Remember Your Trace: Memory-Guided Long-Horizon Agentic Framework for Consistent and Hierarchical Repository-Level Code Documentation

Suyoung Bae[1], Jaehoon Lee[1], Changkyu Choi[2], YunSeok Choi[1], Jee-Hyong Lee[1]
[1] Sungkyunkwan University, South Korea
[2] University of Oslo, Norway
NeurIPS 2026

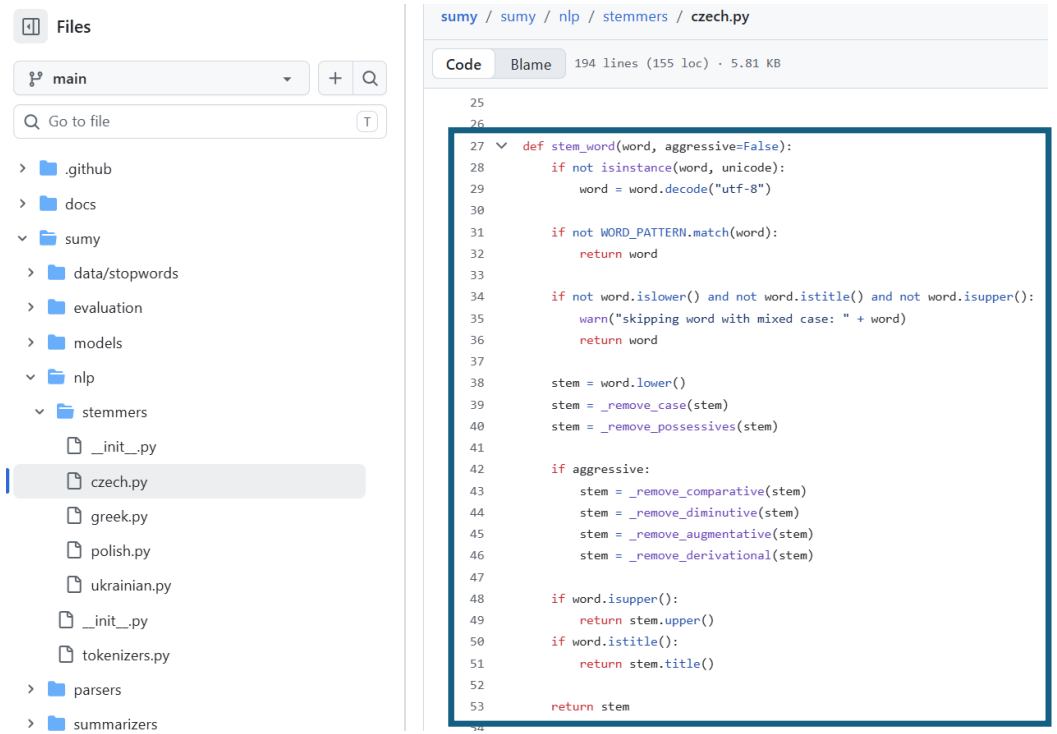
Presenter: Suyoung Bae
Sungkyunkwan University
sybae01@skku.edu

Content

- Introduction
- Proposed Method
- Experimental Results
- Conclusion

Introduction

- Repository-Level Code Documentation is a **long and descriptive summaries for functions, methods, or classes within a repository**,
- where each target entity is documented not only as a standalone unit but also in terms of its **inter-relations with other files, functions, and external information across the repository**.



Document the “stem_word” function.

Description: This function performs Czech word stemming with optional aggressive morphological processing. This function serves as the main entry point for Czech text stemming, applying basic case removal and possessive ending stripping by default, …

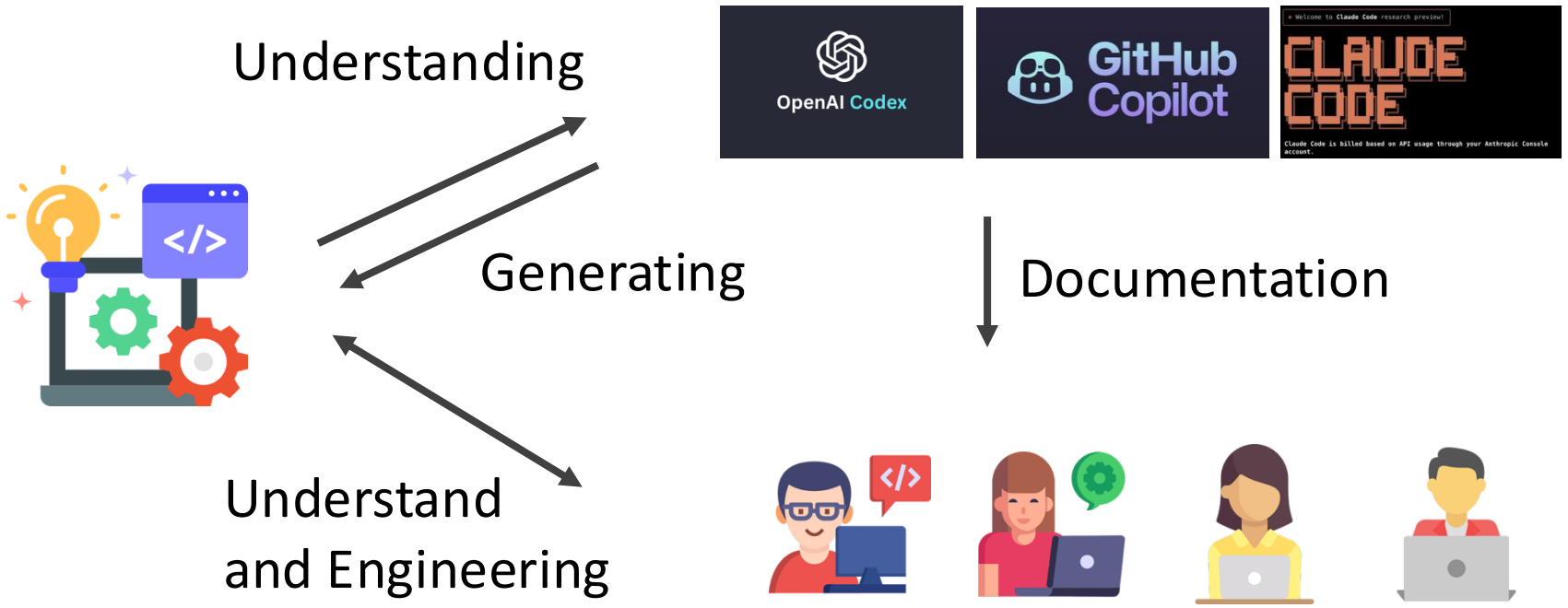
Args: word(str) A Czech word to be stemmed, which …

Returns: str: The stemmed version of the input word, …

Raises: None explicitly raised, though mixed-case …

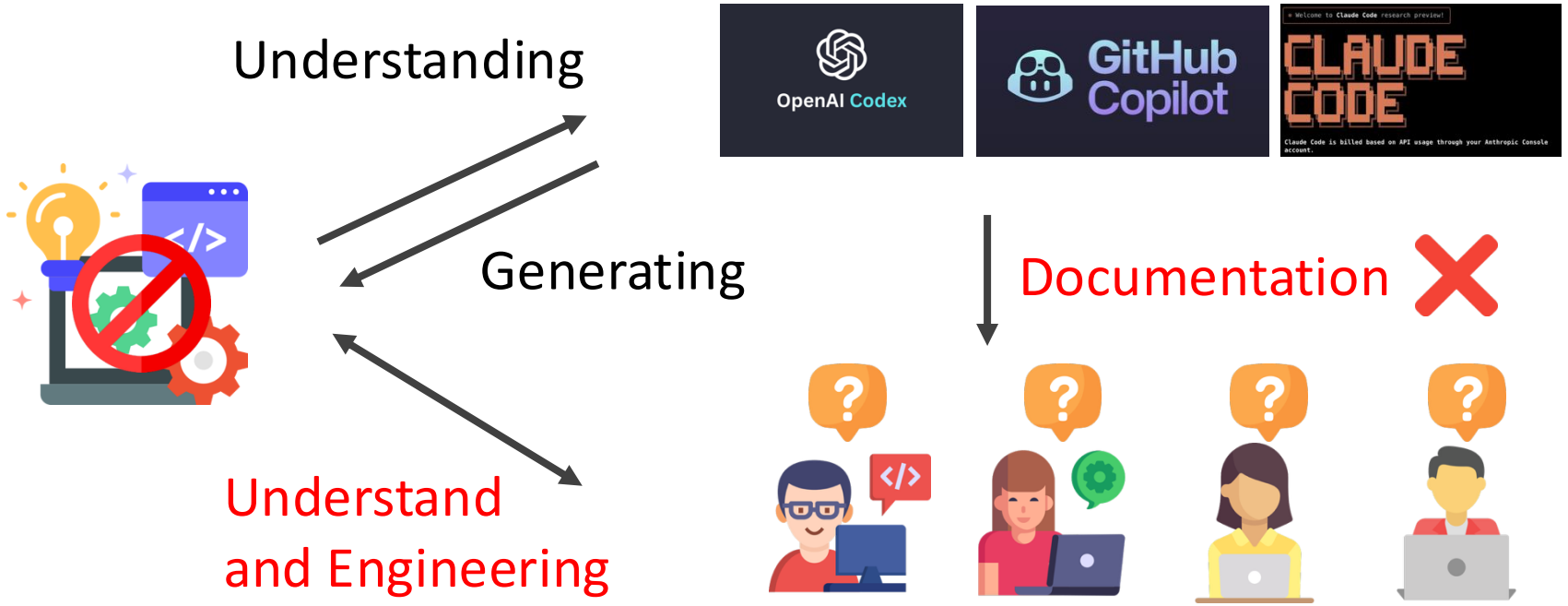
Introduction (Cont.)

- **Repository-level code documentation** are increasingly integrated into real-world developmen t workflows to assist engineers in understanding and reviewing code.



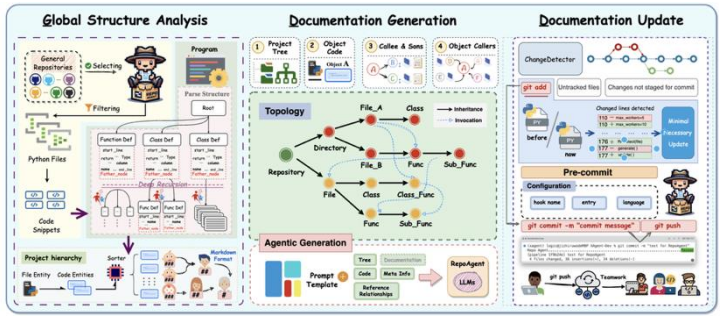
Introduction (Cont.)

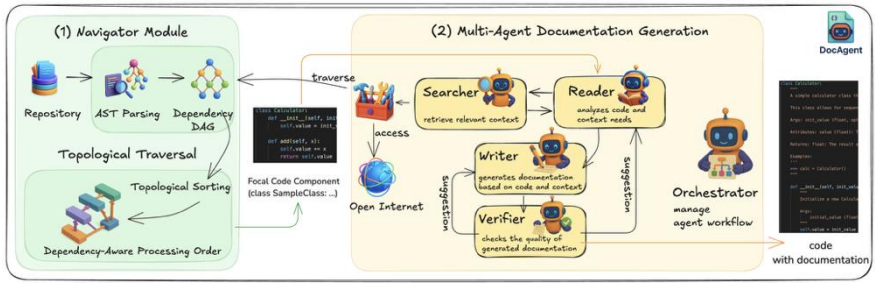
- However, when the generated summary does not accurately reflect the code's actual impleme ntation, it can cause developers to **misunderstand the code**, leading to **delayed debugging** an d **increased maintenance costs.**

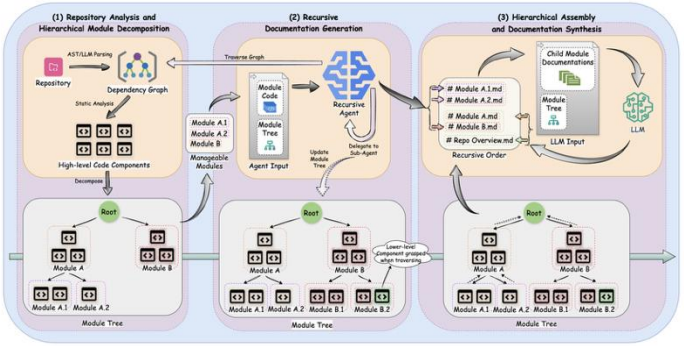


Previous Method

- Mostly on Multi-Agent based framework
  - [1] RepoAgent: An LLM-Powered Open-Source Framework for Repository-level Code Documentation Generation (EMNLP 2024)
  - [2] DocAgent: A Multi-Agent System for Automated Code Documentation Generation (ACL 2025)
  - [3] CodeWiki: Automated Repository-Level Documentation at Scale (2025 Arxiv)







Previous Method (Cont.)

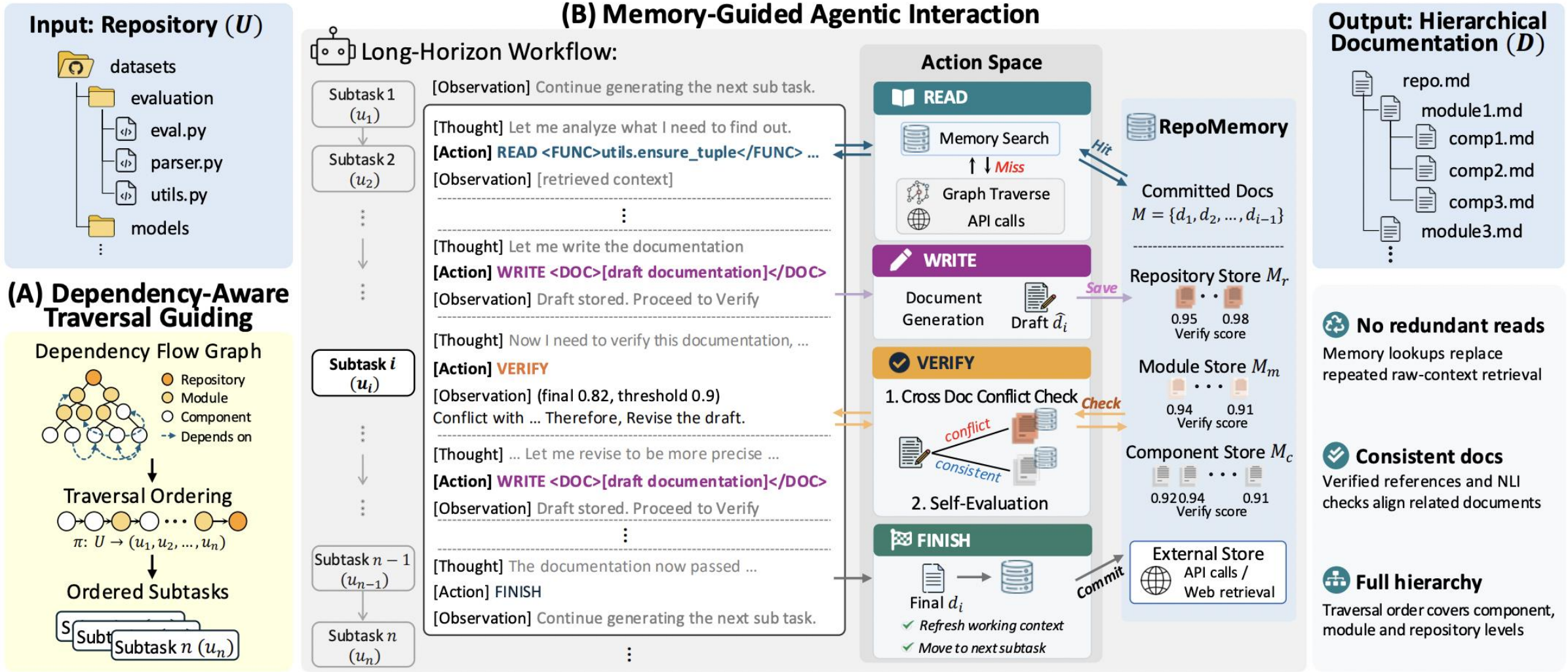
- Limitation1 : Treating each as a self-contained task without shared state across the repository
  - Lack Repository-level consistency :
    - Multi-Agents operate in isolate context windows with no awareness of how other agents have expressed the same concept.
    - Existing systems lack a shared memory mechanism across agent boundaries. When a new component is assigned, the agent has no access to the accumulated state of prior documentation decisions.
  - Redundant dependency retrieval :
    - Prior retrieval results are never shared, every component searches from scratch
    - Each component independently re-fetches the same dependencies, incurring redundant retrieval cost
    - Docstrings generated in earlier steps are not reused as context in subsequent steps — a structural disconnect across the trajectory
- Limitation2 : Lack hierarchical structure in their documentation outputs

Proposed Method

- We therefore formulate repository-level documentation as a **long-horizon agentic problem**
  - Solution 1. Single long horizon agentic workflow
    - Unified context : A single agent accumulates all prior documentation in working memory, ensuring global coherence
    - Long-horizon planning : Topological traversal with dependency-aware ordering maintains causal consistency across components
    - Adaptive replanning : Agent revises plans mid-trajectory based on intermediate outputs and discovered dependencies
  - Solution 2. Memory-guided selective retrieval
    - **Write** after each component -> accumulated documentation memory
    - **Recall** before search -> check if dependency already in memory -> skip redundant search
    - **Update** cross-document conflict

See 4.2 in our paper

Proposed Method (Cont.)



See 5.1 in our paper

Proposed Method (Cont.)

- Evaluation Framework
  1. **Completeness** : Whether the documentation contains all structurally required sections and mentions all source-level entities that a developer needs to understand the component without reading source code
     - Section Presence Score, Entity Coverage Score
  2. **Truthfulness** : Verifies the factual accuracy of documentation by extracting atomic claims and checking each against the actual source code, scored as the ratio of supported claims to supported plus contradicted claims
     - Consistency Score, Relevance Score
  3. **Helpfulness** : How effectively the documentation enables a developer to make correct decisions about using the component
  4. **Information Efficiency** : Whether an LLM can reconstruct a correct function implementation using only the generated documentation

See 5.3 in our paper

Proposed Method (Cont.)

- Benchmarks
  - 20 Python repositories from DevEval dataset

Table 20: Repository statistics across 20 repositories spanning 10 domains. *#Test fn.* and *#Test cases* indicate the number of component-level functions covered by human-curated unit tests and the total number of test cases collected from DevEval [69], respectively.

#	Domain	Repository	Files	Classes	Methods	Functions	Components	#Test fn.	#Test cases
1	Communications	IMAPClient	28	13	36	35	84	33	71
2		Wikipedia-API	4	5	47	7	59	13	46
3	Database	bplustree	10	1	30	2	33	19	27
4		datasette	42	53	174	98	325	42	59
5	Internet	Jinja2	37	22	84	63	169	17	40
6		sumy	44	34	209	67	310	30	65
7	Multimedia	hypertools	92	11	45	52	108	11	16
8		mingus	43	56	373	240	669	52	60
9	Scientific-Engineering	csvkit	25	28	107	31	166	4	12
10		folium	47	47	74	70	191	3	9
11	Security	zxcvbn-python	9	1	1	40	42	18	25
12		trailscraper	14	9	43	35	87	11	28
13	Software-Development	PySnooper	8	10	37	17	64	3	3
14		ydata-profiling	194	125	209	342	676	17	25
15	System	exodus-bundler	10	15	35	26	76	16	23
16		flower	29	52	117	38	207	4	15
17	Text-Processing	parsel	8	11	49	19	79	3	3
18		online-judge-tools	20	13	17	58	88	3	13
19	Utilities	mackup	9	6	32	17	55	7	11
20		PyJWT	17	24	54	28	106	6	13
Total			**690**	**536**	**1,773**	**1,285**	**3,594**	**312**	**564**

See 6.1 in our paper

Experimental Results

Table 2: Main comparison results. (a) Overall results on average completeness, truthfulness, and helpfulness and (b) the same metrics separated by documentation granularity. Within each backbone group, the best result is **bold** and the second-best underlined. * marks in section scores set to 1.0 by default, as their internal format instruction is inaccessible. "–" denotes non-applicable granularities.

(a) Overall results on average completeness, truthfulness, and helpfulness.

Backbone	Method	#Doc	Completeness			Truthfulness			Helpfulness
			Section Score	Coverage Score	Complete.	Consistency	Relevance	Truthful.	
Open-Source Baselines									
Qwen3-Coder	Prompting	3,323	0.838	0.837	0.838	0.809	0.803	0.649	0.458
	RepoAgent	3,650	0.659	0.967	0.813	0.910	0.918	0.835	0.391
	DocAgent	3,198	0.792	0.898	0.845	0.900	0.889	0.800	0.556
	CodeWiki	80	0.770	0.473	0.622	0.906	0.852	0.772	0.628
	MemDocAgent	3,323	**0.959**	**0.998**	**0.979**	**0.968**	**0.946**	**0.916**	**0.690**
GPT-5-mini	Prompting	3,323	0.790	0.473	0.632	0.883	0.840	0.741	0.617
	RepoAgent	3,650	0.555	**1.000**	0.778	0.898	0.947	0.850	0.449
	DocAgent	3,198	0.776	0.944	0.860	0.898	0.925	0.831	0.671
	CodeWiki	80	**0.975**	0.579	0.777	0.847	0.771	0.653	0.708
	MemDocAgent	3,323	0.916	**1.000**	**0.958**	**0.983**	**0.968**	**0.952**	**0.800**
Closed-Source Baselines									
Claude-Code (`/init`)		20	1.000*	0.488	0.744	0.845	0.869	0.734	0.696
DeepWiki		404	1.000*	0.840	0.920	0.815	0.935	0.762	0.755

(b) Results separated by documentation granularity (component-, module-, and repository-level).

Backbone	Method	Component-Level			Module-Level			Repository-Level		
		Complete.	Truthful.	Helpful.	Complete.	Truthful.	Helpful.	Complete.	Truthful.	Helpful.
Open-Source Baselines										
Qwen3-Coder	Prompting	0.886	0.694	0.577	0.870	0.666	0.313	0.758	0.587	0.483
	RepoAgent	0.813	0.835	0.391	–	–	–	–	–	–
	DocAgent	0.845	0.800	0.556	–	–	–	–	–	–
	CodeWiki	–	–	–	0.630	0.769	0.606	0.613	0.775	0.650
	MemDocAgent	**0.987**	**0.962**	**0.691**	**0.975**	**0.902**	**0.702**	**0.975**	**0.883**	**0.677**
GPT-5-mini	Prompting	0.631	0.761	0.627	0.626	0.766	0.626	0.640	0.696	0.598
	RepoAgent	0.778	0.850	0.449	–	–	–	–	–	–
	DocAgent	0.860	0.831	0.671	–	–	–	–	–	–
	CodeWiki	–	–	–	0.809	0.597	0.737	0.744	0.708	0.679
	MemDocAgent	**0.952**	**0.974**	**0.750**	**0.974**	**0.924**	**0.858**	**0.949**	**0.959**	**0.791**

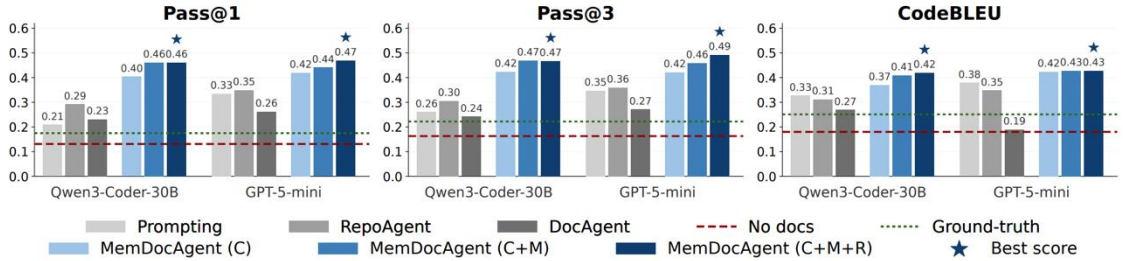


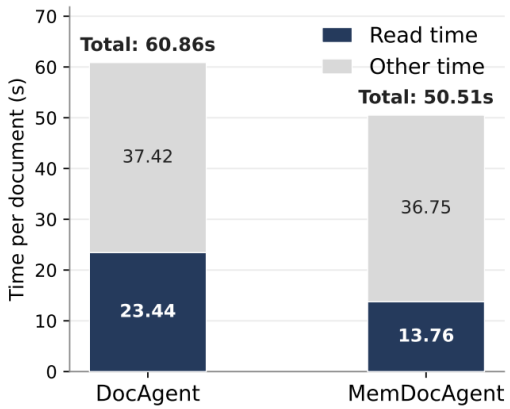
Figure 3: Information sufficiency comparison results. Pass@1, @3 and CodeBLEU when regenerating code from each method's documentation alone, averaged over two code generators, with the x-axis denoting the backbone used for each open-source baseline.

See 6.2, 6.3, and 6.5 in our paper

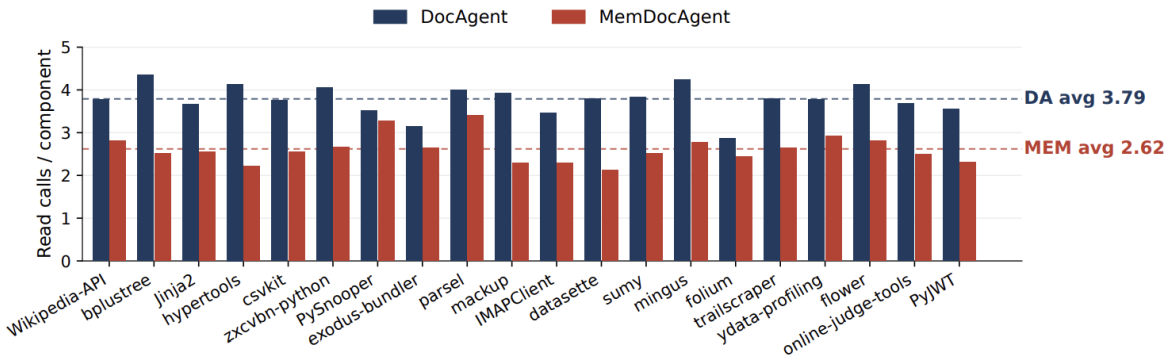
Experimental Results (Cont.)

Table 3: Ablation study on key components of MemDocAgent. All ablations are conducted under the Qwen3-Coder backbone. Values in parentheses indicate the drop in percentage points relative to ours.

	Documentation Quality			Information Sufficiency		
Method	Completeness	Truthfulness	Helpfulness	Pass@1	Pass@3	CodeBLEU
MemDocAgent	**0.979**	**0.916**	**0.690**	**0.461**	**0.467**	**0.419**
w/o think	0.917 (−6.2)	0.777 (−13.9)	0.583 (−10.7)	0.312 (−14.9)	0.319 (−14.8)	0.225 (−19.4)
w/o memory system	0.903 (−7.6)	0.768 (−14.8)	0.593 (−9.7)	0.324 (−13.7)	0.333 (−13.4)	0.190 (−22.9)
w/o conflict verify	0.900 (−7.9)	0.781 (−13.5)	0.592 (−9.8)	0.282 (−17.9)	0.290 (−17.7)	0.233 (−18.6)



(a) Average read time across 20 repositories.



(b) Read calls per component across 20 repositories. Dashed lines mark the overall averages.

Figure 4: Efficiency comparison between DocAgent (previous SOTA) and MemDocAgent.

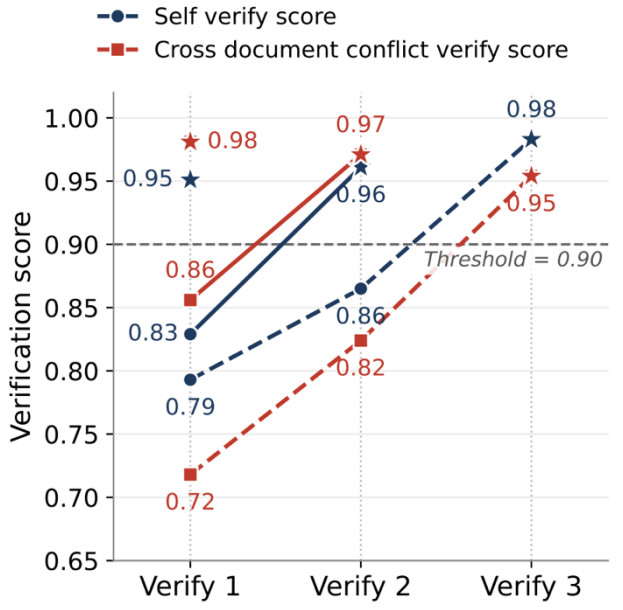


Figure 6: Documentation improvements across VERIFY attempts.

Thank you