91	57.83	8.43%↑
	$ICL_{CoCoSoDa}$	42.17	48.26	52.61	53.48	50.87	46.96	51.74	56.52	7.80%↑
	RepoCoder	40.87	50.0	50.87	52.17	50.87	47.39	53.91	56.96	7.85%↑
	SA_{Method}	52.17	54.35	56.52	54.78	53.91	49.13	56.96	60.43	17.51%↑
	SA_{File}	53.91	55.22	56.09	55.22	54.35	52.17	58.26	58.7	19.19%↑
	A3-CodGen	53.48	54.78	57.39	55.22	53.91	50.0	57.83	60.87	18.95%↑
	Nav_{Raw}	35.22	52.61	53.91	50.87	50.0	57.83	63.04	60.87	14.47%↑
	Nav_{Plus}	47.39	54.35	53.91	55.22	52.6	56.09	64.78	62.61	17.17%↑
	SWE-Agent	38.26	53.48	53.91	53.04	51.3	56.96	63.91	61.74	15.61%↑
RepoClassBench (Class-level)	Base	24.47	29.79	30.85	26.60	31.91	27.66	30.85	32.98	—
	ICL_{BM25}	23.40	30.85	32.98	27.66	35.11	29.79	35.11	38.30	7.22%↑
	$ICL_{UniXcoder}$	23.40	29.79	32.98	26.60	31.91	27.66	30.85	38.30	2.34%↑
	$ICL_{CoCoSoDa}$	23.40	29.79	32.98	26.60	30.85	28.72	38.30	36.17	4.61%↑
	RepoCoder	-	-	-	-	-	-	-	-	-
	SA_{Method}	-	-	-	-	-	-	-	-	-
	SA_{File}	26.60	32.98	35.11	29.79	35.11	30.85	37.23	37.23	12.34%↑
	A3-CodGen	-	-	-	-	-	-	-	-	-
	Nav_{Raw}	19.15	35.11	31.91	28.72	37.23	34.04	43.62	42.55	14.72%↑
	Nav_{Plus}	21.28	39.36	37.23	31.91	40.43	39.36	45.74	43.62	26.16%↑
	SWE-Agent	20.21	31.91	35.11	29.79	36.17	37.23	44.68	42.55	17.17%↑

‡ RepoCoder, SA_{Method}, and A3-CodGen are not applicable to class-level tasks (see text).

Class-level tasks

Navigation gains more room on full-class generation.

When the target is a full class, file-level static cues weaken and exploration becomes more useful.

Red boxes mark the class-level navigation rows.

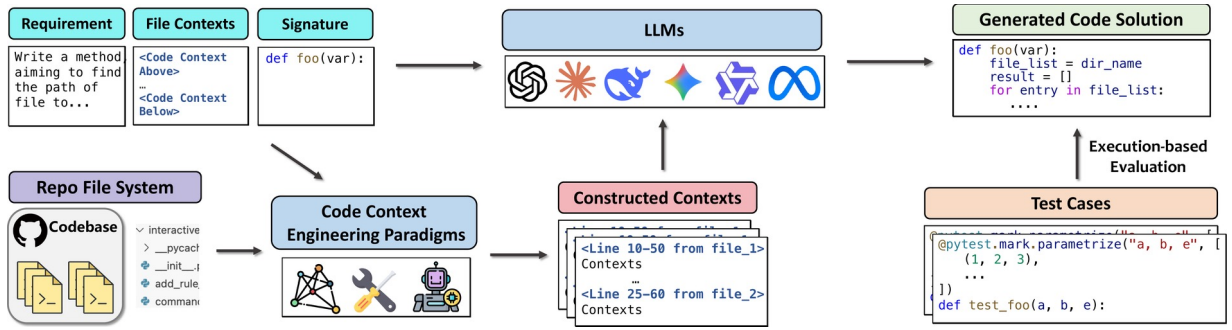
Which Paradigm Works Best?

Task granularity changes which context strategy works best.

Function-level generation

Static analysis usually has enough structure to work well.

For a missing function, sibling code and imports often expose the relevant dependencies.



Class-level and complex tasks

Navigation becomes useful when local cues disappear.

Exploration can recover usage patterns and cross-file behavior beyond the target file.

RQ1 should be read by task granularity.

Function-level and class-level results tell different stories, so one aggregate score would hide the main effect.

RQ2

Efficiency and Cost

What do we pay for extra exploration?

RQ2: Time Efficiency

Metric guide: Offline prep is one-time repo setup; online prep and LLM calls are spent per task.

Table 7. Time efficiency on DevEval. Offline costs are one-time per repository. Online costs are per-task. Navigation costs (*marked with* *) are averaged across all evaluated language models, as they are model-dependent.

Method Paradigm		Offline Phase	Online Phase (Per-Task)	
		$T_{\text{offline\_prep}}$ (ms)	$T_{\text{online\_prep}}$ (ms)	N_{llm} (# Invocations)
Base	—	N/A	≈ 0	1
Similarity-based ICL	ICL_{BM25}	133	11	1
	$\text{ICL}_{\text{UniXcoder}}$	7811	56	1
	$\text{ICL}_{\text{CoCoSoDa}}$	5175	47	1
	RepoCoder	487	127	2
Static Analysis	$\text{SA}_{\text{Method}}$	3011	383	1
	SA_{File}	2756	269	1
	A3-CodGen	4687	421	1
Navigation	Nav_{Raw}	N/A	1426*	9.85*
	Nav_{Plus}	9380	2075*	4.62*
	SWE-Agent	N/A	1386*	11.47*

RQ2: Token Cost Efficiency

Table 8. Model cost efficiency analysis (Logarithmic Scale Heatmap). For RAG and Static Analysis, the token count is model-agnostic. For Navigation, the token count is model-dependent. Costs are estimated based on public pricing models and their own tokenizers.

Paradigm		Avg. Total Tokens	Averaged Cost per Task ($)							
			L-3.3	Q-3	QC-3	G-O	DS-R1	GPT-4o	C-S-4	G-P-2.5
Base	—	4,977	.0009	.0011	.0012	.0007	.0028	.0113	.0395	.0332
Similarity-based ICL	ICL_{BM25}	28,412	.0059	.0065	.0069	.0074	.0104	.0647	.1129	.0802
	$ICL_{UniXcoder}$	29,790	.0064	.0072	.0076	.0081	.0114	.0711	.1239	.0835
	$ICL_{CoCoSoDa}$	28,581	.0058	.0071	.0069	.0074	.0102	.0647	.1160	.0793
	RepoCoder	33,247	.0067	.0077	.0080	.0082	.0131	.0762	.1519	.1131
Static Analysis	SA_{Method}	8,195	.0016	.0020	.0020	.0026	.0053	.0253	.0539	.0455
	A3-CodGen	9,695	.0019	.0024	.0024	.0035	.0065	.0318	.0606	.0512
	SA_{File}	43,497	.0084	.0107	.0095	.0112	.0216	.0958	.2506	.1787
Navigation	Nav_{Raw}	92,147	.0161	.0385	.0281	.0314	.1272	.3184	.7136	.4158
	SWE-Agent	82,463	.0168	.0361	.0254	.0296	.1095	.2743	.6483	.3651
	Nav_{Plus}	56,785	.0107	.0274	.0196	.0240	.0539	.1972	.5295	.2619

[a] A deeper red color indicates a higher cost.

Context Choice Depends on Workflow

The results point to different choices in different workflows.

Synchronous IDE

Prefer predictable context.

Static analysis gives stable context with low latency. It is the safer default when the user is waiting.

Background agent

Spend exploration only when it pays off.

Navigation can reach a higher ceiling, but the extra calls and tokens belong in slower, deeper workflows.

Tool design

Better tools change what the agent reads.

Definition and reference queries make navigation more structured, closer to how developers inspect code.

Accuracy, latency, and cost have to be chosen together.

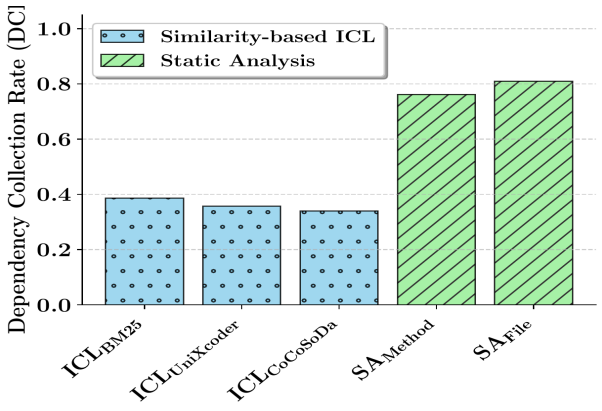
RQ3

Context Quality

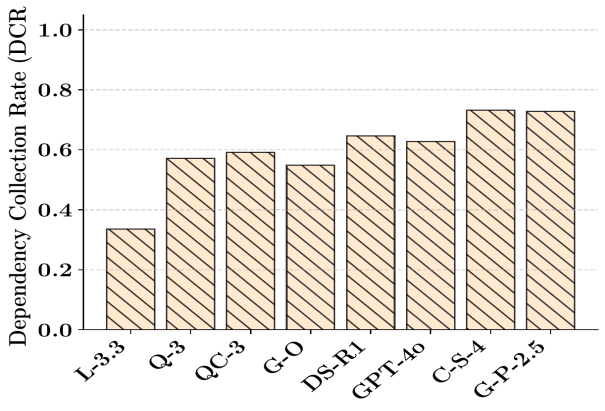
Did the method actually collect useful context?

RQ3: Dependency Collection

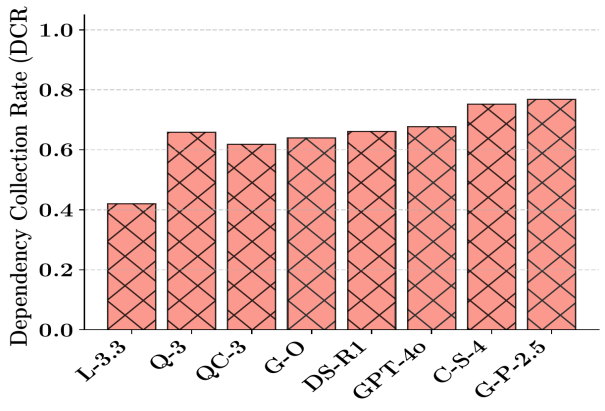
DCR checks whether the method collected the dependencies needed for the task.



Passive methods



Nav Raw



Nav Plus

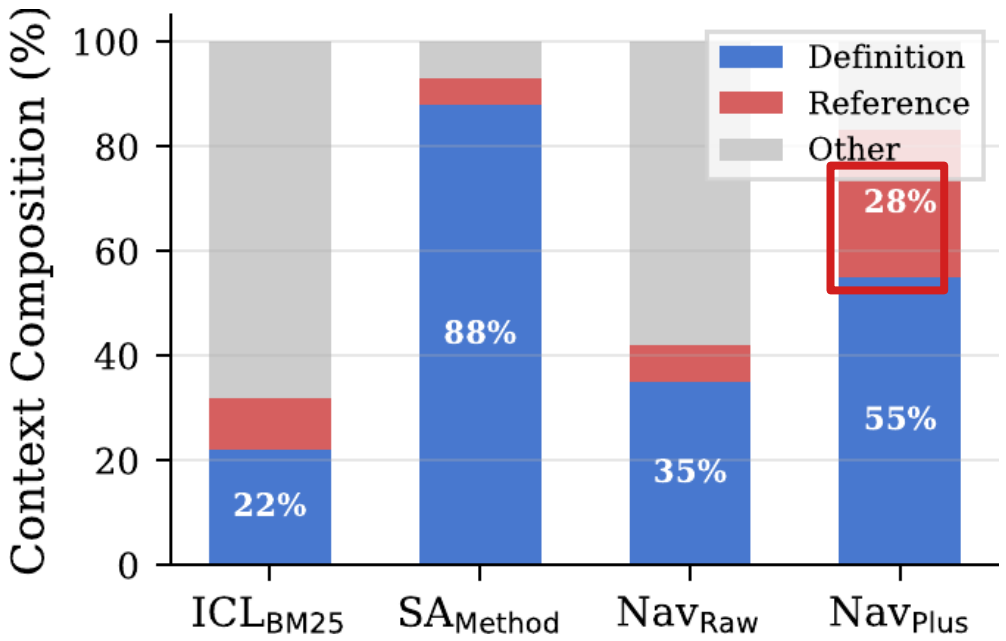
Finding: Static analysis and Nav Plus collect dependencies reliably. Similarity retrieval still struggles to find the right code.

The Paradox: Lower Coverage, Higher Performance?

Nav Plus can improve performance even when its DCR is not the highest.

DCR measures whether dependencies are collected.
It does not say whether the context contains usage evidence.

The gap is explained by context composition: definitions plus references, not volume alone.

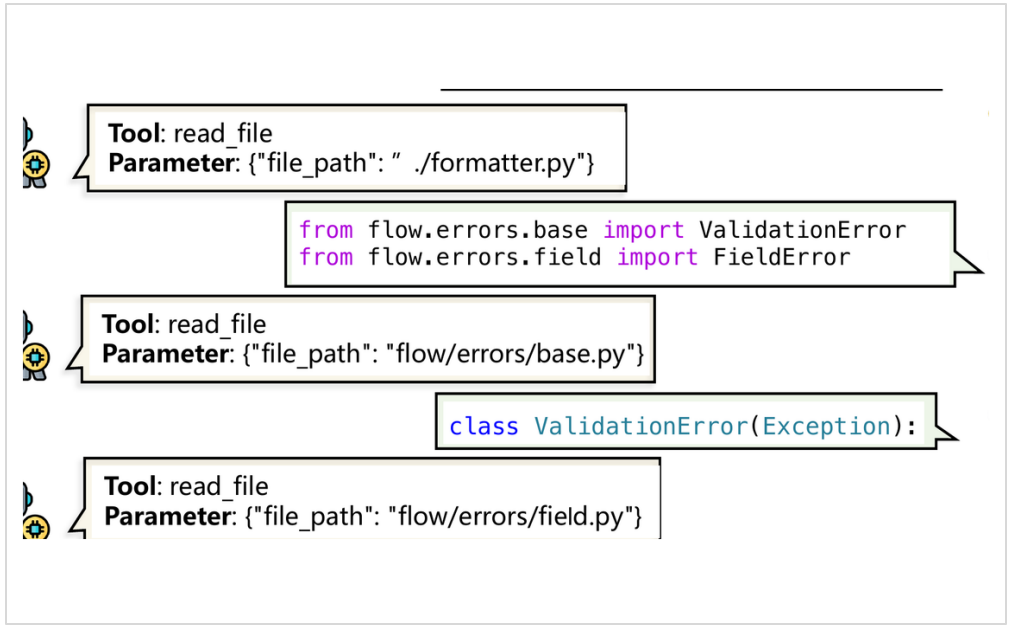


Paper figure: context composition by paradigm.

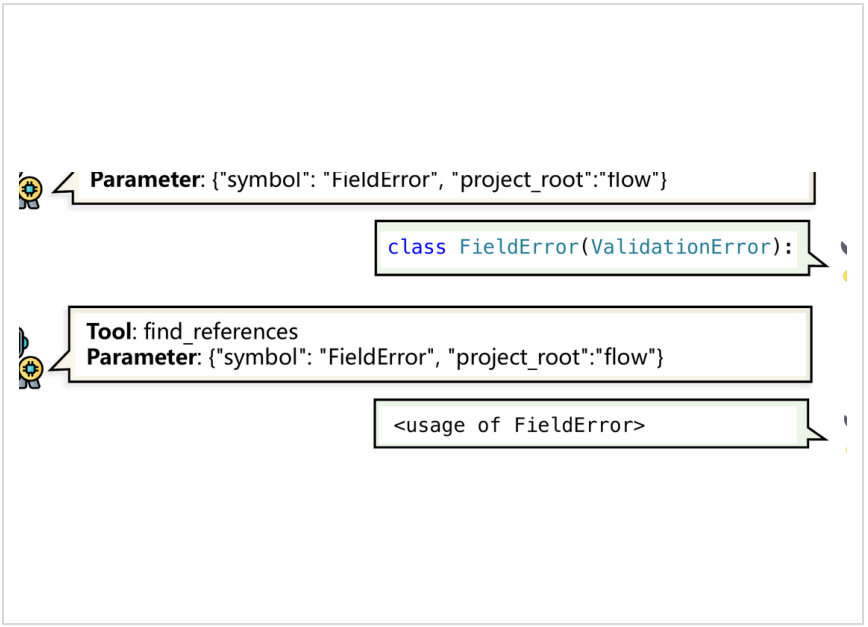
Coverage tells us whether the right code arrived. Composition tells us what kind of code arrived.

Case Study: Stronger Tools Change the Context

On the same task, basic tools read files; stronger tools jump to references.



Nav Raw: read tree -> read files -> still missing usage pattern



Nav Plus: semantic search -> definition -> references

Many reads

Three focused steps

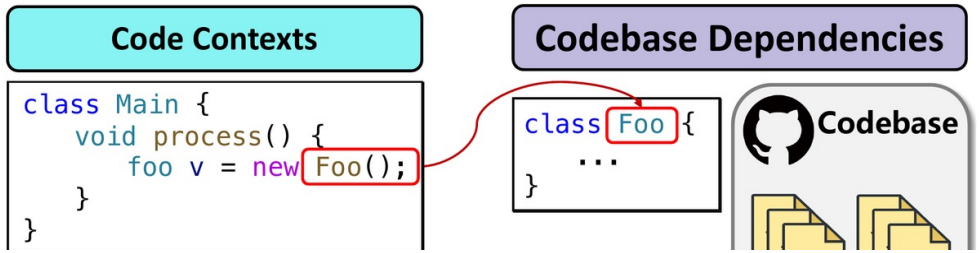
Reference search exposes the usage pattern that definitions do not show.

Finding 3: Evaluate Coverage and Composition

Context quality has two complementary signals.

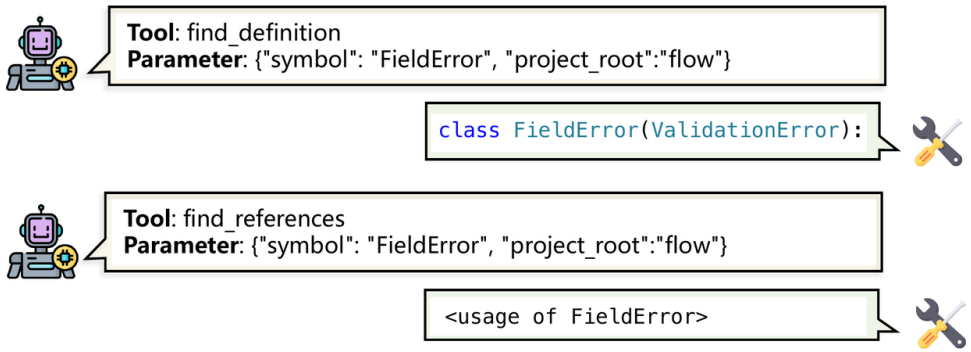
Definition coverage

Definitions tell the model what an entity is.



Reference coverage

References show how the entity is used.



Coverage tells us whether the right code arrived. Composition separates definitions from usage evidence.

Lessons Learned

Match Paradigm to Task

Static analysis is a practical default for local, fast assistance.

Navigation is worth the cost for complex tasks.

Design Better Tools

Structural tools reduce aimless exploration.

Better tools create cleaner context and cleaner trajectories.

Evaluate the Context

Pass@1 is not enough.

DCR and composition explain what the agent actually collected.

Context acquisition should be selected, measured, and engineered.

Takeaway

Coding agents still need someone to decide what they should read.

Static analysis gives a reliable low-cost default.

Navigation helps when the task needs exploration.

Future coding agents should retrieve both definitions and references.

Code, data, and paper

香港中文大學
The Chinese University of Hong Kong