

A Related Work

Code Generation with Large Language Models. Recent advances in large language models (LLMs) have significantly boosted code generation by leveraging pretraining on large-scale code corpora (Lu et al., 2021; Christopoulou et al., 2022; Guo et al., 2024; Hui et al., 2024). At inference time, two main strategies have emerged to further enhance performance. Construction-based methods generate solutions step-by-step, often guided by value models (Yao et al., 2023; Wang et al., 2024a), process-based reward models (Lightman et al., 2023), or planning techniques like Monte Carlo Tree Search (Hao et al., 2023; Zhou et al., 2023). In contrast, improvement-based approaches iteratively refine full code drafts using multi-turn updates, agentic workflows (Wang et al., 2024b; Zhong et al., 2024; Zhang et al., 2024a), and test-time feedback, sometimes with multi-agent collaboration (Li et al., 2024; Zan et al., 2024). While effective, many of these methods are resource-intensive and complex. Our work introduces a lightweight local search framework that streamlines key components into an efficient and scalable loop.

Reward Models. Reward models play a central role in RLHF (Ouyang et al., 2022), providing learning signals for policy optimization. To reduce dependence on human-labeled data, RLAIIF (Lee et al., 2023) proposed an automated reward data pipeline. More recently, reward models have been extended to reasoning tasks. Math-Shepherd (Wang et al., 2023) and others (Zhang et al., 2025) trained process-based reward models to guide inference-time strategies, while outcome-based models have supported tree search (Jiang et al., 2024). Generative Reward Models (GRMs) (Mahan et al., 2024; McAleese et al., 2024; Zhang et al., 2024b; Chen et al., 2025) further leverage CoT-based self-critique for scoring outputs. Distinct from these paradigms, we propose a reward model trained to estimate *revision distance*, capturing the minimal steps needed to reach a correct solution. This enables more efficient candidate evaluation and improves the effectiveness of iterative code refinement.

B Experimental Setup.

B.1 Revision Reward Model

This section outlines the hyperparameters and settings used during the training phase of the revision reward model. We trained the revision reward model on Qwen2.5-7B-Instruct using the TRL library (von Werra et al., 2020) with DeepSpeed ZeRO Stage 2 on 4 NVIDIA H100 GPUs. The training was conducted for one epoch on a combined dataset of LiveCodeBench and TACO. Table 1 details the training configuration.

Table 1: Revision reward model training Configuration

Parameter	Value
Mixed Precision	bf16
Batch Size per Device	8
Number of Epochs	1
Gradient Checkpointing	True
Learning Rate	5.0e-6
Logging Steps	25
Evaluation Strategy	Steps
Evaluation Interval	Every 500 steps
Save Interval	Every 3000 steps
Max Sequence Length	2048
Push to Hub	False
Optimizer	paged_adamw_32bit
Warmup Ratio	0.05
Learning Rate Scheduler	Cosine
Number of GPUs	4 × NVIDIA H100

32 B.2 Local Search Hyperparameters

33 We use Qwen2.5-32B-Instruct as the inference model throughout all experiments, with a decoding
 34 temperature of 0.2 and a top- p value of 0.95. All algorithms are run under a fixed token budget of
 35 7,000 tokens per task.

36 For **Hill Climbing (HC)**, we initialize with 5 draft codes and expand 3 neighbors for each candidate
 37 during each improvement iteration.

38 For the **Genetic Algorithm (GA)**, we similarly maintain a population of 5 draft codes. In each
 39 iteration, we select 2 codes from the candidate pool as parent codes, with each code allowed to be
 40 selected as a parent up to 3 times.

41 C Additional Evaluation on GPT-4o

42 To further validate the effectiveness of RELOC, we conduct experiments on the closed-source model
 43 gpt-4o-2024-1120. Specifically, we evaluate RELOC and several baselines, including the state-of-
 44 the-art *Plan Search* algorithm and *Best of N* sampling, on the LIVECODEBENCH benchmark. For
 45 RELOC, we employ the revision reward model trained as described in Section B.1 to guide the search
 46 process.

47 In our evaluation, the revision reward model guiding RELOC’s local search was trained entirely on
 48 data sampled from the open-source Qwen2.5-32B-Instruct model, a setting that differs in both
 49 distribution and model family from the target inference model, GPT-4o. Remarkably, as shown in
 50 Table 2, RELOC achieves the highest Pass@1 score (44.2%) while consuming only 8.7K tokens on
 51 average—representing a 52% reduction in token usage compared to *Plan Search*. This underscores
 52 the efficiency of RELOC’s local search mechanism.

53 More importantly, these results demonstrate that the revision reward model, trained on Qwen gen-
 54 erated code trajectories, generalizes robustly to guide search on GPT-4o. This transferability is
 55 non-trivial: GPT-4o may exhibit different stylistic tendencies, error patterns, and semantic representa-
 56 tions compared to Qwen2.5-32B. Yet, the reward model still provides reliable signals for ranking
 57 candidate revisions, suggesting that it captures model-agnostic features of code quality, such as
 58 syntactic closeness to correct solutions, functional coherence, and local editability.

59 Such robustness to distributional shifts suggests broader applicability of our approach. It indicates
 60 that RELOC, and particularly its reward model component, can serve as a plug-and-play module to
 61 improve inference-time performance across diverse LLMs, without the need for costly re-annotation
 62 or model-specific retraining. This is particularly valuable for deployment in scenarios involving
 63 closed-source or frequently updated models, where direct supervision signals or fine-tuning access
 are unavailable.

Table 2: Performance comparison on GPT-4o (gpt-4o-2024-1120).

Methods	Rew.	Pass@1 (%)	Tokens (1K)
Plan Search	self-evaluation	42.7	16749
BoN	Pass Rate	41.8	12278
ReLoc_HC (Ours)	Revision Reward Model	44.2	8733

64

Table 3: Write diverse observations based on the problem to guide the generation of draft code.

Attribute	Description
DRAFTCODE	You are an expert Python programmer. You will be given a competitive programming question (problem specification). You will return 5 useful, non-obvious, and correct observations about the problem, like hints to solve the problem. You will NOT return any code. Be as creative as possible, going beyond what you think is intuitively correct. For each observation should be between tag [OBSERVATION] and [/OBSERVATION].
Outputs	<p>[OBSERVATION] Observation 1: A permutation can be fixed with at most one swap if and only if it has at most 2 inversions. [/OBSERVATION]</p> <p>[OBSERVATION] Observation 2: Only 3-cycles (like "bca" and "cab") cannot be solved with a single swap operation. [/OBSERVATION]</p> <p>...</p>

Table 4: ReLoc_HC generates diverse modification strategies for incumbent code

Attribute	Description
HC_NEIGHBOR	You are an expert in debugging Python code. You will be provided with a code snippet that requires debugging, along with a revision history for reference. Your task is to begin by writing a brief textual explanation of the current code—summarize its intended behavior and any evident issues between [explanation] and [/explanation]. Next, propose three refinement directions that could help improve the code. Please put each direction between tag [direction] and [/direction]. Important: Do not include any Python code in your response—only the explanation and the three refinement directions.
Outputs	<p>[explanation] The feedback indicates that the main issue is the function returns only the first element of min-k and max-k, instead of the entire lists. [/explanation]</p> <p>[direction] Modify the return statement to return the full min-k and max-k lists. [/direction]</p> <p>[direction] Include input validation to ensure K is a non-negative integer. [/direction]</p> <p>...</p>

Table 5: ReLoc_HC revises code according to different strategies

Attribute	Description
HC_NEIGHBOR	You are a helpful programming assistant and an expert in Python. The user has written code that contains errors. You will be provided with a Python programming problem, the user’s code, revision history, an explanation, and directions for refinement. Your task is to debug and revise the code to correctly solve the problem. Generate a corrected version of the complete program, incorporating the explanation, refinement directions, and revision history. Output only one corrected program enclosed within a single pair of code delimiters. Do not include any additional commentary or text.
Outputs	<pre> def minimumBoxes(apple_packs, box_capacities): for box_capacity in box_capacities: current_capacity += box_capacity boxes_used += 1 if current_capacity >= total_apples: return boxes_used return boxes_used apple_packs = [1, 3, 2] box_capacities = [4, 3, 1, 5, 2] print(minimumBoxes(apple_packs, box_capacities)) apple_packs = [5, 5, 5] box_capacities = [2, 4, 2, 7] print(minimumBoxes(apple_packs, box_capacities)) </pre>

Table 6: ReLoc_GA directly prompts LLM to generate neighborhood by referencing parent code

Attribute	Description
GA_NEIGHBOR	You are a helpful programming assistant and an expert in Python. The user has written code that contains errors. You will be provided with a Python programming problem and the user’s code intended to solve it. Your task is to refer to the input code and revise it to correctly solve the problem. Generate a corrected version of the complete program. Output only one corrected program enclosed within a single pair of code delimiters. Do not include any additional commentary or text.
Outputs	<pre> def minimumBoxes(apple_packs, box_capacities): for box_capacity in box_capacities: current_capacity += box_capacity boxes_used += 1 if current_capacity >= total_apples: return boxes_used return boxes_used apple_packs = [1, 3, 2] box_capacities = [4, 3, 1, 5, 2] print(minimumBoxes(apple_packs, box_capacities)) apple_packs = [5, 5, 5] box_capacities = [2, 4, 2, 7] print(minimumBoxes(apple_packs, box_capacities)) </pre>

66 E Step by Step Revisions

67 Below, we demonstrate how ReLoc guides Qwen2.5-32B-Instruct step by step to revise completely incorrect draft code into correct code.

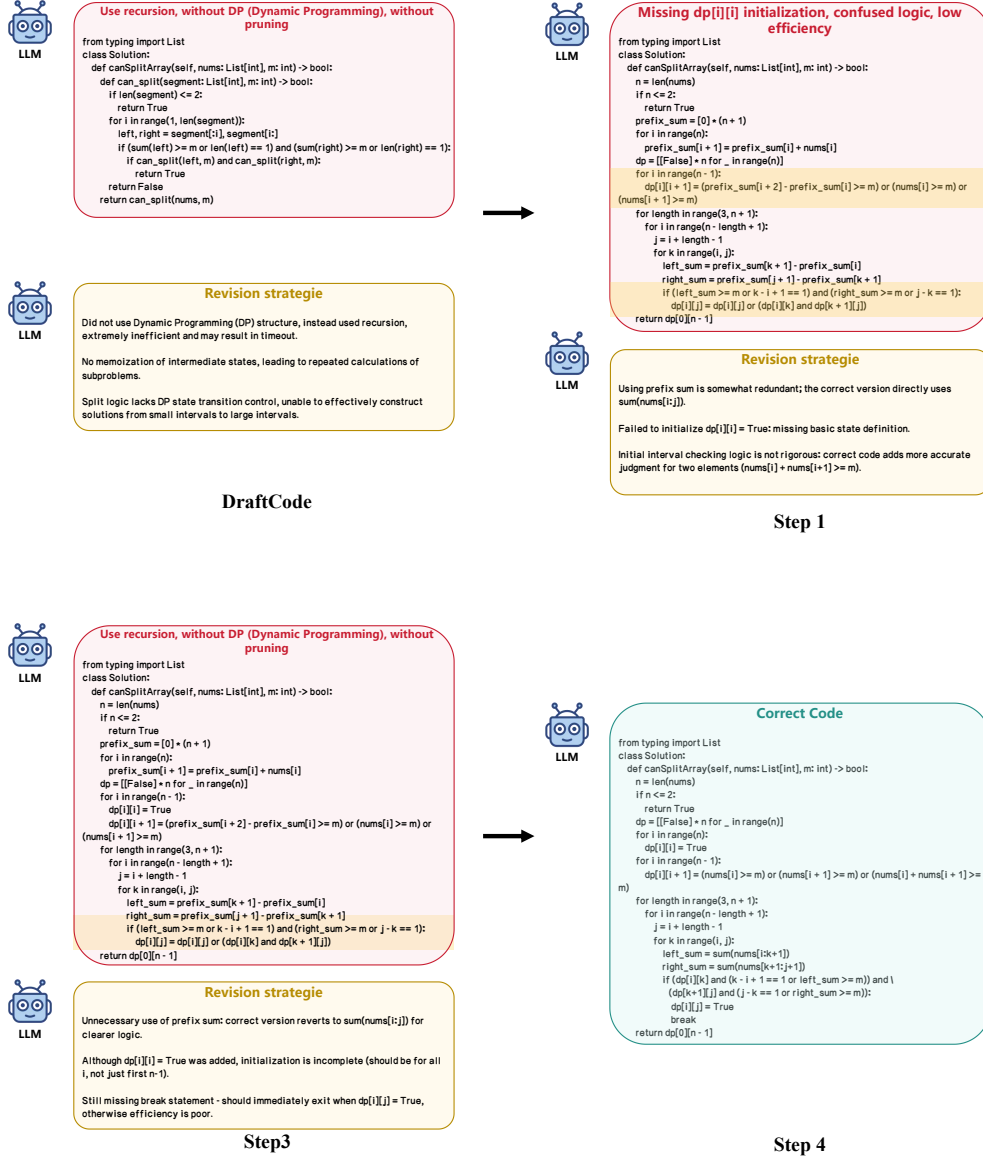


Figure 1: ReLoc step-by-step revise incorrect code

68

69 References

- 70 Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin
 71 Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark
 72 dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*, 2021.
- 73 Fenia Christopoulou, Gerasimos Lampouras, Milan Gritta, Guchun Zhang, Yinpeng Guo, Zhongqi Li,
 74 Qi Zhang, Meng Xiao, Bo Shen, Lin Li, et al. Pangu-coder: Program synthesis with function-level
 75 language modeling. *arXiv preprint arXiv:2207.11280*, 2022.

76 Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi,
77 Yu Wu, YK Li, et al. DeepSeek-Coder: When the large language model meets programming—the
78 rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

79 Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang,
80 Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*,
81 2024.

82 Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan.
83 Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural*
84 *Information Processing Systems*, 36:11809–11822, 2023.

85 Chaojie Wang, Yanchen Deng, Zhiyi Lyu, Liang Zeng, Jujie He, Shuicheng Yan, and Bo An.
86 Q*: Improving multi-step reasoning for LLMs with deliberative planning. *arXiv preprint*
87 *arXiv:2406.14283*, 2024a.

88 Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan
89 Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *ICLR*, 2023.

90 Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu.
91 Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*,
92 2023.

93 Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language
94 agent tree search unifies reasoning acting and planning in language models. *arXiv preprint*
95 *arXiv:2310.04406*, 2023.

96 Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan,
97 Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for AI software
98 developers as generalist agents. In *ICLR*, 2024b.

99 Li Zhong, Zilong Wang, and Jingbo Shang. Debug like a human: A large language model debugger
100 via verifying runtime execution step-by-step. *arXiv preprint arXiv:2402.16906*, 2024.

101 Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation
102 with tool-integrated agent systems for real-world repo-level coding challenges. *arXiv preprint*
103 *arXiv:2401.07339*, 2024a.

104 Jierui Li, Hung Le, Yingbo Zhou, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Code-
105 tree: Agent-guided tree search for code generation with large language models. *arXiv preprint*
106 *arXiv:2411.04329*, 2024.

107 Daoguang Zan, Zhirong Huang, Ailun Yu, Shaoxin Lin, Yifan Shi, Wei Liu, Dong Chen, Zongshuai
108 Qi, Hao Yu, Lei Yu, et al. Swe-bench-java: A github issue resolving benchmark for java. *arXiv*
109 *preprint arXiv:2408.14354*, 2024.

110 Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong
111 Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow
112 instructions with human feedback. In *NeurIPS*, pages 27730–27744, 2022.

113 Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton
114 Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, et al. Rlaif vs. rlhf: Scaling reinforcement
115 learning from human feedback with ai feedback. *arXiv preprint arXiv:2309.00267*, 2023.

116 Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang
117 Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. *arXiv*
118 *preprint arXiv:2312.08935*, 2023.

119 Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu,
120 Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical
121 reasoning. *arXiv preprint arXiv:2501.07301*, 2025.

122 Jinhao Jiang, Zhipeng Chen, Yingqian Min, Jie Chen, Xiaoxue Cheng, Jiapeng Wang, Yiru Tang,
123 Haoxiang Sun, Jia Deng, Wayne Xin Zhao, et al. Technical report: Enhancing llm reasoning with
124 reward-guided tree search. *arXiv preprint arXiv:2411.11694*, 2024.

- 125 Dakota Mahan, Duy Van Phung, Rafael Rafailov, Chase Blagden, Nathan Lile, Louis Castricato,
126 Jan-Philipp Fränken, Chelsea Finn, and Alon Albalak. Generative reward models. *arXiv preprint*
127 *arXiv:2410.12832*, 2024.
- 128 Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz,
129 and Jan Leike. Llm critics help catch llm bugs. *arXiv preprint arXiv:2407.00215*, 2024.
- 130 Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal.
131 Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*,
132 2024b.
- 133 Xiusi Chen, Gaotang Li, Ziqi Wang, Bowen Jin, Cheng Qian, Yu Wang, Hongru Wang, Yu Zhang,
134 Denghui Zhang, Tong Zhang, et al. Rm-r1: Reward modeling as reasoning. *arXiv preprint*
135 *arXiv:2505.02387*, 2025.
- 136 Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan
137 Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformer reinforcement
138 learning. <https://github.com/huggingface/trl>, 2020.