

# An Empirical Study on Strong-Weak Model Collaboration for Repo-level Code Generation

Shubham Gandhi, Atharva Naik, Yiqing Xie, Carolyn Rose

Language Technologies Institute, Carnegie Mellon University  
{srgandhi, arnaik, yiqingxi, cp3a}@andrew.cmu.edu

## Abstract

We study cost-efficient collaboration between strong and weak language models for repository-level code generation, where the weak model handles simpler tasks at lower cost, and the most challenging tasks are delegated to the strong model. While many works propose architectures for this task, few analyze performance relative to cost. We evaluate a broad spectrum of collaboration strategies: context-based, pipeline-based, and dynamic, on GitHub issue resolution. Our most effective collaborative strategy achieves equivalent performance to the strong model while reducing the cost by 40%. Based on our findings, we offer actionable guidelines for choosing collaboration strategies under varying budget and performance constraints. Our results show that strong-weak collaboration substantially boosts the weak model’s performance at a fraction of the cost, pipeline and context-based methods being most efficient. We have also open-sourced the code <sup>1</sup> for our work.

## 1 Introduction

Recent advances in large language models (LLMs) have demonstrated impressive capabilities across complex reasoning and generation tasks. However, benchmark gains are often prioritized over practical deployment concerns, leading to systems that rely on repeated calls to expensive proprietary LLMs. For example, SWE-Agent (Yang et al., 2024) caps each run at \$4 (Kapoor et al., 2024), making even modest-scale evaluations prohibitively expensive. In contrast, systems such as Retrieval-Augmented Generation (RAG) can reduce cost by 99% (Jin et al., 2025), motivating a critical question: *How can we design effective yet cost-efficient LLM systems?* Retrieval-Augmented

Code Generation (RACG) (Wang et al., 2025b) systems like Agentless (Xia et al., 2025) and Agentless Lite (Dunn, 2025) try to address this by selectively retrieving relevant context for generation, significantly reducing costs (\$0.25 / run). However, varying task complexity necessitates hybrid setups where weak LLMs economically handle simpler tasks and escalate the rest. Yet, designing such systems is non-trivial because weak LLMs struggle with complexity, and overusing strong LLMs negates cost benefits.

While prior work explored strong-weak collaboration in NLP via LLM cascades (Chen et al., 2024b; Zhang et al., 2023; Yue et al., 2024), context augmentation (Chen et al., 2025b), ensembling (Zhang et al., 2024; Chen et al., 2024a), etc, a systematic analysis for repository level code generation remains unexplored.

To bridge this gap, we present a three-pronged taxonomy with 12 diverse strong-weak collaboration methods, including static context augmentation (e.g., Few-Shot examples, FAQs, Planning), pipeline division (e.g., Strong LM First, Prompt Reduction), and dynamic collaboration (e.g., Routing). Our taxonomy (Figure 1) draws on prior work along with tailored adaptations for code generation. We evaluate these methods across both API-based and hybrid (API + open-source) LLM pairs on the SWE-Bench Lite (Jimenez et al., 2024) benchmark. Our goal is to empirically characterize the cost-performance trade-offs of these methods and identify best practices under different budget and accuracy constraints. While issue-solving serves as a grounded and challenging testbed, our broader focus is on the general design space of strong-weak collaboration.

Our findings show that pipeline and context-based strategies offer the best average efficiency, while weak-only baselines consistently underperform. Yet, we find that examining average efficiency does not always predict optimal perfor-

<sup>1</sup><https://github.com/shubhamgandhi/codegen-strong-weak-collab>

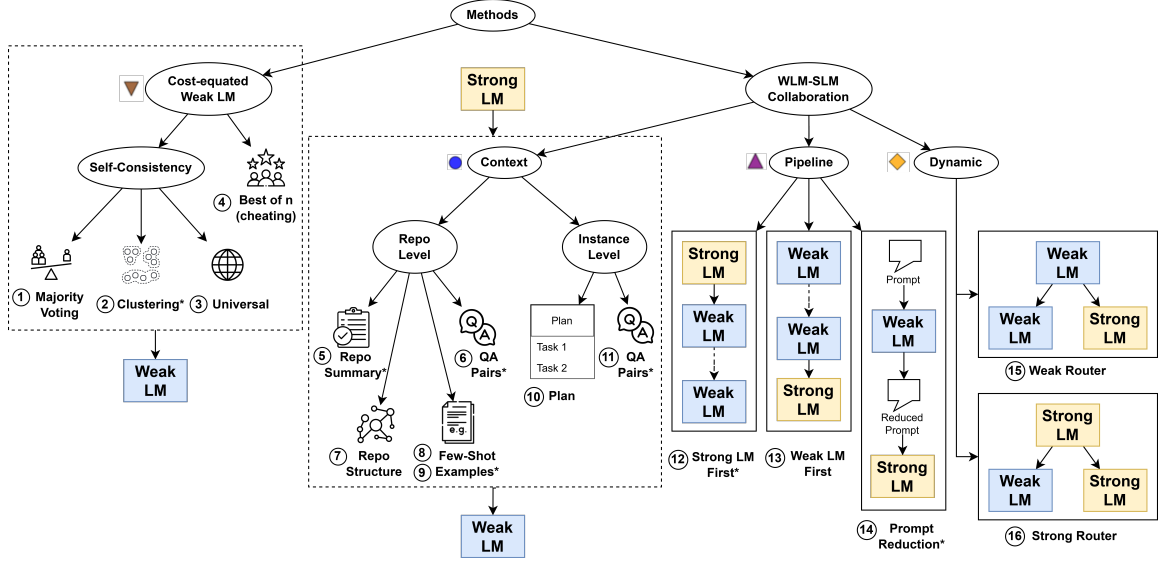


Figure 1: Taxonomy of the 14 techniques studied. \* denotes methods newly proposed or adapted in this study. We categorize them into cost-equated weak-only, context-based, pipeline-based, and dynamic collaboration methods.

mance for a given cost. Through cost-performance curves, we answer the question: *What is the best method given a performance requirement and budget constraint?* Our best strategy improves the weak LLM’s performance by  $\sim 62\%$ , almost matching the strong LLM, at  $\sim 60\%$  cost. Overall, we provide actionable principles for designing cost-efficient, collaborative LLM systems for complex code generation tasks in the future.

## 2 Related Work

### 2.1 Repository-Level Code Generation

Repository-Level Code Generation is a complex problem that requires LLMs to leverage the context of an entire repository and understand the structure and interleaving dependencies. The SWE-Bench benchmark (Jimenez et al., 2024) exemplifies this in the form of issue-resolution, a real-world repository-level code generation task, where the system must make edits to the code given an issue description and the relevant repository snapshot. This task can mainly be broken down into two sub-tasks: Fault Localization and Program Repair. Recent methods often use retrieval or structured pipelines to tackle this problem. For example, Retrieval Augmented Code Generation (RACG) frameworks first retrieve relevant code or documentation from the repository to aid in efficient fault localization before generation (Wang et al., 2025b; Li et al., 2025). While there exist agentic techniques like SWE-Agent (Yang et al., 2024) that are able to achieve a significant performance on this task, it

often comes at a steep cost involving multiple calls to expensive LLMs. On the other hand, Agentless (Xia et al., 2025) entirely avoids multi-turn agent calls and instead sticks to a fixed three-step process involving localization, repair and validation, showing strong performance and low cost on SWE-Bench.

### 2.2 Cost reduction techniques for LLMs

Strong LLMs achieve high performance but are often too expensive to use at scale. To tackle this, several lines of work explore cost-efficient alternatives. Self-consistency (Wang et al., 2023; Chen et al., 2024c) aggregates multiple generations from the same LLM to improve performance. Other approaches rely on strong-weak collaboration. Amayuelas et al. (2025); Wang et al. (2025a) leverage a strong LM’s superior reasoning capabilities to generate a plan for execution by a cheaper weak LM. Routing-based methods like FrugalGPT (Chen et al., 2024b) dynamically select LLMs to optimize cost and quality. Similarly, LLM Cascades (Yue et al., 2024) escalate to strong LLMs only when weak LLMs fail. Drawing from this, we curate a taxonomy of strong-weak collaboration strategies, including self-consistency, planning, LLM cascades (Weak LM First), routing among others, adapted to repository-level code generation. This allows us to empirically evaluate their cost-performance trade-offs in a unified framework.

### 3 Methodology

To conduct a systematic analysis, as shown in Figure 1, we present a taxonomy of strong-weak collaboration methods for repo-level RACG, where the methods are either based on or inspired by prior studies (Chen et al., 2025a; Bai et al., 2024; Xu et al., 2024; Lu et al., 2024; Chen et al., 2025c).

**Preliminary.** We conduct our experiments with Agentless-Lite (Dunn, 2025), a two-step repo-level RACG framework. We first apply a retriever to retrieve the top- $k$  relevant documents (e.g., files in a repository) based on their similarity score with the query  $q$ . Then we apply an LM to iteratively generate the code with the documents as context until it is in the correct code patch format.

**Cost-equated weak LM.** We first study several baselines that *only* involve the weak LM (McDonald et al., 2025). Self-consistency (Wang et al., 2023) enhances weak LMs by sampling  $n$  diverse outputs, where  $n \approx \text{Cost}_{\text{strong}} / \text{Cost}_{\text{weak}}$  and selecting the most consistent one via: ① Majority Voting ( $SC_m$ ), ② Clustering ( $SC_c$ ) and ③ Universal ( $SC_u$ ). We also experiment on a “cheating” method, ④ Best of  $n$ . Detailed explanations are included in Appendix §A.1

**Static Context Augmentation.** We experiment methods that leverage the strong LM’s superior problem-understanding capabilities to augment the context and use the weak LM for iterative generation to reduce cost. We first study methods that prompt the strong LM to generate repo-level information to provide background information for code generation, including: ⑤ Repository Summary, which generates a summary of the repository based on its README file and directory structure, ⑥ Repo-Level FAQs, which is similar but generates a set of FAQs instead, ⑦ Repo Structure, which summarizes the repository structure. Here we use RepoGraph (Ouyang et al., 2025), which encodes the repository in a graph. We also study few-shot examples for each repository, which can be constructed by: ⑧ Few-shot (Random) that randomly selects  $k$  (input, successful strong LM output) pairs, and ⑨ Few-shot (Similarity), which selects few-shot demos based on the problem statements’ embedding similarity. We also study methods that use the strong LM to generate instance-specific context. Compared to repo-level context, it typically requires a higher cost but provides more precise and example-relevant knowledge. We analyze: ⑩ Planning, which generates high-level planning for

each instance (Amayuelas et al., 2025; Wang et al., 2025a), and ⑪ Instance-Level QA pairs, which generates a set of FAQs for each instance with the code generation query (e.g., issue description) and retrieved code as context.

**Pipeline Division.** These methods aim to reduce cost while maintaining performance by selectively calling the strong LM and weak LM sequentially as part of a hard-coded pipeline. Specifically, we compare: ⑫ Strong LM First, which prompts the strong LM to make the first attempt and calls the weak LM to iteratively refine its solution until it is correctly formatted. ⑬ Weak LM First, where the weak LM first makes  $n$  attempts to solve the issue, following model cascades (Chen et al., 2024b; Zhang et al., 2023; Yue et al., 2024). If it fails to generate a valid patch, the strong LM makes one attempt. Based on previous study’s conclusion that weak LMs have comparable performance to strong LMs on localization (Xia et al., 2025), we present ⑭ Prompt Reduction, where we perform a preliminary call to the weak LM to reduce the code context by removing irrelevant code, thus reducing the overall context to be passed to the strong LM.

**Dynamic Collaboration.** Unlike §3, where the pipeline is hard-coded, we allow the decision to use the strong or weak model to be made dynamically during inference. Specifically, we invoke a router (Chen et al., 2024b) to decide if the problem is *simple* or *complex* and appropriately delegate it to the weak or strong LM. We evaluate both weak LMs and strong LMs as routers and denote them as ⑮ Weak Router and ⑯ Strong Router.

### 4 Experiments

**Models.** We consider strong-weak model pairs that have a noticeable gap in performance and cost, and the more expensive model performs better: **Strong LMs:** O3-mini, O4-mini (OpenAI, 2025) and GPT-4o-mini (OpenAI, 2024); **Weak LMs:** GPT-4o-mini and Qwen2.5-Coder series (Hui et al., 2024).

**Dataset and Agentic Framework.** We conduct all experiments on the SWE-bench Lite subset (Jimenez et al., 2024) which has 300 issues from 11 Python repositories. We run the Agentless Lite framework (Dunn, 2025) once per instance for our experiments, using voyage-code-3<sup>2</sup> for retrieval, techniques mentioned in §3 for generation (Details in Appendix §A.1).

<sup>2</sup><https://docs.voyageai.com/docs/embeddings>

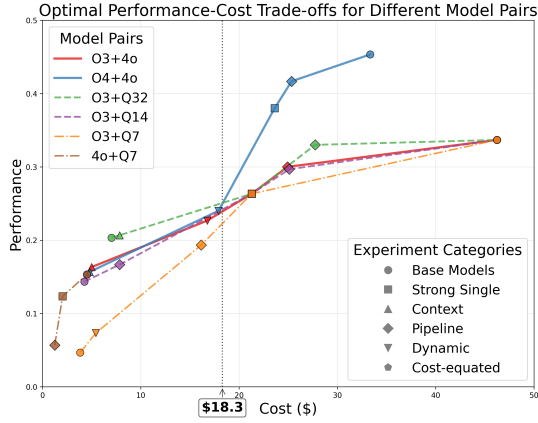


Figure 2: Performance vs. Cost curves for different Strong-Weak Model Pairs. O3 - O3-mini; O4 - O4-mini; 4o - GPT-4o-mini; Qx - Qwen2.5-Coder-xB. Detailed pairwise results in Appendix Figure 4 and Appendix Tables 3-8.

**Evaluation Metrics.** The following metrics are reported for each method: (1) *Resolution Rate*: Proportion of instances for which the generated patch successfully resolves the issue; and (2) *Cost*: Total generation cost (\$) including additional method costs, but excluding retrieval cost, which remains constant across all methods.

## 5 Main Results

We focus our evaluation on the generator, as voyage-code-3 achieves strong retrieval ( $\sim 92\%$  recall,  $\sim 76\%$  MRR). Our findings uncover consistent patterns when strong-weak collaboration succeeds or fails (Appendix Figure 4, Appendix Tables 3-8).

**Cost-equated baselines vs. Strong-Weak collaboration.** As shown in Figure 2, Appendix Figure 4 and Appendix Tables 3-14 and Appendix Figure 4, across nearly all model pairs, the cost-equated baselines, i.e. sampling multiple times to match the strong LLM’s cost, underperform compared to collaboration. Surprisingly, some baselines even hurt weak LLM’s performance, which is likely due to faltering patch selection. Our best collaboration strategy, Strong LM First, achieves a 0.4167 resolution rate, which is  $\sim 92\%$  more than the best corresponding cost-equated baseline, i.e., best-of-n for GPT-4o-mini cost-equated with O4-mini.

**Efficacy of Methods Across Taxonomy Groups.** Analyzing by taxonomy (§3), on average, pipeline and context-based strategies yield the highest cost-efficiency across model pairs, followed by dy-

namic methods and *Best of n*, with self-consistency approaches being the least efficient (Appendix §A.3). However, average efficiency alone is not a reliable indicator of the best method. Performance–cost curves (Figure 2) often intersect, showing that *Weak LM First* is optimal when budget is low due to minimal strong LM usage, while *Strong LM First* dominates once budget increases. For example, with O3-mini + Qwen2.5-Coder-32B, *Strong LM First* achieves equivalent performance to the strong model at just  $\sim 60\%$  cost. We also observe a regime shift around the \$18.3 mark as Figure 2 shows, below which O3 + Qwen2.5-Coder-32B performs best, whereas above it, O4 + GPT-4o-mini becomes more cost-effective.

These results suggest that no method is universally optimal. Instead, choosing the best strategy depends on the deployment scenario, specifically the available budget and required performance, which is enabled by plots like Figure 2. Given a minimum performance threshold and a budget cap, the optimal method corresponds to the curve that reaches the highest resolution rate within the feasible region i.e., the top-left quadrant defined by those constraints. For instance, if the target is at least 20% resolution within a \$20 budget, *Weak Router* with O4-mini + GPT-4o-mini emerges as the most cost-effective choice. These observations yield actionable guidance: (1) Cost-equated weak-only methods are inefficient; (2) Pipeline-based methods outperform context-only methods when budget allows; (3) *Weak LM First* and *Weak Router* are strong choices under tight budgets; and (4) *Strong LM First* performs best in higher-budget regimes, often approaching strong LM performance at reduced cost.

### 5.1 Interesting cases

**Instance-level help is better than repo-level help.** Repo-level context (e.g., summaries, structure, QA pairs) consistently failed to improve weak LM performance. Such information is too coarse and may distract from instance-specific signals. Even few-shot examples, whether random or similar, often reduced accuracy. In contrast, instance-level augmentation (plans, QA pairs) significantly boosted resolution rates, justifying their higher cost.

**Weak Router is better than Strong Router.** Counterintuitively, *Weak Router* frequently outperformed *Strong Router*, both in accuracy and cost. For example, with O4-mini + Qwen2.5-Coder-32B, *Weak Router* achieved 6 percentage points higher



resolution at  $\sim 20\%$  lower cost. We suspect that stronger models, while good at problem solving, may “overthink” routing decisions (Cuadron et al., 2025). Only with very weak LMs (e.g., Qwen-7B) did strong routing slightly edge ahead, suggesting weak routing is often the more efficient choice.

**Prompt Reduction trades turn-wise validity for overall performance.** *Prompt Reduction* offers a surprising trade-off where it lowers the valid patch rate to  $\sim 65\%$ , which is well below the  $\sim 95\%$  average of other methods, yet often achieves higher resolution rates overall. Due to aggressive pruning of irrelevant context by the weak LM, the strong LM is forced to focus on the most salient code, leading to more successful fixes despite more retries. In effect, Prompt Reduction sacrifices turn-wise reliability in favor of sharper attention, making it a high-variance but high-upside strategy, especially when correctness matters more than efficiency.

## 6 Conclusion

We present a detailed taxonomy of strong-weak LLM collaboration techniques for repository-level code generation. Our results show substantial cost-efficiency gains with pipeline and context-based collaboration, maximizing average efficiency. However, no method is universally optimal, and the ideal choice depends on budget and performance constraints. We offer actionable principles for selecting when to prioritize weak models, apply dynamic routing, or escalate to hybrid pipelines. Our insights will help practitioners deploy accurate LLM systems aligned with realistic efficiency constraints. Future work can extend this taxonomy to more complex approaches and domains for improved cost-performance tradeoffs.

## 7 Acknowledgement

This work was supported in part by NSF grant DSES-2222762.

## 8 Limitations and Potential Risk

For simplicity of experimentation, we restrict the scope of this study to Agentless Lite, a simple RAG + Code generation framework. However, we believe that the methods we consider are generalizable and simple enough to be extended to more complex architectures. We also limit the scope of this study to the code generation domain, however, parallels can easily be drawn to other domains and

the methods can be tweaked to suit those applications. To curb costs for experimentation, we use the Lite subset of the SWE-Bench dataset. We focus the study only on inference-based approaches and do not consider any training or finetuning-related approaches since those often require expensive hardware setups which defeats the overall purpose of cost efficiency. We measure cost via Token / API usage and do not consider factors like latency or energy consumption since those would vary significantly with hardware. Potential risks of our work are similar to risks largely associated with LLM-based code generation in general. These include but are not limited to the methods studied being used to purposefully or inadvertently generate vulnerable or malicious code.

## 9 Scientific Artifacts

We use a variety of proprietary and open sourced LLMs for our study. We strictly use commercial APIs to access proprietary LLMs following the conditions outlined by the respective companies. We use an open source framework, Agentless Lite, for our experiments. We use the test set of the SWE-Bench Lite Benchmark for our experiments. To the best of our knowledge, it does not contain personally identifying information or offensive content.

## References

- Alfonso Amayuelas, Jingbo Yang, Saaket Agashe, Ashwin Nagarajan, Antonis Antoniadis, Xin Eric Wang, and William Wang. 2025. [Self-resource allocation in multi-agent llm systems](#). *Preprint*, arXiv:2504.02051.
- Guangji Bai, Zheng Chai, Chen Ling, Shiyu Wang, Jiaying Lu, Nan Zhang, Tingwei Shi, Ziyang Yu, Mengdan Zhu, Yifei Zhang, Xinyuan Song, Carl Yang, Yue Cheng, and Liang Zhao. 2024. [Beyond efficiency: A systematic survey of resource-efficient large language models](#). *Preprint*, arXiv:2401.00625.
- Dong Chen, Shuo Zhang, Yueting Zhuang, Siliang Tang, Qidong Liu, Hua Wang, and Mingliang Xu. 2024a. [Improving large models with small models: Lower costs and better performance](#). *CoRR*, abs/2406.15471.
- Lingjiao Chen, Matei Zaharia, and James Zou. 2024b. [FrugalGPT: How to use large language models while reducing cost and improving performance](#). *Transactions on Machine Learning Research*.
- Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. 2025a. [Towards](#)

- reasoning era: A survey of long chain-of-thought for reasoning large language models. *Preprint*, arXiv:2503.09567.
- Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. 2024c. *Universal self-consistency for large language models*. In *ICML 2024 Workshop on In-Context Learning*.
- Yongchao Chen, Yilun Hao, Yueying Liu, Yang Zhang, and Chuchu Fan. 2025b. *Codesteer: Symbolic-augmented language models via code/text guidance*. In *Forty-second International Conference on Machine Learning*.
- Zhijun Chen, Jingzheng Li, Pengpeng Chen, Zhuoran Li, Kai Sun, Yuankai Luo, Qianren Mao, Dingqi Yang, Hailong Sun, and Philip S. Yu. 2025c. *Harnessing multiple large language models: A survey on llm ensemble*. *Preprint*, arXiv:2502.18036.
- Alejandro Cuadron, Dacheng Li, Wenjie Ma, Xingyao Wang, Yichuan Wang, Siyuan Zhuang, Shu Liu, Luis Gaspar Schroeder, Tian Xia, Huanzhi Mao, Nicholas Thumiger, Aditya Desai, Ion Stoica, Ana Klimovic, Graham Neubig, and Joseph E. Gonzalez. 2025. *The danger of overthinking: Examining the reasoning-action dilemma in agentic tasks*. *Preprint*, arXiv:2502.08235.
- Soren Dunn. 2025. *Agentless-lite*. <https://github.com/sorendunn/Agentless-Lite>. Accessed: 2025-05-02.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, and 5 others. 2024. *Qwen2.5-coder technical report*. *Preprint*, arXiv:2409.12186.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. *SWE-bench: Can language models resolve real-world github issues?* In *The Twelfth International Conference on Learning Representations*.
- Haolin Jin, Linghan Huang, Haipeng Cai, Jun Yan, Bo Li, and Huaming Chen. 2025. *From llms to llm-based agents for software engineering: A survey of current, challenges and future*. *Preprint*, arXiv:2408.02479.
- Sayash Kapoor, Benedikt Stroebel, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. 2024. *AI agents that matter*. *Preprint*, arXiv:2407.01502.
- Jia Li, Xianjie Shi, Kechi Zhang, Lei Li, Ge Li, Zhengwei Tao, Jia Li, Fang Liu, Chongyang Tao, and Zhi Jin. 2025. *Coderag: Supportive code retrieval on bigraph for real-world code generation*. *Preprint*, arXiv:2504.10046.
- Jinliang Lu, Ziliang Pang, Min Xiao, Yaochen Zhu, Rui Xia, and Jiajun Zhang. 2024. *Merge, ensemble, and cooperate! a survey on collaborative strategies in the era of large language models*. *Preprint*, arXiv:2407.06089.
- Tyler McDonald, Anthony Colosimo, Yifeng Li, and Ali Emami. 2025. *Can we afford the perfect prompt? balancing cost and accuracy with the economical prompting index*. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 7075–7086, Abu Dhabi, UAE. Association for Computational Linguistics.
- OpenAI. 2024. Gpt-4o system card. <https://openai.com/index/gpt-4o-system-card/>.
- OpenAI. 2025. *Openai o3 and o4-mini system card*. <https://openai.com/index/o3-o4-mini-system-card/>.
- Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. 2025. *Repograph: Enhancing AI software engineering with repository-level code graph*. In *The Thirteenth International Conference on Learning Representations*.
- Evan Z Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, William Song, Vaskar Nath, Ziwen Han, Sean M. Hendryx, Summer Yue, and Hugh Zhang. 2025a. *Planning in natural language improves LLM search for code generation*. In *The Thirteenth International Conference on Learning Representations*.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. *Self-consistency improves chain of thought reasoning in language models*. In *The Eleventh International Conference on Learning Representations*.
- Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. 2025b. *CodeRAG-bench: Can retrieval augmented code generation?* In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 3199–3214, Albuquerque, New Mexico. Association for Computational Linguistics.
- Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2025. *Demystifying llm-based software engineering agents*. *Proc. ACM Softw. Eng.*, 2(FSE).
- Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. 2024. *A survey on knowledge distillation of large language models*. *Preprint*, arXiv:2402.13116.
- John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. 2024. *SWE-agent: Agent-computer interfaces enable automated software engineering*. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. 2024. [Large language model cascades with mixture of thought representations for cost-efficient reasoning](#). In *The Twelfth International Conference on Learning Representations*.

Jieyu Zhang, Ranjay Krishna, Ahmed H. Awadallah, and Chi Wang. 2023. [Ecoassistant: Using llm assistant more affordably and accurately](#). *Preprint*, arXiv:2310.03046.

Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. 2024. [Small language models need strong verifiers to self-correct reasoning](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 15637–15653, Bangkok, Thailand. Association for Computational Linguistics.

## A Appendix

### A.1 Hyperparameters and Implementation Details

We plan on open-sourcing our code soon. We set the `max_files` parameter for Agentless Lite to 5. For OpenAI’s thinking models like O3-mini and O4-mini, we set the `reasoning_effort` parameter to high. We start the generation at a base temperature of 0.0, which is gradually increased by 0.1 every time the model fails to generate a valid patch. We set the max number of retries as 10 for the base settings. All open-source model inferences were run using vLLM on 4 L40 GPUs, leveraging tensor parallelism for efficient distributed execution. We configured the server with `-tensor-parallel-size 4` and set `-gpu-memory-utilization 0.8` to optimize memory usage without overcommitment. Inference was performed in bfloat16 precision with `-enforce-eager` execution to reduce kernel launch latency. To support long-context models, we enabled chunked prefill and set the `rope_scaling` parameter to use YARN with a factor of 4.0, extending support to sequences up to 32k tokens. Additional parameters included a maximum of 32,768 batched tokens, 8 concurrent sequences, and 16GB of CPU swap space to handle occasional memory spikes.

**Cost-equated weak LM.** We first study several baselines that *only* involve the weak LM (McDonald et al., 2025). Self-consistency (Wang et al., 2023) enhances weak LMs by sampling  $n$  diverse outputs, where  $n \approx \text{Cost}_{\text{strong}} / \text{Cost}_{\text{weak}}$  and selecting the most consistent one via: ① Majority Voting ( $SC_m$ ), where we select the most frequent patch, ② Clustering ( $SC_c$ ), where we cluster the  $n$  candidate patches and select a patch at random from the largest cluster. We cluster the patches based on similarity calculated using `difflib.SequenceMatcher` ③ Universal ( $SC_u$ ), where we follow Chen et al. (2024c) and prompt the weak LM itself to select the final patch. We also experiment on a “cheating” method, ④ Best of  $n$ , which checks whether any of the candidate patches is correct and serves as an upper bound for candidate selection methods.

**Evaluation Metrics.** The following metrics are reported for each method: (1) *Resolution Rate*: Proportion of instances for which the generated patch successfully resolves the issue; (2) *Average #Iterations*: The average number of calls made to

strong and weak models, denoted by  $s$  and  $w$  respectively; (3) *Valid Patch Rate*: Fraction of instances for which the system was able to generate a patch that adheres to the required format; (4) *Cost*: Total generation cost (\$) to process all 300 SWEBench-Lite instances, including additional method costs, but excluding retrieval cost, which remains constant across all methods, and (5) *Efficiency Score*: The ratio of Resolution Rate to Cost.

### A.2 Prompts

#### A.2.1 Agentless Lite Base Prompt

```
We are currently solving the following
issue within our repository. Here is
the issue text:
--- BEGIN ISSUE ---
{problem_statement}
--- END ISSUE ---
```

```
Below are some code segments, each from
a relevant file. One or more of
these files may contain bugs:
--- BEGIN FILE ---
{retrieval}
--- END FILE ---
```

Please first localize the bug based on the issue statement, and then generate `*SEARCH/REPLACE*` edits to fix the issue.

Every `*SEARCH/REPLACE*` edit must use this format:

1. The file path
2. The start of search block: `<<<<<< SEARCH`
3. A contiguous chunk of lines to search for in the existing source code
4. The dividing line: `=====`
5. The lines to replace into the source code
6. The end of the replace block: `>>>>>> REPLACE`

Here is an example:

```
```python
### mathweb/flask/app.py
<<<<<< SEARCH
from flask import Flask
=====
import math
from flask import Flask
>>>>>> REPLACE
```
```

Please note that the `*SEARCH/REPLACE*` edit REQUIRES PROPER INDENTATION. If you would like to add the line `'print(x)'`, you must fully write that out, with all those spaces before the code! Wrap the `*SEARCH/REPLACE*` edit in blocks ````python...````.



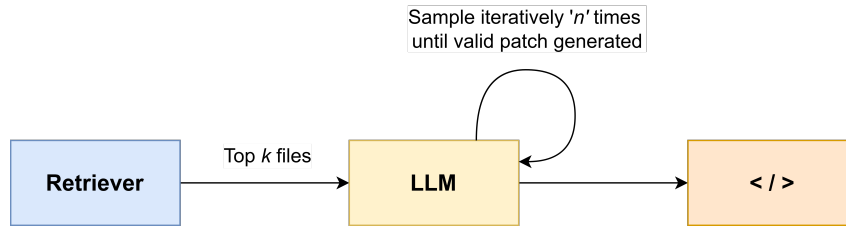


Figure 3: The Agentless Lite Framework: RAG + Code Generation

| Experiment                      | Resolution Rate | Avg. #iterations | Valid Patch Rate | Total Generation Cost | Efficiency    |
|---------------------------------|-----------------|------------------|------------------|-----------------------|---------------|
| o3-mini <sup>1</sup>            | 0.3367          | 1.83             | 0.977            | 46.224                | 0.0073        |
| o4-mini <sup>2</sup>            | <b>0.4533</b>   | <b>1.32</b>      | <b>0.993</b>     | 33.328                | 0.0136        |
| claude-3.5-sonnet <sup>3</sup>  | 0.2667          | 2.13             | 0.957            | <b>140.126</b>        | 0.0019        |
| gpt-4o-mini <sup>4</sup>        | 0.1533          | 2.09             | 0.960            | 4.529                 | <b>0.0338</b> |
| qwen2.5-coder-7b <sup>5</sup>   | 0.0467          | 6.48             | 0.617            | 3.839*                | 0.0122        |
| qwen2.5-coder-14b               | 0.1433          | 2.91             | 0.937            | 4.257*                | 0.0337        |
| qwen2.5-coder-32b               | 0.2033          | 2.35             | 0.967            | 7.021*                | 0.0290        |
| r1-distill-qwen-7b <sup>6</sup> | 0.0033          | 9.91             | 0.013            | 115.161*              | 0.0000        |

<sup>1</sup> o3-mini-2025-01-31 from <https://platform.openai.com/docs/models/o3-mini>

<sup>2</sup> o4-mini-2025-04-16 from <https://platform.openai.com/docs/models/o4-mini>

<sup>3</sup> claude-3-5-sonnet-20241022 from <https://docs.anthropic.com/en/docs/about-claude/models/all-models>

<sup>4</sup> gpt-4o-mini-2024-07-18 from <https://platform.openai.com/docs/models/gpt-4o-mini>

<sup>5</sup> Qwen-2.5-Coder-Instruct series models from <https://huggingface.co/Qwen>

<sup>6</sup> DeepSeek-R1-Distill-Qwen-7B from <https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B>

Table 1: Base performance of various models with Agentless Lite. Efficiency is calculated as resolution rate divided by generation cost. Models like claude-3-5-sonnet-20241022 and r1-distill-qwen-7b are excluded from our main study due to poor cost-effectiveness. \* - Estimated cost based on comparable API pricing.

## A.2.2 Repo Summary

I need you to provide high-level insights about the following repository: {repo\_name}

Based on the repository structure and README below, generate a comprehensive overview of this repository that could help guide a language model in solving technical issues.

Repository Structure:  
{repo\_structure}

README Content:  
{readme\_content}

Please provide the following insights. For each point, provide concrete details and specific examples from the codebase - high-level doesn't mean vague, it means providing a clear architectural overview with specific names, patterns, and implementations:

- Core Purpose and Functionality:
  - What specific problem does this repository solve?
  - What are its primary features and capabilities?

### 2. Main Architectural Patterns:

- Identify concrete architectural patterns used in this codebase
- EXAMPLE: Plugin based architecture, layered architecture, etc

### 3. Module Organization:

- Name the specific key modules and their exact responsibilities
- EXAMPLE: I/O module, error-handling module, etc

### 4. Key Abstractions and Concepts:

- List the actual fundamental abstractions used in the codebase
- EXAMPLE: Quantity class for numerical values, Logger class for logging, etc

### 5. Design Patterns:

- Identify specific recurring code patterns with examples
- EXAMPLE: Factory methods, Decorators, etc

### 6. Error Handling Approaches:

- Describe precise error handling mechanisms used in the codebase
- EXAMPLE: Custom exception hierarchies, warnings, etc

Focus on providing actionable architectural insights that would be

valuable for understanding the repository's design philosophy and core abstractions. Your response should contain specific implementation details that would help someone understand how to navigate, extend, and debug the codebase to solve issues.

### A.2.3 Repo Level QA Pairs

I need you to generate a comprehensive FAQ about the repository: {repo\_name}

Based on the repository structure and README below, create a detailed set of technical FAQs that would help a developer solve issues in this codebase. These FAQs should serve as guidance for someone who is trying to resolve bugs or implement new features.

Repository Structure:  
{repo\_structure}

README Content:  
{readme\_content}

Please generate 15-20 frequently asked questions with detailed answers about:

1. Code Organization and Architecture:
  - How is the codebase structured?
  - What are the key modules and their responsibilities?
  - How do the different components interact?
2. Common Patterns and Conventions:
  - What design patterns are commonly used?
  - What are the naming conventions and code style expectations?
  - Are there specific patterns for implementing new features?
3. Typical Debugging Approaches:
  - What are common error patterns and their solutions?
  - How to debug specific types of issues in this codebase?
  - What are common pitfalls when modifying this code?
4. Implementation Details:
  - How are core abstractions implemented?
  - What are the key algorithms or data structures used?
  - How does the error handling system work?
5. Testing Considerations:
  - How is testing typically done in this codebase?

- What should be considered when writing tests?
- Are there common test fixtures or utilities?

For each question, provide detailed, specific answers with concrete examples from the codebase when possible. Focus on information that would be most valuable to someone trying to fix bugs or implement new features. The FAQs should reflect the actual patterns and practices used in this specific repository, not generic software development advice.

### A.2.4 Repo Structure

We are currently solving the following issue within our repository. Here is the issue text:

```
--- BEGIN ISSUE ---  
{problem_statement}  
--- END ISSUE ---
```

Below are some code segments, each from a relevant file. One or more of these files may contain bugs:

```
--- BEGIN FILE ---  
{retrieval}  
--- END FILE ---
```

To help you better understand the contexts of the code segments, we provide a set of dependencies of the code segments.

The dependencies reflect how the functions/classes in the code segments are referenced in the codebase.

```
--- BEGIN DEPENDENCIES ---  
{dependencies}  
--- END DEPENDENCIES ---
```

Please first localize the bug based on the issue statement, and then generate \*SEARCH/REPLACE\* edits to fix the issue.

Every \*SEARCH/REPLACE\* edit must use this format:

1. The file path
2. The start of search block: <<<<<<< SEARCH
3. A contiguous chunk of lines to search for in the existing source code
4. The dividing line: =====
5. The lines to replace into the source code
6. The end of the replace block: >>>>>>> REPLACE

Here is an example:

```
```python  
### mathweb/flask/app.py  
<<<<<<< SEARCH
```

```

from flask import Flask
=====
import math
from flask import Flask
>>>>>>> REPLACE
```

Please note that the *SEARCH/REPLACE* edit REQUIRES PROPER INDENTATION. If you would like to add the line ' print(x)', you must fully write that out, with all those spaces before the code!
Wrap the *SEARCH/REPLACE* edit in blocks
```python...```.

```

### A.2.5 Few Shot Examples

Here are some {similar}example issues from the same repository along with the target file that were changed and final patch generated by an expert{successful}:

```

--- BEGIN EXAMPLES ---
{few_shot_examples}
--- END EXAMPLES ---

```

We are currently solving the following issue within our repository. Here is the issue text:

```

--- BEGIN ISSUE ---
{problem_statement}
--- END ISSUE ---

```

Below are some code segments, each from a relevant file. One or more of these files may contain bugs:

```

--- BEGIN FILE ---
{retrieval}
--- END FILE ---

```

Please first localize the bug based on the issue statement, and then generate \*SEARCH/REPLACE\* edits to fix the issue.

Every \*SEARCH/REPLACE\* edit must use this format:

1. The file path
2. The start of search block: <<<<<< SEARCH
3. A contiguous chunk of lines to search for in the existing source code
4. The dividing line: =====
5. The lines to replace into the source code
6. The end of the replace block: >>>>>> REPLACE

Here is an example:

```

```python
### mathweb/flask/app.py
<<<<<< SEARCH
from flask import Flask
=====
import math
from flask import Flask
>>>>>> REPLACE

```

```

```

Please note that the *SEARCH/REPLACE* edit REQUIRES PROPER INDENTATION. If you would like to add the line ' print(x)', you must fully write that out, with all those spaces before the code!
Wrap the *SEARCH/REPLACE* edit in blocks
```python...```.

```

### A.2.6 Plan

We are currently solving the following issue within our repository. Here is the issue text:

```

--- BEGIN ISSUE ---
{problem_statement}
--- END ISSUE ---

```

Below are some code segments, each from a relevant file. One or more of these files may contain bugs:

```

--- BEGIN FILE ---
{retrieval}
--- END FILE ---

```

Please analyze the issue and provide a detailed plan to fix it. Do NOT generate any code patches or specific edits.

Your plan should include:

1. Bug localization: Identify which file (s) contain the bug based on the issue statement
2. Root cause analysis: Explain why the bug is occurring
3. Solution approach: Describe conceptually how to fix the issue
4. Implementation strategy: Outline the logical steps needed to implement the solution

Keep your analysis focused on the problem-solving approach rather than specific code changes.

### A.2.7 Instance Level QA Pairs

We are currently solving the following issue within our repository. Here is the issue text:

```

--- BEGIN ISSUE ---
{problem_statement}
--- END ISSUE ---

```

Below are some code segments, each from a relevant file. One or more of these files may contain bugs:

```

--- BEGIN FILE ---
{retrieval}
--- END FILE

```

Please create a detailed FAQ (Frequently Asked Questions) document that would help a junior developer

understand and fix this specific issue. Do NOT generate any code patches or specific edits.

Your FAQ should include 7-10 questions and answers about:

1. Issue Understanding:
  - What is the exact problem described in the issue?
  - What are the expected vs. actual behaviors?
  - What conditions trigger this issue?
2. Codebase Navigation:
  - Which specific files and functions are most relevant to this issue?
  - What are the key components involved in this functionality?
  - How do these components interact?
3. Technical Analysis:
  - What are the potential root causes of this issue?
  - What code patterns or anti-patterns might be contributing to the bug?
  - What specific edge cases might not be handled correctly?
4. Implementation Guidance:
  - What approaches could be used to fix this issue?
  - What implementation pitfalls should be avoided?
  - How should the solution be tested?
5. Codebase Specifics:
  - What patterns or conventions does this codebase use that are relevant to the fix?
  - What existing helper functions or utilities could be leveraged?
  - What dependencies or side effects need to be considered?

Make your questions and answers detailed, specific to this issue, and include concrete references to the code when possible. Avoid generic programming advice - focus on information that directly helps solve this specific issue.

## A.2.8 Prompt Reduction

We are currently solving the following issue within our repository. Here is the issue text:

```
--- BEGIN ISSUE ---  
{problem_statement}  
--- END ISSUE ---
```

Below are some code segments, each from a relevant file. One or more of these files may contain bugs:

```
--- BEGIN FILE ---  
{retrieval}  
--- END FILE ---
```

Your task is to identify the most relevant code sections that contain the bug or need to be modified to fix the issue.

Please output only the file paths and relevant code sections in this format:

```
### <file_path>  
```python  
# relevant code section 1  
```
```

```
### <file_path>  
```python  
# relevant code section 2  
```
```

Only include the minimum necessary code with sufficient context to understand and fix the issue. Ensure that the code is complete and valid Python code.

Don't include any explanations or reasoning - just the file paths and code sections.

## A.2.9 Weak / Strong Router

You are an expert software engineer tasked with analyzing software issues to determine the most efficient debugging approach.

Please analyze the following issue and codebase:

Issue description:

```
--- BEGIN ISSUE ---  
{problem_statement}  
--- END ISSUE ---
```

Relevant code:

```
--- BEGIN CODE ---  
{retrieval}  
--- END CODE ---
```

Your goal is to classify this issue as either SIMPLE or COMPLEX.

SIMPLE issues typically have these characteristics:

- Clear localization of the bug in the code
- Straightforward cause-effect relationship
- Relatively isolated impact (limited to one function or module)
- Common programming patterns or errors
- Solution likely follows established best practices

COMPLEX issues typically have these characteristics:

- Multiple components involved in the bug
- Subtle interactions between different parts of the codebase



- Requires deep reasoning about codebase architecture
- Edge cases that are difficult to identify
- May require creative or non-obvious solutions

Based solely on your analysis of the issue and code, respond with ONLY one of these two options:

- SIMPLE
- COMPLEX

Provide no explanation, reasoning, or additional text - just output the single classification word.

### A.2.10 Universal Self-Consistency Patch Selection

```
I have generated the following {
  n_samples} potential solutions to
  fix this issue:

# Problem Statement
{problem_statement}

# Relevant Files
{formatted_files}

# Generated Solutions
{patches}

Based on the above solutions, select the
most consistent and correct
solution. Analyze the similarities
and differences between the
solutions, and select the one that
best addresses the problem statement
while making minimal and precise
changes.

Your selection should be based on:
1. Correctness (does it solve the issue
described in the problem statement)
2. Consensus (do multiple solutions
agree on a similar approach)
3. Simplicity (does it make minimal
necessary changes)

Return your selection as "SELECTED_PATCH
: X" where X is the number of the
chosen patch (1 to {n_samples}) and
then explain your reasoning.
```

## A.3 Statistical Analysis

In order to systematically compare average efficiency, accuracy, and cost for 19 separate method configurations under 5 different method groups, we evaluated performance on SWE-Bench Lite over 6 strong-weak model pairs using 2 different ANOVA models for each of the 3 dependent measures, namely accuracy, efficiency, and cost.

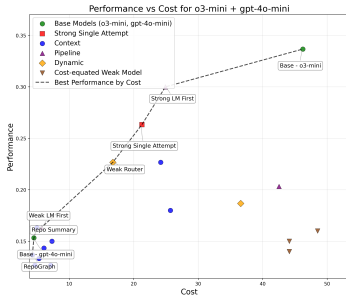
For the first set of ANOVAs, we include Model Pair and Method Group as independent variables

so that we can capture average trends while controlling for the effect of model pair. Both variables were highly significant in all three models ( $p < .00001$ ), and each of the models captured between 66% and 80% of the variance in dependent measure, thus indicating that these models tell a meaningful story about what affects these dependent measures. For accuracy, Pipeline approaches were significantly better than all others. Self-Consistency and Context were significantly worse than all others. Best of n and Dynamic were in the middle. For cost, Self-Consistency and Best of n were significantly more than all others. Context was significantly cheaper than all others. And Pipeline and Dynamic were less than Self-Consistency and Best of n and more than Context. In terms of average efficiency, Pipeline and Context were significantly better than all others.

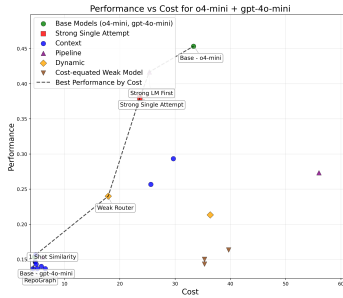
Taking a look at a higher level of granularity, we constructed a second set of ANOVAs shown in Appendix Table 2 with the specific methods rather than method groups. In this case, we see some overlap between groups of methods in terms of their accuracy, cost, and efficiency, but the general trends regarding the sets of methods are still very visible. Again, the two independent variables were highly significant in each model, and the amount of variance in dependent measure captured by the models ranged between 59% and 87%. For performance, Strong LM First and Strong LM Single Attempt were significantly higher than all others. In terms of cost, there were three separable groups, where the three Self-Consistency approaches, Prompt Reduction, and Strong Router were significantly higher than all others, followed by Plan, Instance level QA Pairs, Strong LM First, Strong LM Single Attempt, and Weak Router were significantly less expensive than those but significantly more expensive than the remaining methods. In terms of efficiency, Repo Structure is the best, but it is not significantly better than Weak LM First, Repo-Level QA Pairs, Strong LM Single Attempt, and Few shot (1 shot random and 1 shot similarity).

| Level                                 | Group     | Least Sq Mean |
|---------------------------------------|-----------|---------------|
| Repo Structure                        | A         | 0.02818333    |
| Weak LM First                         | A B       | 0.02365000    |
| Repo Level QA Pairs                   | A B C     | 0.02128333    |
| Strong LM Single Attempt              | A B C     | 0.02100000    |
| Few Shot Examples (1 Shot Random)     | A B C D   | 0.02028333    |
| Few Shot Examples (1 Shot Similarity) | A B C D E | 0.01988333    |
| Repo Summary                          | B C D E   | 0.01821667    |
| Strong LM First                       | B C D E F | 0.01640000    |
| Few Shot Examples (5 Shot Similarity) | C D E F G | 0.01501667    |
| Few Shot Examples (5 Shot Random)     | C D E F G | 0.01483333    |
| Weak Router                           | D E F G H | 0.01170000    |
| Prompt Reduction                      | E F G H   | 0.01135000    |
| Plan                                  | F G H     | 0.00801667    |
| Instance Level QA Pairs               | G H       | 0.00750000    |
| Best of N                             | H         | 0.00605000    |
| Strong Router                         | H         | 0.00585000    |
| Self-Consistency – Clustering         | H         | 0.00491667    |
| Self-Consistency – Majority Vote      | H         | 0.00486667    |
| Self-Consistency – Universal          | H         | 0.00475000    |

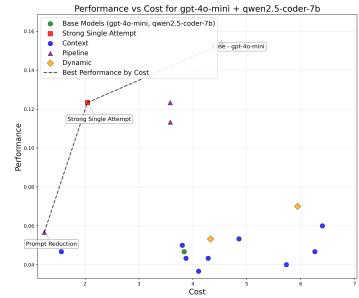
Table 2: Least Squares Mean Differences for Efficiency across Methods. Methods sharing a letter are not significantly different (Student’s t,  $\alpha = 0.05$ ,  $t = 1.98667$ ).



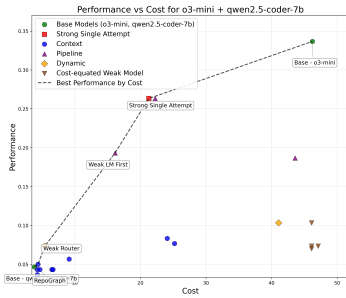
(a) O3-mini (0.3367) + GPT-4o-mini (0.1533)



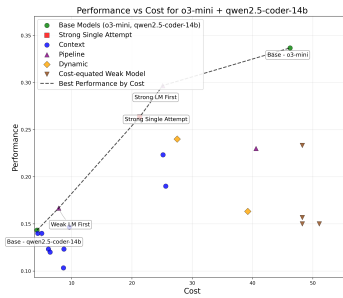
(b) O4-mini (0.4533) + GPT-4o-mini (0.1533)



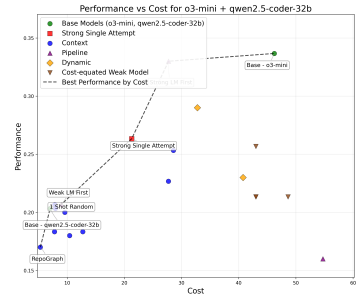
(c) GPT-4o-mini (0.1533) + Qwen2.5-Coder-7B (0.0467)



(d) O3-mini (0.3367) + Qwen2.5-Coder-7B (0.0467)



(e) O3-mini (0.3367) + Qwen2.5-Coder-14B (0.1433)



(f) O3-mini (0.3367) + Qwen2.5-Coder-32B (0.2033)

Figure 4: Performance vs. cost comparison across different Strong-Weak LM pairs, denoted as (Strong LM + Weak LM). The line denotes a monotonically increasing curve, i.e. what is the best method based on performance given a particular cost budget.

| Experiment                     | Resolution Rate | Avg. #iterations      | Valid Patch Rate | Total Generation Cost | Efficiency |
|--------------------------------|-----------------|-----------------------|------------------|-----------------------|------------|
| Base - o4-mini                 | 0.4533          | <b>1.32s + 0.00w</b>  | 0.993            | 33.328                | 0.0136     |
| Base - gpt-4o-mini             | 0.1533          | 0.00s + 2.09w         | 0.960            | 4.529                 | 0.0338     |
| Self-Consistency - Direct      | <b>0.1433</b>   | <b>0.00s + 15.58w</b> | <b>0.997</b>     | 35.307                | 0.0041     |
| Self-Consistency - Clustering  | 0.1500          | <b>0.00s + 15.58w</b> | <b>0.997</b>     | 35.307                | 0.0042     |
| Self-Consistency - Universal   | 0.1633          | <b>0.00s + 16.58w</b> | <b>0.997</b>     | 39.688                | 0.0041     |
| Best of n (cheating)           | 0.2167          | <b>0.00s + 15.58w</b> | <b>0.997</b>     | 35.307                | 0.0061     |
| Plan                           | 0.2933          | <b>1.00s + 2.43w</b>  | <b>0.897</b>     | 29.679                | 0.0099     |
| Instance Level QA Pairs        | 0.2567          | 1.00s + 2.08w         | <b>0.923</b>     | 25.598                | 0.0100     |
| 1 Shot Succesfull - Random     | <b>0.1367</b>   | <b>0.01s + 2.12w</b>  | 0.967            | 5.060                 | 0.0270     |
| 5 Shot Succesfull - Random     | <b>0.1400</b>   | 0.04s + 1.85w         | 0.977            | 5.793                 | 0.0242     |
| 1 Shot Succesfull - Similarity | 0.1567          | 0.01s + 1.99w         | <b>0.957</b>     | 4.796                 | 0.0327     |
| 5 Shot Succesfull - Similarity | <b>0.1367</b>   | <b>0.04s + 2.15w</b>  | <b>0.963</b>     | 6.520                 | 0.0210     |
| Repo Structure                 | <b>0.1367</b>   | 1.00s + 2.08w         | <b>0.930</b>     | 4.288                 | 0.0319     |
| Repo Level QA Pairs            | 0.1467          | 1.00s + 1.92w         | 0.980            | 4.699                 | 0.0312     |
| Repo Summary                   | <b>0.1433</b>   | 1.00s + 2.02w         | <b>0.957</b>     | 4.854                 | 0.0295     |
| Strong LM Single Attempt       | 0.3800          | 1.00s + 0.00w         | 0.793            | 23.616                | 0.0161     |
| Strong LM First                | 0.4167          | 1.00s + 0.72w         | 0.960            | 25.341                | 0.0164     |
| Prompt Reduction               | 0.2733          | 4.36s + 1.00w         | <b>0.753</b>     | <b>56.021</b>         | 0.0049     |
| Weak LM First                  | 0.1533          | 0.08s + 1.76w         | 0.977            | 6.061                 | 0.0253     |
| Weak Router                    | 0.2400          | <b>1.14s + 2.42w</b>  | 0.963            | 17.903                | 0.0134     |
| Strong Router                  | 0.2133          | 2.29s + 1.36w         | 0.967            | 36.342                | 0.0059     |

Table 3: Results with O4-mini as strong LM and GPT-4o-mini as weak LM. **Red** denotes drop. We consider a variance of 0.67% owing to the non-determinism introduced by the growing temperature values so a drop less than or equal to this is not marked in red.

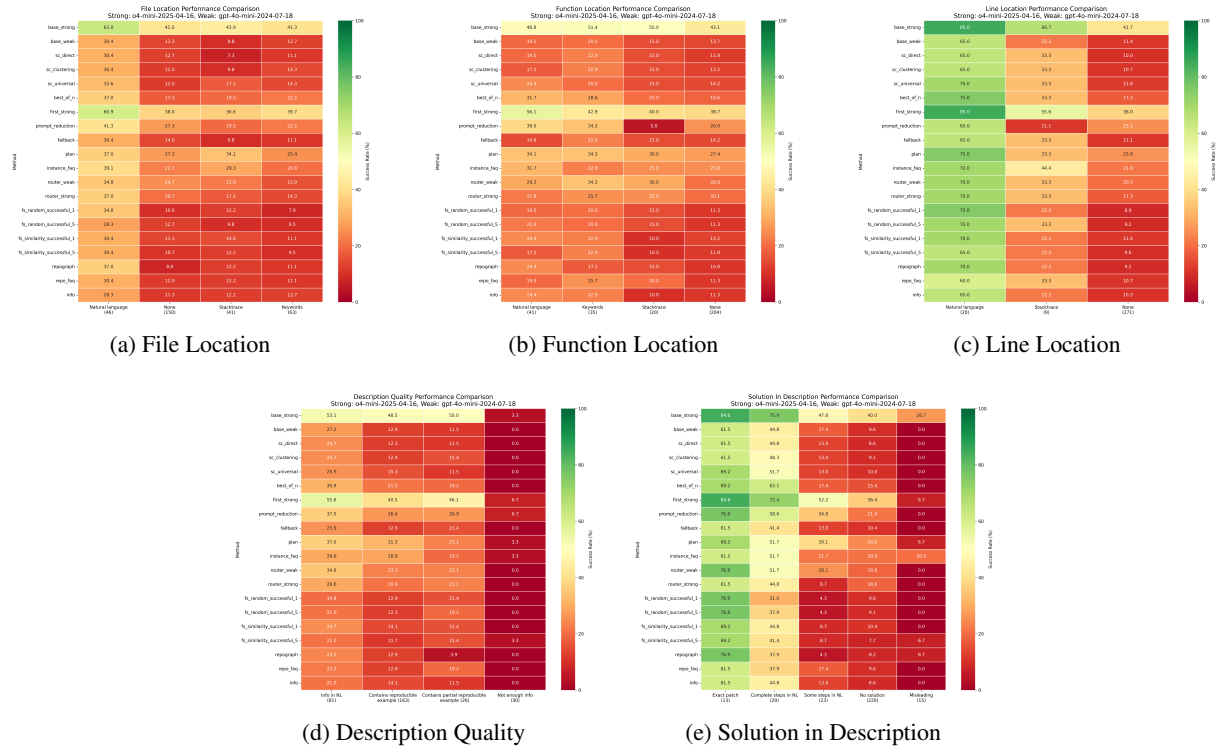


Figure 5: Heatmaps of issue category wise performance for O4-mini + GPT-4o-mini model pair.

| Experiment                     | Resolution Rate | Avg. #iterations      | Valid Patch Rate | Total Generation Cost | Efficiency    |
|--------------------------------|-----------------|-----------------------|------------------|-----------------------|---------------|
| Base - o3-mini                 | 0.3367          | <b>1.83s + 0.00w</b>  | <b>0.977</b>     | 46.224                | 0.0073        |
| Base - gpt-4o-mini             | 0.1533          | 0.00s + 2.09w         | 0.960            | 4.529                 | <b>0.0338</b> |
| Self-Consistency - Direct      | <b>0.1400</b>   | <b>0.00s + 19.48w</b> | 0.997            | 44.134                | 0.0032        |
| Self-Consistency - Clustering  | 0.1500          | <b>0.00s + 19.48w</b> | 0.997            | 44.134                | 0.0034        |
| Self-Consistency - Universal   | 0.1600          | <b>0.00s + 20.48w</b> | 0.997            | <b>48.524</b>         | 0.0033        |
| Best of n (cheating)           | 0.2200          | <b>0.00s + 19.48w</b> | 0.997            | 44.134                | 0.0050        |
| Plan                           | 0.2267          | 1.00s + 1.93w         | <b>0.957</b>     | 24.182                | 0.0094        |
| Instance Level QA Pairs        | 0.1800          | 1.00s + 2.01w         | <b>0.943</b>     | 25.699                | 0.0070        |
| 1 Shot Succesfull - Random     | <b>0.1333</b>   | <b>0.01s + 2.19w</b>  | <b>0.943</b>     | 5.323                 | 0.0250        |
| 3 Shot Succesfull - Random     | <b>0.1433</b>   | <b>0.03s + 2.21w</b>  | <b>0.940</b>     | 6.583                 | 0.0218        |
| 5 Shot Succesfull - Random     | <b>0.1267</b>   | 0.05s + 2.03w         | 0.963            | 7.127                 | 0.0178        |
| 1 Shot Succesfull - Similarity | <b>0.1433</b>   | <b>0.01s + 2.46w</b>  | <b>0.923</b>     | 6.081                 | 0.0236        |
| 3 Shot Succesfull - Similarity | <b>0.1367</b>   | 0.03s + 2.01w         | <b>0.950</b>     | 6.093                 | 0.0224        |
| 5 Shot Succesfull - Similarity | 0.1500          | <b>0.05s + 2.14w</b>  | <b>0.947</b>     | 7.320                 | 0.0205        |
| 1 Shot All - Random            | 0.1467          | <b>0.00s + 2.35w</b>  | <b>0.927</b>     | 5.716                 | 0.0257        |
| 3 Shot All - Random            | <b>0.1333</b>   | <b>0.01s + 2.19w</b>  | <b>0.937</b>     | 5.366                 | 0.0248        |
| 5 Shot All - Random            | 0.1500          | <b>0.02s + 2.10w</b>  | <b>0.950</b>     | 5.712                 | 0.0263        |
| 1 Shot All - Similarity        | <b>0.1200</b>   | <b>0.00s + 2.44w</b>  | <b>0.923</b>     | 5.627                 | 0.0213        |
| 3 Shot All - Similarity        | <b>0.1367</b>   | <b>0.01s + 2.37w</b>  | <b>0.930</b>     | 5.959                 | 0.0229        |
| 5 Shot All - Similarity        | 0.1500          | <b>0.02s + 2.19w</b>  | 0.960            | 5.818                 | 0.0258        |
| Repo Structure                 | <b>0.1367</b>   | 1.00s + 2.08w         | <b>0.930</b>     | 4.288                 | 0.0319        |
| Repo Level QA Pairs            | <b>0.1267</b>   | 1.00s + 1.86w         | 0.977            | 4.686                 | 0.0270        |
| Repo Summary                   | 0.1633          | 1.00s + 2.07w         | 0.960            | 5.000                 | 0.0327        |
| Strong LM Single Attempt       | 0.2633          | 1.00s + 0.00w         | <b>0.690</b>     | 21.280                | 0.0124        |
| Strong LM First                | 0.3000          | 1.00s + 1.17w         | <b>0.930</b>     | 24.922                | 0.0120        |
| Prompt Reduction               | 0.2033          | 4.73s + 1.00w         | <b>0.653</b>     | 42.537                | 0.0048        |
| Weak LM First                  | 0.1633          | 0.08s + 1.61w         | 0.990            | 5.687                 | 0.0287        |
| Weak Router                    | 0.2267          | 0.48s + 2.30w         | 0.977            | 16.764                | 0.0135        |
| Strong Router                  | 0.1867          | 1.71s + 1.22w         | 0.977            | 36.583                | 0.0051        |

Table 4: Results with O3-mini as strong LM and GPT-4o-mini as weak LM. **Red** denotes drop. We consider a variance of 0.67% owing to the non-determinism introduced by the growing temperature values so a drop less than or equal to this is not marked in red.



| Experiment                     | Resolution Rate | Avg. #iterations      | Valid Patch Rate | Total Generation Cost | Efficiency    |
|--------------------------------|-----------------|-----------------------|------------------|-----------------------|---------------|
| Base - o3-mini                 | 0.3367          | <b>1.83s + 0.00w</b>  | <b>0.977</b>     | 46.224                | 0.0073        |
| Base - qwen2.5-coder-7b        | 0.0467          | 0.00s + 6.48w         | 0.600            | 3.839                 | 0.0122        |
| Self-Consistency - Direct      | 0.0733          | <b>0.00s + 78.38w</b> | 0.877            | 46.090                | 0.0016        |
| Self-Consistency - Clustering  | 0.0700          | <b>0.00s + 78.38w</b> | 0.877            | 46.090                | 0.0015        |
| Self-Consistency - Universal   | 0.0733          | <b>0.00s + 79.38w</b> | 0.877            | <b>47.095</b>         | 0.0016        |
| Best of n (cheating)           | 0.1033          | <b>0.00s + 78.38w</b> | 0.877            | 46.090                | 0.0022        |
| Plan                           | 0.0833          | <b>1.00s + 6.79w</b>  | <b>0.517</b>     | 24.092                | 0.0035        |
| Instance Level QA Pairs        | 0.0767          | 1.00s + 6.95w         | <b>0.563</b>     | 25.203                | 0.0030        |
| 1 Shot Succesfull - Random     | 0.0433          | <b>0.01s + 6.49w</b>  | 0.617            | 4.281                 | 0.0101        |
| 5 Shot Succesfull - Random     | 0.0433          | <b>0.05s + 6.61w</b>  | <b>0.590</b>     | 6.501                 | 0.0067        |
| 1 Shot Succesfull - Similarity | <b>0.0367</b>   | <b>0.01s + 6.63w</b>  | 0.603            | 4.282                 | 0.0086        |
| 5 Shot Succesfull - Similarity | 0.0433          | <b>0.05s + 6.86w</b>  | <b>0.570</b>     | 6.665                 | 0.0065        |
| Repo Structure                 | 0.0500          | <b>1.00s + 7.32w</b>  | <b>0.487</b>     | 4.388                 | 0.0114        |
| Repo Level QA Pairs            | 0.0433          | <b>1.00s + 6.75w</b>  | <b>0.577</b>     | 4.676                 | 0.0093        |
| Repo Summary                   | 0.0567          | <b>1.00s + 6.70w</b>  | <b>0.587</b>     | 9.139                 | 0.0062        |
| Strong LM Single Attempt       | 0.2633          | 1.00s + 0.00w         | <b>0.690</b>     | 21.280                | <b>0.0124</b> |
| Strong LM First                | 0.2633          | 1.00s + 2.66w         | 0.773            | 22.261                | 0.0118        |
| Prompt Reduction               | 0.1867          | 4.56s + 1.00w         | 0.663            | 43.590                | 0.0043        |
| Weak LM First                  | 0.1933          | 0.62s + 3.82w         | 0.810            | 16.147                | 0.0120        |
| Weak Router                    | 0.0733          | <b>0.07s + 7.42w</b>  | 0.607            | 5.432                 | 0.0135        |
| Strong Router                  | 0.1033          | 1.77s + 3.72w         | 0.820            | 41.081                | 0.0025        |

Table 5: Results with O3-mini as strong LM and Qwen2.5-Coder-7B as weak LM. **Red** denotes drop. We consider a variance of 0.67% owing to the non-determinism introduced by the growing temperature values so a drop less than or equal to this is not marked in red.

| Experiment                     | Resolution Rate | Avg. #iterations      | Valid Patch Rate | Total Generation Cost | Efficiency    |
|--------------------------------|-----------------|-----------------------|------------------|-----------------------|---------------|
| Base - o3-mini                 | 0.3367          | <b>1.83s + 0.00w</b>  | 0.977            | 46.224                | 0.0073        |
| Base - qwen2.5-coder-14b       | 0.1433          | 0.00s + 2.91w         | 0.937            | 4.257                 | <b>0.0337</b> |
| Self-Consistency - Direct      | 0.1500          | <b>0.00s + 32.59w</b> | <b>0.987</b>     | 48.249                | 0.0031        |
| Self-Consistency - Clustering  | 0.1567          | <b>0.00s + 32.59w</b> | <b>0.987</b>     | 48.249                | 0.0032        |
| Self-Consistency - Universal   | 0.1500          | <b>0.00s + 33.59w</b> | <b>0.987</b>     | <b>51.069</b>         | 0.0029        |
| Best of n (cheating)           | 0.2333          | <b>0.00s + 32.59w</b> | <b>0.987</b>     | 48.249                | 0.0048        |
| Plan                           | 0.2233          | <b>1.00s + 3.21w</b>  | <b>0.893</b>     | 25.158                | 0.0089        |
| Instance Level QA Pairs        | 0.1900          | 1.00s + 2.82w         | <b>0.927</b>     | 25.599                | 0.0074        |
| 1 Shot Succesfull - Random     | <b>0.1233</b>   | <b>0.01s + 3.78w</b>  | <b>0.863</b>     | 6.140                 | 0.0201        |
| 5 Shot Succesfull - Random     | <b>0.1033</b>   | <b>0.05s + 4.15w</b>  | <b>0.830</b>     | 8.641                 | 0.0120        |
| 1 Shot Succesfull - Similarity | <b>0.1200</b>   | <b>0.01s + 3.92w</b>  | <b>0.880</b>     | 6.427                 | 0.0187        |
| 5 Shot Succesfull - Similarity | <b>0.1233</b>   | <b>0.05s + 3.97w</b>  | <b>0.860</b>     | 8.713                 | 0.0142        |
| Repo Structure                 | 0.1400          | <b>1.00s + 3.04w</b>  | <b>0.880</b>     | 4.393                 | 0.0319        |
| Repo Level QA Pairs            | 0.1400          | <b>1.00s + 2.99w</b>  | <b>0.907</b>     | 5.036                 | 0.0278        |
| Repo Summary                   | 0.1467          | <b>1.00s + 3.12w</b>  | <b>0.930</b>     | 9.609                 | 0.0153        |
| Strong LM Single Attempt       | 0.2633          | 1.00s + 0.00w         | <b>0.690</b>     | 21.280                | 0.0124        |
| Strong LM First                | 0.2967          | 1.00s + 2.20w         | <b>0.830</b>     | 25.099                | 0.0118        |
| Prompt Reduction               | 0.2300          | 4.36s + 1.00w         | <b>0.707</b>     | 40.568                | 0.0057        |
| Weak LM First                  | 0.1667          | 0.20s + 2.72w         | <b>0.913</b>     | 7.847                 | 0.0212        |
| Weak Router                    | 0.2400          | 1.34s + 2.45w         | 0.947            | 27.507                | 0.0087        |
| Strong Router                  | 0.1633          | 2.32s + 1.68w         | 0.937            | 39.201                | 0.0042        |

Table 6: Results with O3-mini as strong LM and Qwen2.5-Coder-14B as weak LM. **Red** denotes drop. We consider a variance of 0.67% owing to the non-determinism introduced by the growing temperature values so a drop less than or equal to this is not marked in red.

| Experiment                     | Resolution Rate | Avg. #iterations      | Valid Patch Rate | Total Generation Cost | Efficiency    |
|--------------------------------|-----------------|-----------------------|------------------|-----------------------|---------------|
| Base - o3-mini                 | 0.3367          | <b>1.83s + 0.00w</b>  | 0.977            | 46.224                | 0.0073        |
| Base - qwen2.5-coder-32b       | 0.2033          | 0.00s + 2.35w         | 0.967            | 7.021                 | 0.0290        |
| Self-Consistency - Direct      | 0.2133          | <b>0.00s + 12.98w</b> | <b>0.987</b>     | 43.011                | 0.0050        |
| Self-Consistency - Clustering  | 0.2133          | <b>0.00s + 12.98w</b> | <b>0.987</b>     | 43.011                | 0.0050        |
| Self-Consistency - Universal   | 0.2133          | <b>0.00s + 13.98w</b> | <b>0.987</b>     | <b>48.634</b>         | 0.0044        |
| Best of n                      | 0.2567          | <b>0.00s + 12.98w</b> | <b>0.987</b>     | 43.011                | 0.0060        |
| Plan                           | 0.2533          | <b>1.00s + 2.64w</b>  | <b>0.913</b>     | 28.591                | 0.0089        |
| Instance Level QA Pairs        | 0.2267          | 1.00s + 2.22w         | <b>0.957</b>     | 27.700                | 0.0082        |
| 1 Shot Succesfull - Random     | 0.2067          | <b>0.01s + 2.38w</b>  | <b>0.957</b>     | 7.843                 | 0.0264        |
| 5 Shot Succesfull - Random     | <b>0.1800</b>   | <b>0.05s + 2.35w</b>  | <b>0.940</b>     | 10.426                | 0.0173        |
| 1 Shot Succesfull - Similarity | 0.2033          | <b>0.01s + 2.46w</b>  | <b>0.947</b>     | 7.933                 | 0.0256        |
| 5 Shot Succesfull - Similarity | 0.2000          | 0.05s + 2.34w         | 0.967            | 9.583                 | 0.0209        |
| Repo Structure                 | <b>0.1700</b>   | <b>1.00s + 2.63w</b>  | <b>0.887</b>     | 5.302                 | <b>0.0321</b> |
| Repo Level QA Pairs            | <b>0.1833</b>   | 1.00s + 2.33w         | <b>0.950</b>     | 7.785                 | 0.0235        |
| Repo Summary                   | <b>0.1833</b>   | <b>1.00s + 2.47w</b>  | <b>0.933</b>     | 12.730                | 0.0144        |
| Strong LM Single Attempt       | 0.2633          | 1.00s + 0.00w         | <b>0.690</b>     | 21.280                | 0.0124        |
| Strong LM First                | 0.3300          | 1.00s + 1.80w         | <b>0.870</b>     | 27.737                | 0.0119        |
| Prompt Reduction               | <b>0.1600</b>   | 5.63s + 1.00w         | <b>0.557</b>     | 54.735                | 0.0029        |
| Weak LM First                  | 0.2067          | 0.12s + 2.31w         | <b>0.930</b>     | 8.9637                | 0.0231        |
| Weak Router                    | 0.2900          | 1.42s + 1.83w         | 0.970            | 32.772                | 0.0088        |
| Strong Router                  | 0.2300          | 2.31s + 1.50w         | 0.973            | 40.763                | 0.0056        |

Table 7: Results with O3-mini as strong LM and Qwen2.5-Coder-32B as weak LM. **Red** denotes drop. We consider a variance of 0.67% owing to the non-determinism introduced by the growing temperature values so a drop less than or equal to this is not marked in red.

| Experiment                     | Resolution Rate | Avg. #iterations     | Valid Patch Rate | Total Generation Cost | Efficiency    |
|--------------------------------|-----------------|----------------------|------------------|-----------------------|---------------|
| Base - gpt-4o-mini             | <b>0.1533</b>   | <b>2.09s + 0.00w</b> | <b>0.960</b>     | 4.529                 | 0.0338        |
| Base - qwen2.5-coder-7b        | 0.0467          | 0.00s + 6.48w        | 0.600            | 3.839                 | 0.0122        |
| Self-Consistency - Direct      | 0.0467          | 0.00s + 6.48w        | 0.600            | 3.839                 | 0.0122        |
| Self-Consistency - Clustering  | 0.0467          | 0.00s + 6.48w        | 0.600            | 3.839                 | 0.0122        |
| Self-Consistency - Universal   | 0.0467          | 0.00s + 6.48w        | 0.600            | 3.839                 | 0.0122        |
| Best of n (cheating)           | 0.0467          | 0.00s + 6.48w        | 0.600            | 3.839                 | 0.0122        |
| Plan                           | 0.0467          | <b>1.00s + 7.14w</b> | <b>0.483</b>     | 6.262                 | 0.0075        |
| Instance Level QA Pairs        | 0.0600          | <b>1.00s + 6.99w</b> | <b>0.533</b>     | <b>6.402</b>          | 0.0094        |
| 1 Shot Succesfull - Random     | 0.0500          | <b>0.02s + 6.25w</b> | 0.633            | 3.805                 | 0.0131        |
| 5 Shot Succesfull - Random     | 0.0533          | <b>0.10s + 6.90w</b> | <b>0.547</b>     | 4.854                 | 0.0110        |
| 1 Shot Succesfull - Similarity | 0.0433          | <b>0.02s + 6.58w</b> | <b>0.590</b>     | 4.284                 | 0.0101        |
| 5 Shot Succesfull - Similarity | 0.0400          | <b>0.10s + 6.79w</b> | <b>0.553</b>     | 5.733                 | 0.0070        |
| Repo Structure                 | 0.0467          | <b>1.00s + 7.42w</b> | <b>0.483</b>     | 1.562                 | 0.0299        |
| Repo Level QA Pairs            | <b>0.0367</b>   | <b>1.00s + 6.73w</b> | <b>0.563</b>     | 4.105                 | 0.0089        |
| Repo Summary                   | 0.0433          | <b>1.00s + 6.63w</b> | <b>0.570</b>     | 3.873                 | 0.0112        |
| Strong LM Single Attempt       | 0.1233          | 1.00s + 0.00w        | 0.693            | 2.046                 | <b>0.0603</b> |
| Strong LM First                | 0.1233          | 1.00s + 3.28w        | 0.770            | 3.579                 | 0.0344        |
| Prompt Reduction               | 0.0567          | 4.83s + 1.00w        | 0.647            | 1.246                 | 0.0455        |
| Weak LM First                  | 0.1133          | 0.18s + 2.86w        | 0.823            | 3.581                 | 0.0316        |
| Weak Router                    | 0.0533          | <b>0.62s + 7.26w</b> | 0.623            | 4.326                 | 0.0123        |
| Strong Router                  | 0.0700          | 1.72s + 5.14w        | 0.723            | 5.941                 | 0.0118        |

Table 8: Results with GPT-4o-mini as strong LM and Qwen2.5-Coder-7B as weak LM. **Red** denotes drop. We consider a variance of 0.67% owing to the non-determinism introduced by the growing temperature values so a drop less than or equal to this is not marked in red.

| Method                          | File Precision | File Recall | File F1 | Module Precision | Module Recall | Module F1 | Line Precision | Line Recall | Line F1 |
|---------------------------------|----------------|-------------|---------|------------------|---------------|-----------|----------------|-------------|---------|
| Base Strong                     | 84.23          | 84.23       | 84.23   | 84.23            | 84.23         | 84.23     | 55.13          | 47.33       | 48.43   |
| Base Weak                       | 78.65          | 78.82       | 78.7    | 78.65            | 78.82         | 78.7      | 35.92          | 28.49       | 29.97   |
| Self-Consistency - Direct       | 76.92          | 76.92       | 76.92   | 76.92            | 76.92         | 76.92     | 35.56          | 27.74       | 29.21   |
| Self-Consistency - Clustering   | 76.09          | 76.25       | 76.14   | 76.09            | 76.25         | 76.14     | 34.92          | 27.99       | 29.35   |
| Self-Consistency - Universal    | 77.42          | 77.59       | 77.48   | 77.42            | 77.59         | 77.48     | 37.36          | 29.22       | 31.05   |
| Best of N                       | 76.25          | 76.25       | 76.25   | 76.25            | 76.25         | 76.25     | 36.55          | 29.11       | 30.34   |
| Strong LM Single Attempt        | 80.5           | 81.6        | 80.84   | 80.5             | 81.6          | 80.84     | 50.94          | 44.05       | 44.76   |
| Prompt Reduction                | 85.18          | 85.4        | 85.25   | 85.18            | 85.4          | 85.25     | 51.1           | 40.03       | 42.36   |
| Weak LM First                   | 77.82          | 77.82       | 77.82   | 77.82            | 77.82         | 77.82     | 37.4           | 30.17       | 31.41   |
| Plan                            | 85.5           | 86.62       | 85.87   | 85.5             | 86.62         | 85.87     | 49.7           | 44.09       | 43.99   |
| Instance Level QA Pairs         | 83.57          | 84.12       | 83.75   | 83.57            | 84.12         | 83.75     | 49.5           | 40.7        | 41.74   |
| Weak Router                     | 79.87          | 80.28       | 79.99   | 79.87            | 80.28         | 79.99     | 42.78          | 37.03       | 37.43   |
| Strong Router                   | 80.17          | 80.34       | 80.23   | 80.17            | 80.34         | 80.23     | 43.08          | 36.01       | 36.91   |
| 1 Shot Successfull - Random     | 77.93          | 77.93       | 77.93   | 77.93            | 77.93         | 77.93     | 34.93          | 25.86       | 28.02   |
| 5 Shot Successfull - Random     | 78.16          | 78.16       | 78.16   | 78.16            | 78.16         | 78.16     | 36.14          | 28.47       | 29.87   |
| 1 Shot Successfull - Similarity | 78.22          | 78.4        | 78.28   | 78.22            | 78.4          | 78.28     | 35.89          | 28.8        | 29.95   |
| 5 Shot Successfull - Similarity | 78.22          | 78.2        | 78.2    | 78.2             | 78.2          | 78.2      | 38.7           | 29.47       | 31.64   |
| Repo Structure                  | 71.51          | 71.68       | 71.57   | 71.51            | 71.68         | 71.57     | 36.18          | 27.99       | 29.41   |
| Repo Level QA Pairs             | 77.21          | 77.21       | 77.21   | 77.21            | 77.21         | 77.21     | 36.83          | 28.86       | 30.5    |
| Repo Summary                    | 76.25          | 76.66       | 76.36   | 76.25            | 76.66         | 76.36     | 36.24          | 28.64       | 30.23   |

Table 9: Localization performance for O4-mini as strong LM and GPT-4o-mini as weak LM. Cells highlighted in red depict lower score than the base weak LM setting.

| Method                          | File Precision | File Recall | File F1 | Module Precision | Module Recall | Module F1 | Line Precision | Line Recall | Line F1 |
|---------------------------------|----------------|-------------|---------|------------------|---------------|-----------|----------------|-------------|---------|
| Base Strong                     | 76.62          | 76.79       | 76.68   | 76.62            | 76.79         | 76.68     | 45.53          | 37.83       | 39.31   |
| Base Weak                       | 78.65          | 78.82       | 78.7    | 78.65            | 78.82         | 78.7      | 35.92          | 28.49       | 29.97   |
| Self-Consistency - Direct       | 76.25          | 76.25       | 76.25   | 76.25            | 76.25         | 76.25     | 35.44          | 28.03       | 29.36   |
| Self-Consistency - Clustering   | 75.75          | 75.92       | 75.81   | 75.75            | 75.92         | 75.81     | 35.63          | 28.31       | 29.76   |
| Self-Consistency - Universal    | 76.25          | 76.25       | 76.25   | 76.25            | 76.25         | 76.25     | 35.44          | 28.03       | 29.36   |
| Best of N                       | 76.25          | 76.25       | 76.25   | 76.25            | 76.25         | 76.25     | 36.55          | 29.11       | 30.34   |
| Strong LM Single Attempt        | 77.42          | 77.78       | 77.54   | 77.42            | 77.78         | 77.54     | 47.64          | 39.69       | 40.8    |
| Prompt Reduction                | 83.16          | 83.67       | 83.33   | 83.16            | 83.67         | 83.33     | 49.25          | 39.14       | 41.03   |
| Weak LM First                   | 77.61          | 77.78       | 77.67   | 77.61            | 77.78         | 77.67     | 36.98          | 30.57       | 31.45   |
| Plan                            | 76.83          | 77.7        | 77.12   | 76.83            | 77.7          | 77.12     | 42.43          | 35.5        | 36.11   |
| Instance Level QA Pairs         | 77.3           | 78.45       | 77.64   | 77.3             | 78.45         | 77.64     | 40.76          | 31.68       | 33.57   |
| Weak Router                     | 77.82          | 77.82       | 77.82   | 77.82            | 77.82         | 77.82     | 39.14          | 34.35       | 34.48   |
| Strong Router                   | 73.04          | 73.38       | 73.15   | 73.04            | 73.38         | 73.15     | 36.1           | 30.54       | 31      |
| 1 Shot Successfull - Random     | 77.03          | 77.39       | 77.15   | 77.03            | 77.39         | 77.15     | 34.57          | 27.75       | 28.94   |
| 5 Shot Successfull - Random     | 74.05          | 74.05       | 74.05   | 74.05            | 74.05         | 74.05     | 34.8           | 26.91       | 28.44   |
| 1 Shot Successfull - Similarity | 75.71          | 75.89       | 75.77   | 75.71            | 75.89         | 75.77     | 33.72          | 25.96       | 27.34   |
| 5 Shot Successfull - Similarity | 77.78          | 77.78       | 77.78   | 77.78            | 77.78         | 77.78     | 35.61          | 27.96       | 29.25   |
| Repo Structure                  | 71.51          | 71.68       | 71.57   | 71.51            | 71.68         | 71.57     | 36.18          | 27.99       | 29.41   |
| Repo Level QA Pairs             | 75.6           | 75.77       | 75.65   | 75.6             | 75.77         | 75.65     | 36.48          | 29.27       | 30.43   |
| Repo Summary                    | 77.95          | 78.12       | 78.01   | 77.95            | 78.12         | 78.01     | 36.31          | 29.31       | 30.26   |

Table 10: Localization performance for O3-mini as strong LM and GPT-4o-mini as weak LM. Cells highlighted in red depict lower score than the base weak LM setting.

| Method                          | File Precision | File Recall | File F1 | Module Precision | Module Recall | Module F1 | Line Precision | Line Recall | Line F1 |
|---------------------------------|----------------|-------------|---------|------------------|---------------|-----------|----------------|-------------|---------|
| Base Strong                     | 76.62          | 76.79       | 76.68   | 76.62            | 76.79         | 76.68     | 45.53          | 37.83       | 39.31   |
| Base Weak                       | 76.11          | 76.11       | 76.11   | 76.11            | 76.11         | 76.11     | 24.56          | 19.66       | 19.95   |
| Self-Consistency - Direct       | 71.48          | 71.48       | 71.48   | 71.48            | 71.48         | 71.48     | 23.65          | 17.86       | 18.77   |
| Self-Consistency - Clustering   | 72.24          | 72.24       | 72.24   | 72.24            | 72.24         | 72.24     | 23.61          | 18.02       | 18.9    |
| Self-Consistency - Universal    | 71.48          | 71.48       | 71.48   | 71.48            | 71.48         | 71.48     | 23.65          | 17.86       | 18.77   |
| Best of N                       | 71.86          | 71.86       | 71.86   | 71.86            | 71.86         | 71.86     | 24.08          | 18.62       | 19.36   |
| Strong LM Single Attempt        | 79.31          | 79.74       | 79.45   | 79.31            | 79.74         | 79.45     | 47.91          | 40.23       | 41.19   |
| Prompt Reduction                | 75.04          | 75.88       | 75.29   | 75.04            | 75.88         | 75.29     | 42.84          | 34.8        | 36.01   |
| Weak LM First                   | 77.16          | 77.37       | 77.23   | 77.16            | 77.37         | 77.23     | 36.93          | 31.54       | 31.44   |
| Plan                            | 77.74          | 78.06       | 77.85   | 77.74            | 78.06         | 77.85     | 39.95          | 33.04       | 33.55   |
| Instance Level QA Pairs         | 81.07          | 81.07       | 81.07   | 81.07            | 81.07         | 81.07     | 35.08          | 27.67       | 28.41   |
| Weak Router                     | 77.47          | 77.47       | 77.47   | 77.47            | 77.47         | 77.47     | 26.06          | 22.34       | 22.47   |
| Strong Router                   | 69.92          | 70.33       | 70.05   | 69.92            | 70.33         | 70.05     | 26.3           | 21.71       | 22.35   |
| 1 Shot Successfull - Random     | 74.73          | 74.73       | 74.73   | 74.73            | 74.73         | 74.73     | 22.32          | 16.92       | 17.54   |
| 5 Shot Successfull - Random     | 74.01          | 74.01       | 74.01   | 74.01            | 74.01         | 74.01     | 19.89          | 17          | 16.92   |
| 1 Shot Successfull - Similarity | 67.96          | 67.96       | 67.96   | 67.96            | 67.96         | 67.96     | 22.29          | 16.2        | 17.9    |
| 5 Shot Successfull - Similarity | 71.93          | 71.93       | 71.93   | 71.93            | 71.93         | 71.93     | 24.24          | 19.16       | 20.12   |
| Repo Structure                  | 69.86          | 69.86       | 69.86   | 69.86            | 69.86         | 69.86     | 32.96          | 25.97       | 27.28   |
| Repo Level QA Pairs             | 73.12          | 73.41       | 73.22   | 73.12            | 73.41         | 73.22     | 23.76          | 18.12       | 18.87   |
| Repo Summary                    | 73.48          | 73.86       | 73.58   | 73.48            | 73.86         | 73.58     | 24.31          | 19.55       | 20.01   |

Table 11: Localization performance for O3-mini as strong LM and Qwen2.5-Coder-7B as weak LM. Cells highlighted in red depict lower score than the base weak LM setting.

| Method                         | File Precision | File Recall | File F1 | Module Precision | Module Recall | Module F1 | Line Precision | Line Recall | Line F1 |
|--------------------------------|----------------|-------------|---------|------------------|---------------|-----------|----------------|-------------|---------|
| Base Strong                    | 76.62          | 76.79       | 76.68   | 76.62            | 76.79         | 76.68     | 45.53          | 37.83       | 39.31   |
| Base Weak                      | 74.67          | 75.8        | 75.03   | 74.67            | 75.8          | 75.03     | 37.41          | 30.85       | 32.01   |
| Self-Consistency - Direct      | 76.35          | 76.35       | 76.35   | 76.35            | 76.35         | 76.35     | 38.41          | 29.98       | 32      |
| Self-Consistency - Clustering  | 76.13          | 76.35       | 76.18   | 76.13            | 76.35         | 76.18     | 38.13          | 30.16       | 31.91   |
| Self-Consistency - Universal   | 76.35          | 76.35       | 76.35   | 76.35            | 76.35         | 76.35     | 38.41          | 29.98       | 32      |
| Best of N                      | 75             | 75          | 75      | 75               | 75            | 75        | 37.24          | 29.15       | 30.97   |
| Strong LM Single Attempt       | 83.13          | 83.53       | 83.27   | 83.13            | 83.53         | 83.27     | 51.72          | 42.03       | 43.96   |
| Prompt Reduction               | 77.59          | 78.3        | 77.83   | 77.59            | 78.3          | 77.83     | 43.74          | 35.83       | 37.21   |
| Weak LM First                  | 76.64          | 77.37       | 76.89   | 76.64            | 77.37         | 76.89     | 38.25          | 31.28       | 32.62   |
| Plan                           | 74.22          | 76.12       | 74.78   | 74.22            | 76.12         | 74.78     | 42.65          | 35.47       | 36.35   |
| Instance Level QA Pairs        | 79.26          | 80.94       | 79.8    | 79.26            | 80.94         | 79.8      | 42.5           | 36.41       | 37.05   |
| Weak Router                    | 77.46          | 77.82       | 77.58   | 77.46            | 77.82         | 77.58     | 40.91          | 34.46       | 35.13   |
| Strong Router                  | 75.09          | 75.8        | 75.33   | 75.09            | 75.8          | 75.33     | 37.09          | 30.98       | 31.79   |
| 1 Shot Succesfull - Random     | 76.77          | 77.61       | 77.03   | 76.77            | 77.61         | 77.03     | 31.97          | 25.69       | 26.77   |
| 5 Shot Succesfull - Random     | 72.82          | 73.49       | 73.03   | 72.82            | 73.49         | 73.03     | 31.04          | 23.6        | 24.89   |
| 1 Shot Succesfull - Similarity | 72.92          | 73.48       | 73.11   | 72.92            | 73.48         | 73.11     | 31.38          | 24.39       | 25.89   |
| 5 Shot Succesfull - Similarity | 72.16          | 72.87       | 72.35   | 72.16            | 72.87         | 72.35     | 32.34          | 25.15       | 26.51   |
| Repo Structure                 | 69.7           | 70.45       | 69.95   | 69.7             | 70.45         | 69.95     | 37.79          | 30.11       | 31.57   |
| Repo Level QA Pairs            | 77.39          | 77.94       | 77.57   | 77.39            | 77.94         | 77.57     | 34.48          | 28.5        | 29.36   |
| Repo Summary                   | 75.09          | 75.27       | 75.15   | 75.09            | 75.27         | 75.15     | 34.87          | 28.8        | 29.62   |

Table 12: Localization performance for O3-mini as strong LM and Qwen2.5-Coder-14B as weak LM. Cells highlighted in red depict lower score than the base weak LM setting.

| Method                         | File Precision | File Recall | File F1 | Module Precision | Module Recall | Module F1 | Line Precision | Line Recall | Line F1 |
|--------------------------------|----------------|-------------|---------|------------------|---------------|-----------|----------------|-------------|---------|
| Base Strong                    | 76.62          | 76.79       | 76.68   | 76.62            | 76.79         | 76.68     | 45.53          | 37.83       | 39.31   |
| Base Weak                      | 76.9           | 78.28       | 77.36   | 76.9             | 78.28         | 77.36     | 41.65          | 34.76       | 35.46   |
| Self-Consistency - Direct      | 78.63          | 79.73       | 78.96   | 78.63            | 79.73         | 78.96     | 42.58          | 34.75       | 35.76   |
| Self-Consistency - Clustering  | 79.31          | 80.41       | 79.64   | 79.31            | 80.41         | 79.64     | 43.16          | 35.34       | 36.33   |
| Self-Consistency - Universal   | 78.63          | 79.73       | 78.96   | 78.63            | 79.73         | 78.96     | 42.58          | 34.75       | 35.76   |
| Best of N                      | 78.63          | 80.07       | 79.08   | 78.63            | 80.07         | 79.08     | 42.78          | 34.72       | 35.87   |
| Strong LM Single Attempt       | 79.31          | 79.31       | 79.31   | 79.31            | 79.31         | 79.31     | 51.91          | 43.96       | 45.32   |
| Prompt Reduction               | 73.35          | 73.65       | 73.45   | 73.35            | 73.65         | 73.45     | 46.04          | 36.86       | 38.79   |
| Weak LM First                  | 76.7           | 77.06       | 76.82   | 76.7             | 77.06         | 76.82     | 42.05          | 34.32       | 35.48   |
| Plan                           | 75.55          | 79.2        | 76.76   | 75.55            | 79.2          | 76.76     | 44.5           | 39.48       | 39.15   |
| Instance Level QA Pairs        | 76.54          | 78.05       | 77      | 76.54            | 78.05         | 77        | 45.86          | 39.95       | 39.82   |
| Weak Router                    | 75.6           | 75.95       | 75.72   | 75.6             | 75.95         | 75.72     | 47.65          | 37.9        | 40.02   |
| Strong Router                  | 74.23          | 75.34       | 74.57   | 74.23            | 75.34         | 74.57     | 42.13          | 35.18       | 36.14   |
| 1 Shot Succesfull - Random     | 78.86          | 80.14       | 79.27   | 78.86            | 80.14         | 79.27     | 42.43          | 35.69       | 36.2    |
| 5 Shot Succesfull - Random     | 75.53          | 76.6        | 75.89   | 75.53            | 76.6          | 75.89     | 41.6           | 34.84       | 35.34   |
| 1 Shot Succesfull - Similarity | 75.95          | 77.11       | 76.29   | 75.95            | 77.11         | 76.29     | 43.71          | 36.77       | 37.27   |
| 5 Shot Succesfull - Similarity | 76.79          | 77.59       | 77.01   | 76.79            | 77.59         | 77.01     | 41.71          | 35.02       | 35.48   |
| Repo Structure                 | 71.99          | 72.93       | 72.31   | 71.99            | 72.93         | 72.31     | 40.63          | 34.8        | 35.62   |
| Repo Level QA Pairs            | 77.49          | 79.65       | 78.19   | 77.49            | 79.65         | 78.19     | 41.87          | 37.01       | 36.43   |
| Repo Summary                   | 75.83          | 77.14       | 76.25   | 75.83            | 77.14         | 76.25     | 41.64          | 34.88       | 35.3    |

Table 13: Localization performance for O3-mini as strong LM and Qwen2.5-Coder-32B as weak LM. Cells highlighted in red depict lower score than the base weak LM setting.

| Method                         | File Precision | File Recall | File F1 | Module Precision | Module Recall | Module F1 | Line Precision | Line Recall | Line F1 |
|--------------------------------|----------------|-------------|---------|------------------|---------------|-----------|----------------|-------------|---------|
| Base Strong                    | 78.65          | 78.82       | 78.7    | 78.65            | 78.82         | 78.7      | 35.92          | 28.49       | 29.97   |
| Base Weak                      | 76.11          | 76.11       | 76.11   | 76.11            | 76.11         | 76.11     | 24.56          | 19.66       | 19.95   |
| Self-Consistency - Direct      | 71.48          | 71.48       | 71.48   | 71.48            | 71.48         | 71.48     | 23.65          | 17.86       | 18.77   |
| Self-Consistency - Clustering  | 72.24          | 72.24       | 72.24   | 72.24            | 72.24         | 72.24     | 23.61          | 18.02       | 18.9    |
| Self-Consistency - Universal   | 71.48          | 71.48       | 71.48   | 71.48            | 71.48         | 71.48     | 23.65          | 17.86       | 18.77   |
| Best of N                      | 71.86          | 71.86       | 71.86   | 71.86            | 71.86         | 71.86     | 24.08          | 18.62       | 19.36   |
| Strong LM Single Attempt       | 78.14          | 78.35       | 78.21   | 78.14            | 78.35         | 78.21     | 38.13          | 32.03       | 33.06   |
| Prompt Reduction               | 70.88          | 71.13       | 70.96   | 70.88            | 71.13         | 70.96     | 31.91          | 23.21       | 24.78   |
| Weak LM First                  | 77.73          | 77.73       | 77.73   | 77.73            | 77.73         | 77.73     | 33.98          | 27.07       | 28.62   |
| Plan                           | 84.37          | 84.83       | 84.48   | 84.37            | 84.83         | 84.48     | 37.48          | 29.69       | 31.28   |
| Instance Level QA Pairs        | 77.19          | 77.5        | 77.29   | 77.19            | 77.5          | 77.29     | 32.2           | 26.17       | 26.87   |
| Weak Router                    | 75.4           | 75.4        | 75.4    | 75.4             | 75.4          | 75.4      | 23.26          | 18.98       | 19.08   |
| Strong Router                  | 74.04          | 74.65       | 74.19   | 74.04            | 74.65         | 74.19     | 24.98          | 19.89       | 20.59   |
| 1 Shot Succesfull - Random     | 75.79          | 75.79       | 75.79   | 75.79            | 75.79         | 75.79     | 18.15          | 12.72       | 14.01   |
| 5 Shot Succesfull - Random     | 72.56          | 72.56       | 72.56   | 72.56            | 72.56         | 72.56     | 23.67          | 17.08       | 18.6    |
| 1 Shot Succesfull - Similarity | 71.75          | 71.75       | 71.75   | 71.75            | 71.75         | 71.75     | 20.17          | 17.15       | 17.06   |
| 5 Shot Succesfull - Similarity | 72.89          | 72.89       | 72.89   | 72.89            | 72.89         | 72.89     | 21.59          | 15.72       | 17.12   |
| Repo Structure                 | 68.97          | 68.97       | 68.97   | 68.97            | 68.97         | 68.97     | 29.55          | 22.8        | 24.31   |
| Repo Level QA Pairs            | 75.74          | 75.74       | 75.74   | 75.74            | 75.74         | 75.74     | 22.34          | 16.72       | 17.79   |
| Repo Summary                   | 76.61          | 76.61       | 76.61   | 76.61            | 76.61         | 76.61     | 24.02          | 18.62       | 19.21   |

Table 14: Localization performance for GPT-4o-mini as strong LM and Qwen2.5-Coder-7B as weak LM. Cells highlighted in red depict lower score than the base weak LM setting.