UNDERSTANDING SCATTERED FOREST SEARCH: A VERSION-SPACE PERSPECTIVE ON MULTI-TURN PROGRAM CORRECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

In multi-turn program correction, the state-of-the-art method SCATTERED FOREST SEARCH (SFS) has been proposed, employing Monte Carlo Tree Search (MCTS) with carefully crafted initial seeds and text-based optimization. However, since SFS integrates multiple components, the effects of each component on performance and the overall behavior of SFS have not been sufficiently analyzed. In this work, we theoretically analyze the refinement process of SFS from the perspective of *version spaces* in learning theory and clarify its behavior. First, as a basis for the theoretical analysis, we introduce a sequential self-refinement method (Line), which starts from an initial program and repeatedly refines the resulting program. Furthermore, while Line progresses the refinement process in the depth direction, we introduce ITERATIVE REFINEMENT OF REPAIR INSTRUCTIONS (IRRI) to capture the refinement process in the width direction, which fixes initial programs and iteratively refines repair instructions. We then analyze Line and IRRI and conduct a theoretical analysis of SFS by positioning it as an intermediate method between the two. Our theoretical analysis reveals that SFS exhibits behavior closer to IRRI than to Line, and this theoretical characteristic is also confirmed experimentally. Considering computational resources and methodological simplicity, these results suggest that a simpler method, IRRI, may achieve a similar refinement process without relying on the complex correction mechanism of SFS.

1 INTRODUCTION

A representative task in the study of inference scaling is multi-turn program correction, in which an initial program is refined through multiple rounds of feedback. Several representative approaches have been proposed, including best-of-N (BoN) sampling (Cobbe et al., 2021; Lightman et al., 2023), self-correction methods (Madaan et al., 2023; Shinn et al., 2023), and tree-search-based methods that integrate multiple reasoning trajectories (Tian et al., 2024; Zhou et al., 2024). These methods have been shown to improve performance (Chen et al., 2022; Wang et al., 2024). One reason for this effectiveness is that providing enough trials increases the probability that an LLM will sample a correct answer (Brown et al., 2024; Snell et al., 2024). However, Light et al. (2025) have shown that existing methods often generate highly similar outputs, resulting in insufficient exploration of the solution space.

Light et al. (2025) have proposed SCATTERED FOREST SEARCH (SFS) that builds on Monte Carlo Tree Search (MCTS) and incorporates techniques such as careful seed initialization and text-based optimization. In SFS, initial programs serve as the root nodes, and the program space is efficiently explored through MCTS by iteratively applying repair instructions generated by an LLM. Moreover, feedback is provided for the repair instructions used during the refinement process, and the resulting insights are accumulated as shared information. They have empirically reported that SFS achieves state-of-the-art performance in multi-turn program correction (Light et al., 2025). However, it remains unclear why SFS leads to performance improvements and how its refinement process proceeds. Understanding these factors is crucial for developing simpler and more efficient approaches that could retain the benefits of inference scaling without relying on costly or complex search mechanisms.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To address this gap, we develop a theoretical framework for analyzing refinement processes from the perspective of *version spaces* in learning theory (Mitchell, 1977). As a basis for this framework, we first introduce a sequential self-refinement method, Line, which starts from an initial program and repeatedly refines the resulting program. This method captures the characteristics of refinement in the depth direction, where each refinement is applied to the program produced by the preceding step. However, Line does not capture refinement in the width direction, where initial programs can be refined using progressively improved repair instructions. To capture this aspect, we introduce ITERATIVE REFINEMENT OF REPAIR INSTRUCTIONS (IRRI), which fixes initial programs and iteratively refines repair instructions. We then theoretically analyze the refinement processes of Line and IRRI from the perspective of version spaces in learning theory. Based on these analyses, we characterize SFS as an intermediate method between Line and IRRI and analyze its refinement process within the same theoretical framework. Our theoretical analysis shows that SFS exhibits refinement behavior closer to IRRI than to Line. This theoretical characterization is also confirmed experimentally. These results suggest that IRRI can capture a refinement process similar to that of SFS despite its simpler design.

2 RELATED WORK

Recent studies have increasingly focused on improving model performance by allocating more computation at inference time (Brown et al., 2024; Snell et al., 2024). One of the simplest and most common methods is best-of-N (BoN) sampling (Cobbe et al., 2021; Lightman et al., 2023). In this approach, multiple samples are generated from the same prompt to ensure diversity among responses and improve output quality. Prior work has also proposed self-correction methods in which an initial response is generated and then iteratively verified and revised (Madaan et al., 2023; Shinn et al., 2023). These methods improve the model's responses by incorporating internal or external feedback as additional information. Yang et al. (2025) has proposed a probabilistic theory of self-correction to explain the performance improvements observed in these methods.

Furthermore, incorporating tree search methods such as DFS, BFS, and MCTS has been shown to be effective for inference scaling. Prior work has applied tree search to multi-step tasks, including selecting the next reasoning step in reasoning processes (Yao et al., 2023; Besta et al., 2024; Tian et al., 2024; Zhou et al., 2024) and decoding the next token during generation (Feng et al., 2023; Zhang et al., 2023). Tree search has also been applied to optimization problems, where it functions as a black-box method for exploring a solution space (Light et al., 2025).

Overall, we build upon previous research (Light et al., 2025) and aim to identify the factors that truly contribute to the behavior and performance of increasingly complex inference scaling methods. Motivated by the lack of a theoretical understanding of such methods, we analyze their iterative refinement processes from the perspective of version spaces in learning theory (Mitchell, 1977) and develop a theoretical framework for characterizing their behavior.

3 PROBLEM SETTING AND REFINEMENT METHODS

3.1 PROBLEM SETTING

Following Light et al. (2025), we formulate program generation as a black-box optimization problem (Golovin et al., 2017). Each task is represented as $x = \langle p, D_{\text{gt}} \rangle \in X$, where p denotes a prompt, and D_{gt} represents a ground-truth test-case set. The objective is to provide an LLM with a prompt p and generate a program $c \sim \text{LLM}(p)$ that passes all test cases in D_{gt}. Although the model cannot access D_{gt}, it can generate a validation test-case set D_{val}. Using D_{val} and the program execution environment, the model obtains feedback on its generated program c. This feedback is then used to determine whether the program should be refined. In the evaluation phase, generated programs are evaluated against D_{gt} and are deemed correct only when all test cases are passed. For each task, an LLM is allowed to generate programs up to k times, and a task is considered solved if at least one of them is correct. Following these rules, the evaluation metric Pass@k Rate (Chen et al., 2021) is calculated as the proportion of tasks $x \in X$ that are solved with at most k generated programs. An overview of this problem setting is illustrated in Appendix A.

3.2 SCATTERED FOREST SEARCH (SFS)

Several inference-time methods have been explored in prior work on multi-turn program correction. However, Light et al. (2025) have shown that these methods often generate highly similar candidate programs, resulting in insufficient exploration of the solution space. To address this challenge, they have proposed SCATTERED FOREST SEARCH (SFS) (Light et al., 2025). SFS is a MCTS-based method that combines careful seed initialization with text-based optimization and consists of the following three components.

- SCATTERING: Dynamically varies input prompts with different repair instructions when sampling revised programs.
- FORESTING: Generates multiple initial programs that act as starting points for MCTS.
- SCOUTING: Provides feedback on repair instructions employed during the refinement process and distills general insights that can be shared across search branches.

Their study has confirmed that SFS outperforms existing methods in terms of inference scaling performance (Light et al., 2025). Details of SFS are provided in Appendix B.

3.3 SEQUENTIAL SELF-REFINEMENT METHOD (LINE)

As a basis for our theoretical analysis, we introduce a sequential self-refinement method, Line, which builds upon the frameworks of Madaan et al. (2023) and Shinn et al. (2023). This method captures the characteristics of the refinement process in the depth direction.

Initial Generation: Given an input prompt p, Line generates an initial program c_1.

$$c_1 \sim \mathrm{LLM}(p). \tag{1}$$

Refinement Process: First, based on the refinement target c_t, Line generates a repair instruction r_t. Next, Line refines c_t using r_t.

$$r_t \sim \mathrm{LLM}(p, c_t), \quad c_{t+1} \sim \mathrm{LLM}(p, c_t, r_t). \tag{2}$$

3.4 ITERATIVE REFINEMENT OF REPAIR INSTRUCTIONS (IRRI)

While Line captures the refinement process in the depth direction, it does not capture refinement in the width direction, where initial programs can be revised using increasingly refined repair instructions. To capture this aspect, we introduce ITERATIVE REFINEMENT OF REPAIR INSTRUCTIONS (IRRI), which refines initial programs through iterative feedback on repair instructions. IRRI accumulates feedback from previous repair instructions as shared information and uses it to generate subsequent ones, thereby progressively improving the repair instructions used for correction. An overview of IRRI is illustrated in Appendix C.

Initial Generation: Given an input prompt p, IRRI generates one or multiple initial programs $c_1, \ldots, c_n$.

$$c_1 \sim \mathrm{LLM}(p), \ldots, \; c_n \sim \mathrm{LLM}(p). \tag{3}$$

Refinement Process: First, based on the selected initial program c_t and the shared information I_t, IRRI generates one or more repair instructions $r_1, \ldots, r_m$. Next, IRRI refines c_t using one of them r_t.

$$r_1, \ldots, r_m \sim \mathrm{LLM}(p, c_t, I_t), \quad c_{\text{ref}} \sim \mathrm{LLM}(p, c_t, r_t). \tag{4}$$

Feedback for Repair Instructions: After the refinement process, IRRI evaluates the revised program c_{ref} on D_{val} and updates the shared information I_t based on c_t, r_t, and c_{ref}.

$$I_{t+1} \sim \mathrm{LLM}(p, c_t, I_t, r_t, c_{\text{ref}}). \tag{5}$$

4 THEORETICAL ANALYSIS USING VERSION SPACES

In this section, we present a theoretical analysis of Line, IRRI, and SFS. We begin by analyzing two contrasting methods, Line and IRRI, and then analyze SFS, which occupies an intermediate position between the two.

In multi-turn program correction, the prompt for program refinement at each turn can be represented as a pair consisting of the program to be revised and the repair instruction applied to it. These two elements are the only components that vary across turns, and thus fully characterize the prompt used to revise the program. Throughout the refinement process, the goal of each method can be regarded as identifying the true pair among these. From this perspective, each method can be viewed as a process in which the feedback obtained at each refinement step is used as a clue to eliminate pairs that are likely incorrect, thereby narrowing down the candidates to the correct pair.

To this end, we adopt the notion of a version space from learning theory (Mitchell, 1977), treating the product space of programs and repair instructions as the hypothesis space, and analyze how this hypothesis space shrinks and evolves throughout the refinement process. This framework allows us to compare how each method utilizes feedback to update the hypothesis space and how differences in this updating process relate to differences in performance among the methods.

4.1 DEFINITION AND ASSUMPTION

Let C denote the space of programs, R denote the space of repair instructions, $H = C \times R$ denote the hypothesis space, and E denote the space of observations. Following Section 3.1, each element $e \in E$ represents a failed test case on D_{val}. For $c \in C$ and $r \in R$, we write $\text{Pass}(c, r)$ to indicate whether applying r to c yields a program that satisfies the specification and $\text{Cons}(c, r, e)$ to indicate whether applying r to c is consistent with an observation $e \in E$. Using these, we define the true hypothesis space as $H^\star = \{h = (c, r) \in H \mid \text{Pass}(c, r) = 1\}$. For a fixed program $c \in C$, we write H_c for the hypothesis space associated with c and $H_c^\star$ for the corresponding true hypothesis space. We denote by E_c the space of observations that can be obtained from c, and we define $H_c(e) = \{h = (c, r) \in H_c \mid \text{Cons}(c, r, e) = 1\}$ as the subset of H_c that is consistent with $e \in E_c$. We then denote the history up to time t by $O_t = ((h_1, e_1), \cdots, (h_t, e_t))$, where, at each time step $s\,(1 \le s \le t)$, the hypothesis $h_s = (c_s, r_s)$ belongs to H_{c_s} and the observation e_s belongs to E_{c_s}. Finally, we write $\mathcal{T}_t$ for the true hypothesis space at time t and $\mathcal{V}_t$ for the hypothesis space that survives (i.e., remains consistent with the history O_t) at time t.

Throughout this analysis, we make the following assumption, which states that any true hypothesis is consistent with every observation.

Assumption 1. *(Soundness)*

$$h^\star = (c^\star, r^\star) \in H^\star \Longrightarrow \forall e \in E_{c^\star}.h^\star \in H_{c^\star}(e). \tag{6}$$

4.2 SEQUENTIAL SELF-REFINEMENT METHOD (LINE)

We first analyze Line, where the refinement target changes throughout the correction process. Let $c_1 \in C$ denote the initial program, and let $c_{t+1} \in C$ denote the program obtained by the t-th correction. Since the refinement target keeps changing, the refinement target c_t may be either a previously unknown program or an already known program. We therefore define the probability that the refinement target c_t is a known program as q_t^{old}. Since the refinement target at time 1 is the initial program, $q_1^{\text{old}} = 1$ holds. In the early stages of the refinement process, only a small number of programs have been generated, so q_t^{old} remains relatively low. However, as the process proceeds, the number of generated programs increases, and q_t^{old} increases accordingly. Based on this observation, we model q_t^{old} as

$$q_t^{\text{old}} = 1 - \exp(-\tau t) \quad (t \ge 2), \tag{7}$$

where τ is a small positive constant that depends on factors such as the model and the dataset.

In Line, the hypothesis space surviving at time t can be expressed as

$$\mathcal{V}_0 = H_{c_1}, \quad \mathcal{V}_t = \begin{cases} \mathcal{V}_{t-1} \cup H_{c_t}(e_t) & (c_t = \text{unknown}) \\ (\mathcal{V}_{t-1} \cap H_{c_t}(e_t)) \cup (\mathcal{V}_{t-1} \setminus H_{c_t}) & (c_t = \text{known}) \end{cases} (t \ge 1). \tag{8}$$

Correspondingly, the true hypothesis space at time t can be expressed as

$$\mathcal{T}_0 = H^\star_{c_1}, \quad \mathcal{T}_t = \begin{cases} \mathcal{T}_{t-1} \cup H^\star_{c_t} & (c_t = \text{unknown}) \\ \mathcal{T}_{t-1} & (c_t = \text{known}) \end{cases} \ (t \geq 1). \tag{9}$$

In addition to Assumption 1, we make the following assumption for analyzing Line.

Assumption 2. *(Probabilistic realizability) For the refinement target $c_t \in C$, define the random variable $Z_{c_t} = \mathbb{1}[H^\star_{c_t} \neq \varnothing] \in \{0, 1\}$. We assume the following properties of the LLM's sampling distribution.*

1. *(Initial distribution) For the initial program $c_1 \in C$, there exists $q^{\textit{Line}} \in (0, 1)$ such that $\Pr[H^\star_{c_1} \neq \varnothing] = q^{\textit{Line}}$.*
2. *(Uniform lower bound) There exists $\pi^{\textit{Line}} \in (0, 1)$ such that $\Pr[Z_{c_t} = 1 \mid c_t = \textit{unknown}] \geq \pi^{\textit{Line}}$ for every time t.*
3. *(Independence) For different programs $c_t \neq c'_t$, Z_{c_t} and $Z_{c'_t}$ are mutually independent.*

This assumption captures the realizability of the LLM, requiring that the initial program admit a true hypothesis with probability q^{Line} and that an unknown program admit a true hypothesis with probability at least a uniform positive lower bound π^{Line}. Once the true hypothesis space $\mathcal{T}_t$ first becomes non-empty, the true hypothesis space is never pruned, so it remains non-empty at all subsequent times. We therefore define the first time at which $\mathcal{T}_t$ becomes non-empty as $t_c = \min\{t \geq 0 \mid \mathcal{T}_t \neq \varnothing\}$. Then, we obtain the following theorem, which characterizes the refinement process of Line.

Theorem 1. *Under Assumptions 1 and 2, for any history $O_t = ((h_1, e_1), (h_2, e_2), \cdots, (h_t, e_t))$ and time $t_c = \min\{t \geq 0 \mid \mathcal{T}_t \neq \varnothing\}$, the following properties hold.*

1. *($(1 - \delta_{\textit{Line}})$-monotonic non-increasing)*
   At each time t,
   - *With probability $1 - q^{old}_t$, the refinement target c_t is unknown, in which case $\mathcal{V}_t \supseteq \mathcal{V}_{t-1}$.*
   - *With probability q^{old}_t, the refinement target c_t is known, in which case $\mathcal{V}_t \subseteq \mathcal{V}_{t-1}$.*

   Moreover, with probability at most

   $$1 - \delta_{\textit{Line}} = \exp\left(-A_\tau \left(1 - e^{-\tau(t-1)}\right)\right) \quad \left(A_\tau = \frac{e^{-2\tau}}{1 - e^{-\tau}}\right), \tag{10}$$

   all refinement targets $c_1, \ldots, c_t$ are known, in which case $\mathcal{V}_t \subseteq \mathcal{V}_{t-1} \subseteq \cdots \subseteq \mathcal{V}_0$.
2. *(Preservation of the true hypothesis) $\mathcal{T}_t \subseteq \mathcal{V}_t \quad (t \geq t_c)$.*
3. *(Non-emptiness)*

   $$\Pr[\mathcal{V}_t \neq \varnothing] \geq 1 - (1 - q^{\textit{Line}}) \exp\left(-\pi^{\textit{Line}} A_\tau \left(1 - e^{-\tau(t-1)}\right)\right) \quad \left(A_\tau = \frac{e^{-2\tau}}{1 - e^{-\tau}}\right). \tag{11}$$

The proof of Theorem 1 is provided in Appendix D.1. This theorem shows that there is always a upper bound on the probability that all refinement targets are known and, consequently, the hypothesis space is monotonically non-increasing. This indicates that there is a limit to the extent to which Line can properly narrow down the hypothesis space. There is also a further limitation in that the lower bound on the probability of non-emptiness does not reach 1 even as $t \to \infty$. Further discussion is provided in Appendices E.1 and E.2.

4.3 Iterative Refinement of Repair Instructions (IRRI)

We next turn to the analysis of IRRI. Unlike Line, IRRI first generates a set of initial programs $C_{\text{init}} \subseteq C$ and iteratively refines repair instructions. Given C_{init}, the hypothesis space surviving at

time t can be written as

$$\mathcal{V}_0 = \bigcup_{c \in C_{\text{init}}} H_c, \quad \mathcal{V}_t = (\mathcal{V}_{t-1} \cap H_{c_t}(e_t)) \cup (\mathcal{V}_{t-1} \setminus H_{c_t}) \, (t \geq 1). \tag{12}$$

Correspondingly, the true hypothesis space does not depend on t and is given by

$$\mathcal{T}_t = \bigcup_{c \in C_{\text{init}}} H_c^\star \quad (t \geq 0). \tag{13}$$

In addition to Assumption 1, the analysis of IRRI relies on the following assumption.

Assumption 3. *(Probabilistic realizability) For each initial program $c \in C_{init}$, define the random variable $Y_c = \mathbb{1}[H_c^\star \neq \varnothing] \in \{0, 1\}$. We assume the following properties of the LLM's sampling distribution.*

1. *(Uniform lower bound) There exists $q^{IRRI} \in (0, 1)$ such that $\Pr[Y_c = 1] \geq q^{IRRI}$ for every $c \in C_{init}$.*
2. *(Independence) $\{Y_c\}_{c \in C_{init}}$ are conditionally independent given the prompt.*

This assumption captures the realizability of the LLM, requiring a uniform positive lower bound on the probability that each initial program admits a true hypothesis. Under these assumptions, we obtain the following theorem, which characterizes the refinement process of IRRI.

Theorem 2. *Under Assumptions 1 and 3, for any history $O_t = ((h_1, e_1), (h_2, e_2), \cdots, (h_t, e_t))$, the following properties hold.*

1. *(Monotonic non-increasing) $\mathcal{V}_t \subseteq \mathcal{V}_{t-1}$.*
2. *(Preservation of the true hypothesis) $\mathcal{T}_0 \subseteq \mathcal{V}_t$.*
3. *(Non-emptiness) $\Pr[\mathcal{V}_t \neq \varnothing] \geq 1 - (1 - q^{IRRI})^N$.*

The proof of Theorem 2 is provided in Appendix D.2. This theorem guarantees that IRRI can appropriately narrow down the hypothesis space during the refinement process. Here, letting $G(N) = 1 - (1 - q^{\text{IRRI}})^N$, since $q^{\text{IRRI}} \in (0, 1)$ implies $0 < 1 - q^{\text{IRRI}} < 1$, $G(N)$ is monotonically increasing in N and satisfies

$$\lim_{N \to \infty} G(N) = 1 - \lim_{N \to \infty} (1 - q^{\text{IRRI}})^N = 1. \tag{14}$$

This shows that increasing the number of initial programs generated has the benefit of increasing the probabilities that $\mathcal{T}_0 \neq \varnothing$ and $\mathcal{V}_t \neq \varnothing$.

4.4 Scattered Forest Search (SFS)

We then analyze SFS by positioning it as an intermediate method between Line and IRRI. SFS first generates a set of initial programs $C_{\text{init}} \subseteq C$, but MCTS may select the refinement target outside C_{init}. Let $c_t \in C$ denote the refinement target at time t, and let $\tilde{c}_t \in C$ denote the program obtained at time t. We also denote the tree structure after the t-th correction by $S_t = (U_t, F_t) \quad (U_t \subseteq C,\ F_t \subseteq C \times C)$. At time $t = 0$, $S_0 = (C_{\text{init}}, \varnothing)$. At time $t \geq 1$, since refining c_t yields $\tilde{c}_t$, we have $S_t = (U_{t-1} \cup \{\tilde{c}_t\}, F_{t-1} \cup \{(c_t, \tilde{c}_t)\})$. On the tree structure $S_t = (U_t, F_t)$, we recursively define the depth function $\text{dep} : \bigcup_t U_t \to \mathbb{Z}_{\geq 0}$ as follows. Since the set of initial programs C_{init} forms the roots of the tree structure, $\text{dep}(c) = 0 \quad (c \in C_{\text{init}})$. When $(c_t, \tilde{c}_t)$ is added at time t, $\text{dep}(\tilde{c}_t) = \text{dep}(c_t) + 1$. Due to Foresting, the tree structure S_t may have multiple roots. However, since $(c_t, \tilde{c}_t)$ represents the relation "refining c_t yields $\tilde{c}_t$", each node has at most one parent. Consequently, the function dep is uniquely determined for any $c \in C$.

We denote the maximum depth of the tree structure S_t by $K_t = \max_{c \in U_t} \text{dep}(c)$. We also denote the probability that the refinement target c_t is a node at depth k by $q_t^{\text{dep}}(k)$. Furthermore, we define the known probability conditioned on the depth as $q_t^{\text{old}}(k) = \Pr[c_t = \text{known} \mid \text{dep}(c_t) = k]$. Since nodes of depth 0 belongs to C_{init}, we have $\forall t. q_t^{\text{old}}(0) = 1$. In the early stages of the refinement

process, MCTS performs breadth-wise exploration rather than depth-wise exploration, so $q_t^{\text{dep}}(0)$ remains high. However, as the process proceeds, MCTS shifts its exploration from breadth-wise to depth-wise, and $q_t^{\text{dep}}(0)$ decreases accordingly. Based on this observation, we model $q_t^{\text{dep}}(0)$ as

$$q_t^{\text{dep}}(0) = \exp\left(-\gamma(t-1)\right) \quad (t \geq 1), \tag{15}$$

where γ is a small positive constant that depends on factors such as the model, the dataset, and the number of initial programs.

In SFS, the hypothesis space surviving at time t can be expressed as

$$\mathcal{V}_0 = \bigcup_{c \in C_{\text{init}}} H_c, \quad \mathcal{V}_t = \begin{cases} \mathcal{V}_{t-1} \cup H_{c_t}(e_t) & (c_t = \text{unknown}) \\ (\mathcal{V}_{t-1} \cap H_{c_t}(e_t)) \cup (\mathcal{V}_{t-1} \setminus H_{c_t}) & (c_t = \text{known}) \end{cases} (t \geq 1). \tag{16}$$

Correspondingly, the true hypothesis space at time t can be expressed as

$$\mathcal{T}_0 = \bigcup_{c \in C_{\text{init}}} H_c^\star, \quad \mathcal{T}_t = \begin{cases} \mathcal{T}_{t-1} \cup H_{c_t}^\star & (c_t = \text{unknown}) \\ \mathcal{T}_{t-1} & (c_t = \text{known}) \end{cases} (t \geq 1). \tag{17}$$

In addition to Assumption 1, we make the following assumption for analyzing SFS.

Assumption 4. *(Probabilistic realizability) For each initial program $c \in C_{init}$, define the random variable $Y_c = \mathbb{1}[H_c^\star \neq \varnothing] \in \{0,1\}$. For the refinement target $c_t \in C$, define the random variable $Z_{c_t} = \mathbb{1}[H_{c_t}^\star \neq \varnothing] \in \{0,1\}$. We assume the following properties of the LLM's sampling distribution and MCTS.*

1. *(Uniform lower bound) There exists $q^{SFS} \in (0,1)$ such that $\Pr[Y_c = 1] \geq q^{SFS}$ for every $c \in C_{init}$. There also exists $\pi^{SFS} \in (0,1)$ such that $\Pr[Z_{c_t} = 1 \mid c_t = unknown] \geq \pi^{SFS}$ for every time t.*
2. *(Independence) $\{Y_c\}_{c \in C_{init}}$ are conditionally independent given the prompt. Also, for different programs $c_t \neq c_t'$, Z_{c_t} and $Z_{c_t'}$ are mutually independent.*
3. *(Upper bound on the known probability at depth $k \geq 1$) There exists $\theta^{SFS} \in (0,1)$ such that $q_t^{old}(k) \leq 1 - \theta^{SFS}$ for any time t and any depth $k\,(\geq 1)$.*

This assumption captures the realizability of the LLM and MCTS, requiring a uniform positive bound q^{SFS} (resp. π^{SFS}) on the probability that an initial (resp. unknown) program admits a true hypothesis and a uniform upper bound $1 - \theta^{\text{SFS}}$ on the probability that a program at depth $k\,(\geq 1)$ is already known. Once the true hypothesis space $\mathcal{T}_t$ first becomes non-empty, the true hypothesis space is never pruned, so it remains non-empty at all subsequent times. We therefore define the first time at which $\mathcal{T}_t$ becomes non-empty as $t_c = \min\{t \geq 0 \mid \mathcal{T}_t \neq \varnothing\}$. Then, we obtain the following theorem, which characterizes the refinement process of SFS.

Theorem 3. *Under Assumptions 1 and 4, for any history $O_t = ((h_1, e_1), (h_2, e_2), \cdots, (h_t, e_t))$ and time $t_c = \min\{t \geq 0 \mid \mathcal{T}_t \neq \varnothing\}$, the following properties hold.*

1. *((1 - δ_{SFS})-monotonic non-increasing)*
   At each time t,
   - *With probability $\sum_{k=1}^{K_{t-1}} q_t^{dep}(k)\left(1 - q_t^{old}(k)\right)$, the refinement target c_t is unknown, in which case $\mathcal{V}_t \supseteq \mathcal{V}_{t-1}$.*
   - *With probability $\sum_{k=0}^{K_{t-1}} q_t^{dep}(k) q_t^{old}(k)$, the refinement target c_t is known, in which case $\mathcal{V}_t \subseteq \mathcal{V}_{t-1}$.*

   Moreover, with probability at least

$$1 - \delta_{SFS} = \exp\left(-\gamma \frac{t(t-1)}{2}\right), \tag{18}$$

   all refinement targets $c_1, \ldots, c_t$ are known,, in which case $\mathcal{V}_t \subseteq \mathcal{V}_{t-1} \subseteq \cdots \subseteq \mathcal{V}_0$.

2. *(Preservation of the true hypothesis)* $\mathcal{T}_t \subseteq \mathcal{V}_t \quad (t \geq t_c)$.

3. *(Non-emptiness)*

$$\Pr[\mathcal{V}_t \neq \varnothing] \geq 1 - (1 - q^{SFS})^N \exp\left(-\pi^{SFS}\theta^{SFS}\left(t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}}\right)\right). \tag{19}$$

The proof of Theorem 3 is provided in Appendix D.3. This theorem shows that there is always a lower bound on the probability that the hypothesis space is monotonically non-increasing. Since this lower bound becomes larger as t decreases, the result suggests that SFS can properly narrow down the hypothesis space when the number of refinement steps is relatively small. In such cases, MCTS prioritizes breadth-wise exploration, so the refinement targets are drawn from C_{init} with high probability. This means that SFS exhibits refinement behavior similar to that of IRRI. Moreover, the lower bound on the probability of non-emptiness converges to 1 as $N \to \infty$ and $t \to \infty$. This shows that increasing the number of initial programs generated has the benefit of increasing the probabilities that $\mathcal{T}_0 \neq \varnothing$ and $\mathcal{V}_t \neq \varnothing$. Further discussion is provided in Appendices E.3 and E.4.

5 Experiments

We empirically validate the theoretical characterization presented in Section 4.4, which shows that SFS exhibits refinement behavior closer to IRRI than to Line.

5.1 Empirical analysis of tree structures in SFS

We analyzed the search trees produced by SFS with gpt-4o-mini as the base model across four datasets: HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), APPS (Hendrycks et al., 2021) and CodeContests (Li et al., 2022). We configured SFS to generate 5 initial programs and perform up to 11 revision steps.

We measured the distribution of the maximum depth (Table 1) and the distribution of the depth at which the first correct programs appears for problems where a correct program is found (Table 2). These results suggest that, in relatively short refinement processes, SFS behaves like IRRI by prioritizing breadth-wise exploration over depth-wise exploration, thereby enabling it to reach the correct program.

Table 1: **Maximum depth in SFS.**

Dataset	Pass@1	Pass@16	Depth $= 0$	Depth $= 1$	Depth ≥ 2
HumanEval	88.13%	95.00%	58.12%	33.12%	8.76%
MBPP	77.58%	86.65%	55.92%	30.23%	13.85%
APPS	25.00%	39.50%	8.00%	61.00%	31.00%
CodeContests	7.88%	18.79%	1.82%	72.73%	25.45%

Table 2: **Depth at which SFS first reaches a correct program.**

Dataset	Pass@1	Pass@16	Depth $= 0$	Depth $= 1$	Depth ≥ 2
HumanEval	88.13%	95.00%	93.38%	6.62%	0.00%
MBPP	77.58%	86.65%	91.57%	7.27%	1.16%
APPS	25.00%	39.50%	75.32%	24.68%	0.00%
CodeContests	7.88%	18.79%	58.07%	41.93%	0.00%

5.2 Inference scaling

To examine whether IRRI can serve as a viable alternative to SFS, we compared the inference scaling performance of SFS and IRRI using gpt-4o-mini, gpt-4.1-mini, Llama-3.1-8B-Instruct (Grattafiori et al., 2024), and Llama-3.2-3B-Instruct (Grattafiori et al., 2024) as base models, on several program

Table 3: **Detailed settings of all evaluated methods.**

Method	Description
BoN	Generates 16 programs from the same prompt.
Line	Generates 1 initial program and performs up to 15 refinement steps (Section 3.3).
Tree	Building upon the framework of Zhou et al. (2024), generates 1 initial program followed by up to 15 refinement steps.
SFS	Generates 5 initial programs and performs up to 11 revision steps (Section 3.2 and Appendix B).
IRRI	Generates 5 initial programs with up to 11 revision steps (Section 3.4 and Appendix C).

generation benchmarks, including HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and APPS (Hendrycks et al., 2021). The detailed settings of all evaluated methods are provided in Table 3.

Figure 1 shows the results on APPS obtained using gpt-4o-mini and the results on MBPP obtained using Llama-3.2-3B-Instruct. The results for the other datasets and base models are reported in Appendix F. Each curve represents the mean performance over five runs, and each shaded region denotes 95% confidence intervals computed using the t-distribution. We found that IRRI achieves inference scaling performance comparable to or better than that of prior methods. Notably, although IRRI is simpler, it attains inference performance on par with SFS.

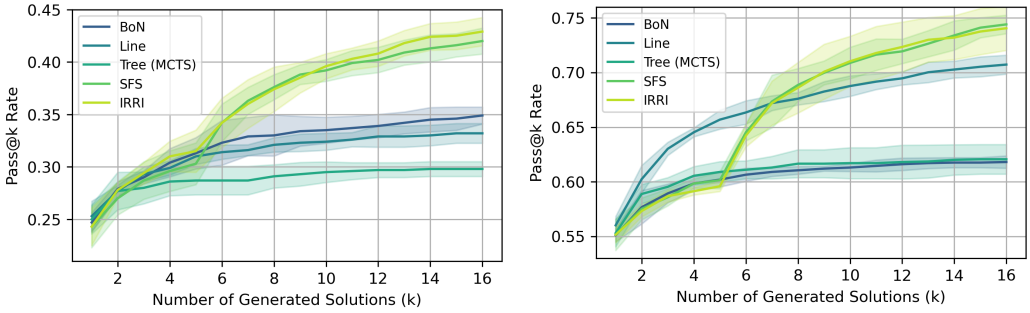


Figure 1: **Scaling curves for different methods on APPS using gpt-4o-mini (left) and on MBPP using Llama-3.2-3B-Instruct (right).**

6 Conclusion

We have introduced two contrasting refinement methods, Line and IRRI, which progress the refinement process in the depth direction and the width direction, respectively, and used them as a basis for theoretically analyzing SFS from the perspective of version spaces in learning theory. Our theoretical analysis reveals that SFS exhibits refinement behavior closer to IRRI than to Line, and this theoretical characteristic is also confirmed experimentally. Experimental results comparing computational costs demonstrate that a simpler method, IRRI, may achieve a similar refinement process without relying on SFS's complex correction mechanism. These findings provide new insights into the mechanisms underlying improved inference performance in multi-turn program correction and suggest promising directions for future research.

AI Use Statement

In this work, we used generative AI tools to help develop theoretical models or conceptual frameworks and to assist with translation. We have not used generative AI tools to formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, and interpret results. Generating synthetic datasets, cleaning and reformatting datasets, and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to create or edit software code, create artifacts, draft parts of a research paper,

summarize or analyze existing literature, brainstorm, search for information, edit a research paper to improve readability, identify relevant literature, and propose a title or keywords for a research paper. We have reviewed all AI-assisted work. AI-assisted brainstorming, including help developing theoretical models, was critically evaluated by the authors and cross-checked against relevant prior literature to confirm correctness. AI-assisted translations were reviewed and corrected by the authors. All AI-generated or AI-edited code was also tested and verified for correctness by the authors. AI-drafted text, summaries of literature, and suggested titles/keywords were rewritten and fact-checked by the authors, with all citations verified against original sources. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 17682–17690, 2024.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397*, 2022.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374. arXiv:2107.03374.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*, 2023.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1487–1495, 2017.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yifan Wu, YK Li, et al. Deepseek-coder: when the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Jonathan Light, Yue Wu, Yiyou Sun, Wenchao Yu, Yanchi Liu, Xujiang Zhao, Ziniu Hu, Haifeng Chen, and Wei Cheng. SFS: Smarter code space search improves LLM inference scaling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=MCHuGOkExF.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Tom M. Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 305–310, 1977.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Lei Han, Haitao Mi, and Dong Yu. Toward self-improvement of llms via imagination, searching, and criticizing. *Advances in Neural Information Processing Systems*, 37:52723–52748, 2024.

Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, Will Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. Planning in natural language improves llm search for code generation. *arXiv preprint arXiv:2409.03733*, 2024.

Zhe Yang, Yichang Zhang, Yudong Wang, Ziyao Xu, Junyang Lin, and Zhifang Sui. A probabilistic inference scaling theory for llm self-correction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13584–13598, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Shun Zhang, Zhenfang Chen, Yikang Shen, Mingyu Ding, Joshua B Tenenbaum, and Chuang Gan. Planning with large language models for code generation. *arXiv preprint arXiv:2303.05510*, 2023.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *ICML*, 2024.

A Overview of problem setting

Figure 2 illustrates an overview of the problem setting described in Section 3.1.

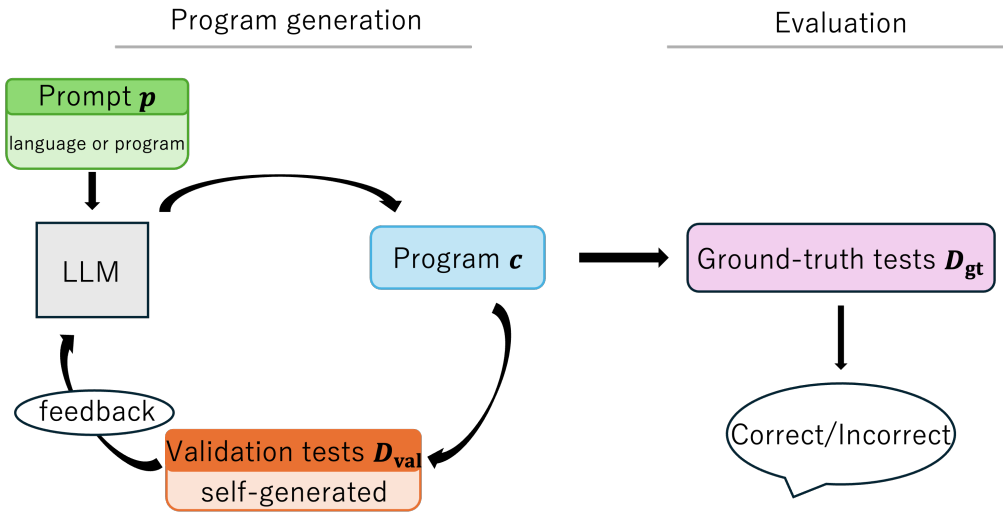


Figure 2: **Overview of problem setting.** Adapted from Light et al. (2025). Given a prompt p, an LLM iteratively generates and refines programs c on the basis of feedback from a validation test-case set D_{val}. The objective is to generate a program c_{true} that passes all ground-truth test cases D_{gt}.

B Details of Scattered Forest Search (SFS)

To enable diverse exploration of the solution space in program generation, Light et al. (2025) have proposed Scattered Forest Search (SFS). SFS builds on MCTS and incorporates techniques such as careful seed initialization and text-based optimization. This method encourages broader exploration of the solution space by combining LLM-driven repair instructions, the introduction of multiple initial programs, and the utilization of feedback and past search experience. Specifically, SFS consists of three main components: Scattering, Foresting, and Scouting, as illustrated in Figure 3.

Scattering: This component dynamically varies input prompts with different repair instructions when sampling revised programs. More concretely, given a program selected by MCTS along with its associated feedback, an LLM generates multiple repair instructions. Then, one of them is selected and incorporated into the prompt for generating the revised program.

Foresting: This component generates multiple initial programs that act as starting points for MCTS. In this process, initial programs are generated using either the same base prompt or diversified prompts obtained by randomly appending instruction statements.

Scouting: This component provides feedback on the repair instruction used during the refinement process and distills general insights that can be shared across search branches. These insights are incorporated into the prompt as auxiliary information when an LLM generates reflections and repair instructions. This enables the promotion of effective repair instructions and the suppression of unproductive ones.

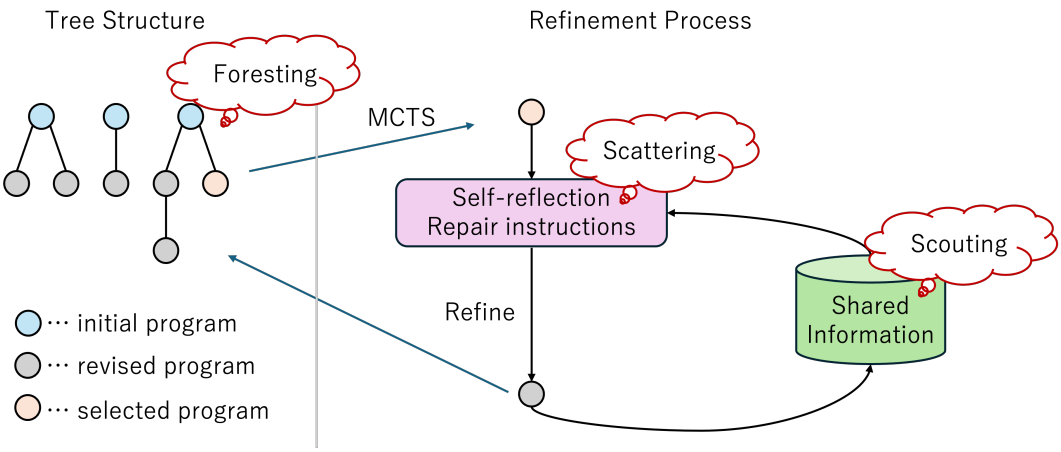


Figure 3: **Overview of SFS.** Adapted from Light et al. (2025). SFS builds on MCTS and incorporates techniques such as careful seed initialization and text-based optimization. SCATTERING diversifies refinements via dynamic prompt variation, FORESTING provides multiple starting programs for MCTS, and SCOUTING propagates insights about repair instructions across the search process.

C OVERVIEW OF ITERATIVE REFINEMENT OF REPAIR INSTRUCTIONS (IRRI)

Algorithm 1 presents the pseudo code of IRRI, following the description in Section 3.4. Figure 4 illustrates an overview of IRRI, and Figure 5 shows the difference between existing methods and IRRI.

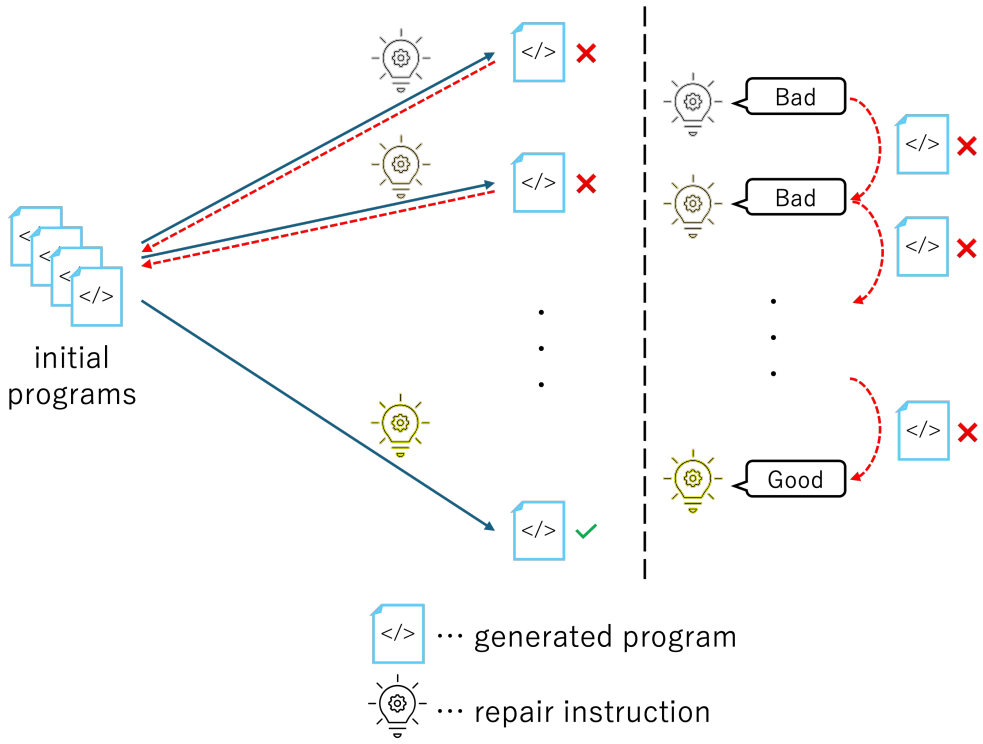
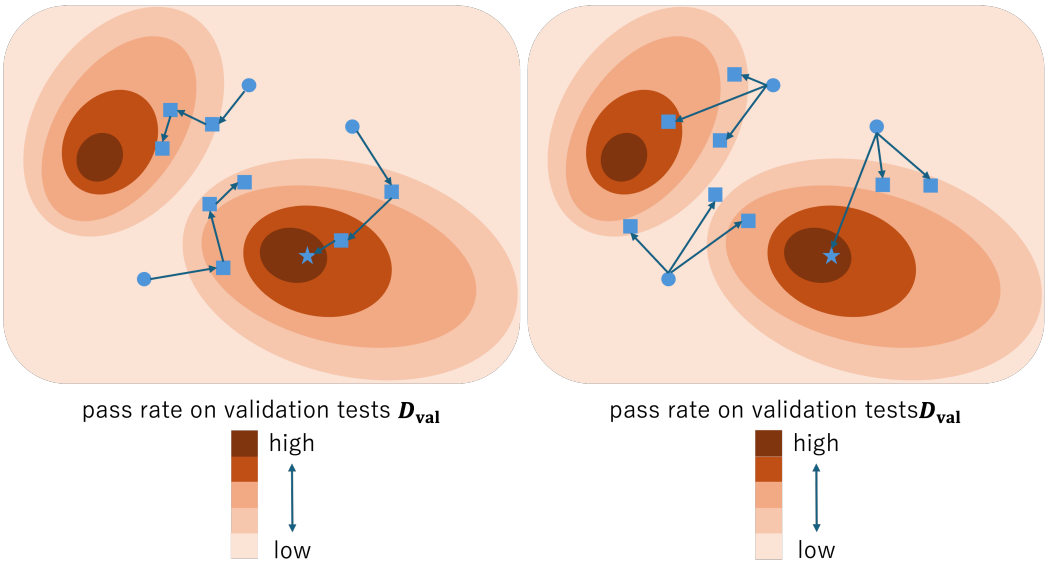


Figure 4: **Overview of IRRI.** To capture the characteristics of refinement in the width direction, we introduce a self-correction method that refines initial programs through iterative feedback on repair instructions.

Algorithm 1 ITERATIVE REFINEMENT OF REPAIR INSTRUCTIONS (IRRI)

Require: LLM $\mathcal{M}$, program generation task $x = \langle p, D_{\text{gt}} \rangle$
Require: Number of initial programs n
Require: Number of repair instructions generated per step m
Require: Maximum number of refinements $T_{\max}$
Generate a validation test-case set $D_{\text{val}} \leftarrow \mathcal{M}(p)$
Initial Generation
$i \leftarrow 1$
for $i \leq n$ **do**
    Generate initial program $c_i \leftarrow \mathcal{M}(p)$
    if c_i passes all validation test cases D_{val} **then**
        Terminate the inference process
    end if
    $i \leftarrow i + 1$
end for
Refinement Process
Shared information $I_1 \leftarrow \varnothing$
$t \leftarrow 1$
while $t \leq T_{max}$ **do**
    for $c_t \in \{c_1, \ldots, c_n\}$ **do**
        Generate repair instructions $r_1, \ldots, r_m \leftarrow \mathcal{M}(p, c_t, I_t)$
        for $r_t \in \{r_1, \ldots, r_m\}$ **do**
            Refine program $c_{\text{ref}} \leftarrow \mathcal{M}(p, c_t, r_t)$
            if c_{ref} passes all validation test cases D_{val} **then**
                Terminate the inference process
            end if
            $t \leftarrow t + 1$
            if $t > T_{max}$ **then**
                Terminate the inference process
            end if
            Feedback for Repair Instructions
            Update shared information $I_{t+1} \leftarrow \mathcal{M}(p, c_t, I_t, r_t, c_{\text{ref}})$
        end for
    end for
end while



(a) Existing methods (b) IRRI

Figure 5: **Difference in the refinement process between existing methods and IRRI.** Circles denote initial programs, squares denote revised programs, and stars denote correct programs.

D PROOFS

D.1 PROOF OF THEOREM 1

Using Assumption 2, the following lemma holds.

Lemma 1. *Under Assumption 2,*

$$\Pr[t \geq t_c] \geq 1 - (1 - q^{Line}) \exp\left(-\pi^{Line} A_\tau \left(1 - e^{-\tau(t-1)}\right)\right) \quad \left(A_\tau = \frac{e^{-2\tau}}{1 - e^{-\tau}}\right). \tag{20}$$

Proof. By the complementary event and Assumption 2,

$$\Pr[t \geq t_c] = 1 - \Pr[t < t_c] \tag{21}$$

$$= 1 - \Pr[H^\star_{c_1} = \varnothing] \prod_{s=1}^{t} (1 - \Pr[Z_{c_s} = 1 \mid c_s = \text{unknown}] \Pr[c_s = \text{unknown}]) \tag{22}$$

$$= 1 - (1 - q^{\text{Line}}) \prod_{s=2}^{t} \left(1 - \Pr[Z_{c_s} = 1 \mid c_s = \text{unknown}](1 - q_s^{\text{old}})\right) \tag{23}$$

$$= 1 - (1 - q^{\text{Line}}) \prod_{s=2}^{t} (1 - \Pr[Z_{c_s} = 1 \mid c_s = \text{unknown}] \exp(-\tau s)) \,. \tag{24}$$

For any time t, we have

$$\Pr[Z_{c_t} = 1 \mid c_t = \text{unknown}] \geq \pi^{\text{Line}} \tag{25}$$

from Assumption 2, so for each term

$$1 - \Pr[Z_{c_s} = 1 \mid c_s = \text{unknown}] \exp(-\tau s) \leq 1 - \pi^{\text{Line}} \exp(-\tau s) \tag{26}$$

holds. It follows that

$$\prod_{s=2}^{t} (1 - \Pr[Z_{c_s} = 1 \mid c_s = \text{unknown}] \exp(-\tau s)) \leq \prod_{s=2}^{t} \left(1 - \pi^{\text{Line}} \exp(-\tau s)\right), \tag{27}$$

and hence

$$\Pr[t \geq t_c] \geq 1 - (1 - q^{\text{Line}}) \prod_{s=2}^{t} \left(1 - \pi^{\text{Line}} \exp(-\tau s)\right). \tag{28}$$

Now, applying the inequality

$$1 - x \leq e^{-x} \tag{29}$$

to each term gives

$$1 - \pi^{\text{Line}} \exp(-\tau s) \leq \exp\left(-\pi^{\text{Line}} e^{-\tau s}\right). \tag{30}$$

Therefore,

$$\prod_{s=2}^{t} \left(1 - \pi^{\text{Line}} \exp(-\tau s)\right) \leq \prod_{s=2}^{t} \exp\left(-\pi^{\text{Line}} e^{-\tau s}\right) \tag{31}$$

$$= \exp\left(-\pi^{\text{Line}} \sum_{s=2}^{t} e^{-\tau s}\right) \tag{32}$$

$$= \exp\left(-\pi^{\text{Line}} e^{-2\tau} \frac{1 - e^{-\tau(t-1)}}{1 - e^{-\tau}}\right), \tag{33}$$

so that

$$\Pr[t \geq t_c] \geq 1 - (1 - q^{\text{Line}}) \exp\left(-\pi^{\text{Line}} A_\tau \left(1 - e^{-\tau(t-1)}\right)\right), \tag{34}$$

where

$$A_\tau = \frac{e^{-2\tau}}{1 - e^{-\tau}}. \tag{35}$$

□

Using this lemma, Theorem 1 can be proved as follows.

Proof. 1. At each time t, the refinement target is an unknown program with probability $1 - q_t^{\text{old}}$. In this case, regarding the hypothesis space that survives at time t,

$$\mathcal{V}_t = \mathcal{V}_{t-1} \cup H_{c_t}(e_t) \supseteq \mathcal{V}_{t-1} \tag{36}$$

holds.
On the other hand, the refinement target is a known program with probability q_t^{old}. In this case, the hypothesis space that survives at time t is expressed as

$$\mathcal{V}_t = (\mathcal{V}_{t-1} \cap H_{c_t}(e_t)) \cup (\mathcal{V}_{t-1} \setminus H_{c_t}). \tag{37}$$

For any program c_t and any observation e_t,

$$H_{c_t}(e_t) \subseteq H_{c_t} \tag{38}$$

and thus

$$\mathcal{V}_{t-1} \cap H_{c_t}(e_t) \subseteq \mathcal{V}_{t-1} \cap H_{c_t} \tag{39}$$

holds. From this,

$$\mathcal{V}_t = (\mathcal{V}_{t-1} \cap H_{c_t}(e_t)) \cup (\mathcal{V}_{t-1} \setminus H_{c_t}) \tag{40}$$

$$\subseteq (\mathcal{V}_{t-1} \cap H_{c_t}) \cup (\mathcal{V}_{t-1} \setminus H_{c_t}) \tag{41}$$

$$= \mathcal{V}_{t-1} \tag{42}$$

follows.
Furthermore, the probability that all refinement targets are known programs throughout the refinement process from time 1 to time t is given by

$$\Pr[\text{all refinement targets are known}] = \prod_{s=1}^{t} \Pr[c_s = \text{known}] \tag{43}$$

$$= \prod_{s=1}^{t} q_s^{\text{old}} \tag{44}$$

$$= \prod_{s=2}^{t} \left(1 - \exp(-\tau s)\right). \tag{45}$$

$$\tag{46}$$

Applying the inequality

$$1 - x \leq e^{-x} \tag{47}$$

to each term gives

$$1 - \exp(-\tau s) \leq \exp(-e^{-\tau s}). \tag{48}$$

Therefore,

$$\Pr[\text{all refinement targets are known}] \leq \prod_{s=2}^{t} \exp(-e^{-\tau s}) \tag{49}$$

$$= \exp\left(-\sum_{s=2}^{t} e^{-\tau s}\right) \tag{50}$$

$$= \exp\left(-e^{-2\tau} \frac{1 - e^{-\tau(t-1)}}{1 - e^{-\tau}}\right). \tag{51}$$

From this, in the refinement process from time 1 to time t, we obtain

$$\mathcal{V}_t \subseteq \mathcal{V}_{t-1} \subseteq \cdots \subseteq \mathcal{V}_0 \tag{52}$$

with probability at most

$$1 - \delta_{\text{Line}} = \exp\left(-A_\tau \left(1 - e^{-\tau(t-1)}\right)\right), \tag{53}$$

where

$$A_\tau = \frac{e^{-2\tau}}{1 - e^{-\tau}}. \tag{54}$$

2. We prove this by mathematical induction.
First, consider the case $t = t_c$.

(i) Case $t_c = 0$.
Since the true hypothesis space is non-empty at time 0,

$$\mathcal{T}_0 = H^\star_{c_1} (\neq \varnothing) \tag{55}$$

holds. Also, since

$$\mathcal{V}_0 = H_{c_1} \tag{56}$$

and

$$H^\star_{c_1} \subseteq H_{c_1}, \tag{57}$$

$$\mathcal{T}_0 = H^\star_{c_1} \subseteq H_{c_1} = \mathcal{V}_0 \tag{58}$$

holds.

(ii) Case $t_c \geq 1$.
Since the true hypothesis space first becomes non-empty at time t_c,

$$\mathcal{T}_{t_c} = H^\star_{c_{t_c}} (\neq \varnothing) \tag{59}$$

holds. Also, since program c_{t_c} should be unknown,

$$H_{c_{t_c}}(e_{t_c}) \subseteq \mathcal{V}_{t_c} \tag{60}$$

holds. Here, taking any $h^\star = (c_{t_c}, r^\star) \in H^\star_{c_{t_c}}$, by Assumption 1,

$$h^\star \in H_{c_{t_c}}(e_{t_c}) \tag{61}$$

holds, and so

$$H^\star_{c_{t_c}} \subseteq H_{c_{t_c}}(e_{t_c}). \tag{62}$$

Therefore,

$$\mathcal{T}_{t_c} = H^\star_{c_{t_c}} \subseteq H_{c_{t_c}}(e_{t_c}) \subseteq \mathcal{V}_{t_c} \tag{63}$$

holds.

Next, suppose that preservation of the true hypothesis holds at $t = s$ $(s \geq t_c)$; that is,

$$\mathcal{T}_s \subseteq \mathcal{V}_s. \tag{64}$$

(i) Case where the refinement target c_{s+1} at time $s+1$ is known.
$\mathcal{T}_{s+1} = \mathcal{T}_s$. Take any $h^\star = (c^\star, r^\star) \in \mathcal{T}_{s+1}$. If $c^\star = c_{s+1}$, then by Assumption 1,

$$h^\star \in H_{c_{s+1}}(e_{s+1}) \tag{65}$$

holds. Also, by the induction hypothesis,

$$h^\star \in \mathcal{V}_s \tag{66}$$

holds as well, so

$$h^\star \in \mathcal{V}_s \cap H_{c_{s+1}}(e_{s+1}) \subseteq \mathcal{V}_{s+1}. \tag{67}$$

If $c^\star \neq c_{s+1}$, then

$$h^\star \notin H_{c_{s+1}}. \tag{68}$$

On the other hand, by the induction hypothesis,

$$h^\star \in \mathcal{V}_s, \tag{69}$$

so

$$h^\star \in \mathcal{V}_s \setminus H_{c_{s+1}} \subseteq \mathcal{V}_{s+1}. \tag{70}$$

Thus, for any $h^\star \in \mathcal{T}_{s+1}$,

$$h^\star \in \mathcal{V}_{s+1} \tag{71}$$

holds, and hence

$$\mathcal{T}_{s+1} = \mathcal{T}_s \subseteq \mathcal{V}_{s+1} \tag{72}$$

holds.

(ii) Case where the refinement target c_{s+1} at time $s+1$ is unknown.
$\mathcal{T}_{s+1} = \mathcal{T}_s \cup H^\star_{c_{s+1}}$. Taking any $h^\star = (c_{s+1}, r^\star) \in H^\star_{c_{s+1}}$, by Assumption 1,

$$h^\star \in H_{c_{s+1}}(e_{s+1}) \tag{73}$$

holds, and so

$$H^\star_{c_{s+1}} \subseteq H_{c_{s+1}}(e_{s+1}) \tag{74}$$

holds. Also, by the induction hypothesis,

$$\mathcal{T}_s \subseteq \mathcal{V}_s. \tag{75}$$

Thus,

$$\mathcal{T}_{s+1} = \mathcal{T}_s \cup H^\star_{c_{s+1}} \subseteq \mathcal{V}_s \cup H_{c_{s+1}}(e_{s+1}) = \mathcal{V}_{s+1}. \tag{76}$$

Therefore, the claim also holds for $t = s + 1$. By the above, preservation of the true hypothesis holds.

3. By Property 2, as events, $\{t \geq t_c\} = \{\mathcal{T}_t \neq \varnothing\} \subseteq \{\mathcal{V}_t \neq \varnothing\}$ holds. Therefore, by Lemma 1 and monotonicity of probability,

$$\Pr[\mathcal{V}_t \neq \varnothing] \geq \Pr[\mathcal{T}_t \neq \varnothing] = \Pr[t \geq t_c] \geq 1 - (1 - q^{\text{Line}}) \exp\left(-\pi^{\text{Line}} A_\tau \left(1 - e^{-\tau(t-1)}\right)\right) \quad (77)$$

follows, where

$$A_\tau = \frac{e^{-2\tau}}{1 - e^{-\tau}}. \quad (78)$$

□

D.2 Proof of Theorem 2

Under Assumption 3, the following lemma holds.

Lemma 2. *Under Assumption 3, if $|C_{init}| = N$, then*

$$\Pr[\mathcal{T}_0 \neq \varnothing] \geq 1 - (1 - q^{IRRI})^N. \quad (79)$$

Proof. $\mathcal{T}_0 \neq \varnothing$ holds when at least one $c \in C_{\text{init}}$ satisfies $Y_c = 1$. Thus, by Assumption 3 and considering the complementary event, we obtain

$$\Pr[\mathcal{T}_0 \neq \varnothing] = \Pr\left[\bigcup_{c \in C_{\text{init}}} \{Y_c = 1\}\right] = 1 - \prod_{c \in C_{\text{init}}} \Pr[Y_c = 0] \geq 1 - (1 - q^{\text{IRRI}})^N. \quad (80)$$

□

Using this lemma, Theorem 2 can be proved as follows.

Proof. 1. For any program c_t and any observation e_t,

$$H_{c_t}(e_t) \subseteq H_{c_t} \quad (81)$$

implies

$$\mathcal{V}_{t-1} \cap H_{c_t}(e_t) \subseteq \mathcal{V}_{t-1} \cap H_{c_t}. \quad (82)$$

From this, we obtain

$$\begin{aligned} \mathcal{V}_t &= (\mathcal{V}_{t-1} \cap H_{c_t}(e_t)) \cup (\mathcal{V}_{t-1} \setminus H_{c_t}) && (83)\\ &\subseteq (\mathcal{V}_{t-1} \cap H_{c_t}) \cup (\mathcal{V}_{t-1} \setminus H_{c_t}) && (84)\\ &= \mathcal{V}_{t-1}. && (85) \end{aligned}$$

Hence, monotonic non-increasing holds.

2. The case $\mathcal{T}_0 = \varnothing$ holds trivially, so we consider only the case $\mathcal{T}_0 \neq \varnothing$ and prove the claim by mathematical induction.
First, at $t = 0$, since for any $c \in C_{\text{init}}$,

$$H_c^\star \subseteq H_c, \quad (86)$$

we have

$$\mathcal{T}_0 = \bigcup_{c \in C_{\text{init}}} H_c^\star \subseteq \bigcup_{c \in C_{\text{init}}} H_c = \mathcal{V}_0, \quad (87)$$

so the claim holds.
Next, suppose that preservation of the true hypothesis holds at $t = s - 1$ $(s \geq 1)$; that is,

$$\mathcal{T}_0 \subseteq \mathcal{V}_{s-1}. \quad (88)$$

Since $\mathcal{T}_0 \neq \varnothing$, take any $h^\star = (c^\star, r^\star) \in \mathcal{T}_0$. If $c^\star = c_s$, then by Assumption 1,

$$h^\star \in H_{c_s}(e_s) \quad (89)$$

holds. Also, by the induction hypothesis,

$$h^\star \in \mathcal{V}_{s-1} \tag{90}$$

holds as well, so

$$h^\star \in \mathcal{V}_{s-1} \cap H_{c_s}(e_s) \subseteq \mathcal{V}_s. \tag{91}$$

If $c^\star \neq c_s$, then

$$h^\star \notin H_{c_s}. \tag{92}$$

On the other hand, by the induction hypothesis,

$$h^\star \in \mathcal{V}_{s-1}, \tag{93}$$

so

$$h^\star \in \mathcal{V}_{s-1} \setminus H_{c_s} \subseteq \mathcal{V}_s. \tag{94}$$

Thus, for any $h^\star \in \mathcal{T}_0$,

$$h^\star \in \mathcal{V}_s \tag{95}$$

holds, so the claim also holds for $t = s$.
By the above, preservation of the true hypothesis holds.

3. By Property 2, as events, $\{\mathcal{T}_0 \neq \varnothing\} \subseteq \{\mathcal{V}_t \neq \varnothing\}$ holds. Therefore, by Lemma 2 and monotonicity of probability,

$$\Pr[\mathcal{V}_t \neq \varnothing] \geq \Pr[\mathcal{T}_0 \neq \varnothing] \geq 1 - (1 - q^{\text{IRRI}})^N \tag{96}$$

follows. □

D.3 Proof of Theorem 3

Under Assumption 4, the following lemma holds.

Lemma 3. *Under Assumption 4, if $|C_{init}| = N$, then*

$$\Pr[t \geq t_c] \geq 1 - (1 - q^{SFS})^N \exp\left(-\pi^{SFS}\theta^{SFS}\left(t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}}\right)\right). \tag{97}$$

Proof. Consider the probability $\Pr[t < t_c]$. By the independence in Assumption 4,

$$\Pr[t < t_c] = \underbrace{\left(\prod_{c \in C_{\text{init}}} \Pr[Y_c = 0]\right)}_{(a)} \cdot \underbrace{\left(\prod_{s=1}^{t} \left(1 - \Pr[Z_{c_s} = 1 \mid c_s = \text{unknown}] \Pr[c_s = \text{unknown}]\right)\right)}_{(b)}. \tag{98}$$

For (a), by the uniform lower bound in Assumption 4, we have

$$\Pr[Y_c = 1] \geq q^{\text{SFS}} \tag{99}$$

for any $c \in C_{\text{init}}$, and therefore

$$\prod_{c \in C_{init}} \Pr[Y_c = 0] = \prod_{c \in C_{init}} (1 - \Pr[Y_c = 1]) \tag{100}$$

$$\leq (1 - q^{\text{SFS}})^N. \tag{101}$$

For each term in (b), by the uniform lower bound in Assumption 4,

$$\Pr[Z_{c_s} = 1 \mid c_s = \text{unknown}] \geq \pi^{\text{SFS}}. \tag{102}$$

Moreover, by Assumption 4, for any time t and any depth $k\ (\geq 1)$,

$$q_t^{\text{old}}(k) \leq 1 - \theta^{\text{SFS}} \tag{103}$$

holds, and hence

$$\Pr[c_s = \text{unknown}] = \sum_{k=1}^{K_{s-1}} q_s^{\text{dep}}(k)\left(1 - q_s^{\text{old}}(k)\right) \tag{104}$$

$$\geq \theta^{\text{SFS}} \sum_{k=1}^{K_{s-1}} q_s^{\text{dep}}(k) \tag{105}$$

$$= \theta^{\text{SFS}}\left(1 - q_s^{\text{dep}}(0)\right) \tag{106}$$

$$= \theta^{\text{SFS}}\left(1 - e^{-\gamma(s-1)}\right). \tag{107}$$

Combining these, we obtain the inequality

$$1 - \Pr[Z_{c_s} = 1 \mid c_s = \text{unknown}]\Pr[c_s = \text{unknown}] \leq 1 - \pi^{\text{SFS}}\theta^{\text{SFS}}\left(1 - e^{-\gamma(s-1)}\right) \tag{108}$$

for each term. Applying the inequality

$$1 - x \leq e^{-x}, \tag{109}$$

we further obtain

$$1 - \Pr[Z_{c_s} = 1 \mid c_s = \text{unknown}]\Pr[c_s = \text{unknown}] \leq \exp\left(-\pi^{\text{SFS}}\theta^{\text{SFS}}\left(1 - e^{-\gamma(s-1)}\right)\right). \tag{110}$$

Altogether, (b) satisfies

$$\prod_{s=1}^{t}\left(1 - \Pr[Z_{c_s} = 1 \mid c_s = \text{unknown}]\Pr[c_s = \text{unknown}]\right) \leq \prod_{s=1}^{t}\exp\left(-\pi^{\text{SFS}}\theta^{\text{SFS}}\left(1 - e^{-\gamma(s-1)}\right)\right) \tag{111}$$

$$= \exp\left(-\pi^{\text{SFS}}\theta^{\text{SFS}}\sum_{s=1}^{t}\left(1 - e^{-\gamma(s-1)}\right)\right) \tag{112}$$

$$= \exp\left(-\pi^{\text{SFS}}\theta^{\text{SFS}}\left(t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}}\right)\right). \tag{113}$$

Therefore,

$$\Pr[t < t_c] \leq (1 - q^{\text{SFS}})^N \exp\left(-\pi^{\text{SFS}}\theta^{\text{SFS}}\left(t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}}\right)\right), \tag{114}$$

and hence

$$\Pr[t \geq t_c] = 1 - \Pr[t < t_c] \geq 1 - (1 - q^{\text{SFS}})^N \exp\left(-\pi^{\text{SFS}}\theta^{\text{SFS}}\left(t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}}\right)\right) \tag{115}$$

holds. □

Using this lemma, Theorem 3 can be proved as follows.

Proof. 1. At each time t, the probability that the refinement target is unknown is given by

$$\sum_{k=1}^{K_{t-1}} q_t^{\text{dep}}(k)\left(1 - q_t^{\text{old}}(k)\right). \tag{116}$$

In this case, regarding the hypothesis space that survives at time t,

$$\mathcal{V}_t = \mathcal{V}_{t-1} \cup H_{c_t}(e_t) \supseteq \mathcal{V}_{t-1} \tag{117}$$

holds.
On the other hand, the probability that the refinement target is known is given by

$$\sum_{k=0}^{K_{t-1}} q_t^{\text{dep}}(k) q_t^{\text{old}}(k). \tag{118}$$

In this case, the hypothesis space that survives at time t is expressed as

$$\mathcal{V}_t = (\mathcal{V}_{t-1} \cap H_{c_t}(e_t)) \cup (\mathcal{V}_{t-1} \setminus H_{c_t}). \tag{119}$$

For any program c_t and any observation e_t,

$$H_{c_t}(e_t) \subseteq H_{c_t} \tag{120}$$

and thus

$$\mathcal{V}_{t-1} \cap H_{c_t}(e_t) \subseteq \mathcal{V}_{t-1} \cap H_{c_t} \tag{121}$$

holds. From this,

$$\begin{aligned} \mathcal{V}_t &= (\mathcal{V}_{t-1} \cap H_{c_t}(e_t)) \cup (\mathcal{V}_{t-1} \setminus H_{c_t}) && (122)\\ &\subseteq (\mathcal{V}_{t-1} \cap H_{c_t}) \cup (\mathcal{V}_{t-1} \setminus H_{c_t}) && (123)\\ &= \mathcal{V}_{t-1} && (124) \end{aligned}$$

follows.
Furthermore, the probability that all refinement targets are known programs throughout the refinement process from time 1 to time t satisfies

$$\begin{aligned} \Pr[\text{all refinement targets are known}] &= \prod_{s=1}^{t} \Pr[c_s = \text{known}] && (125)\\ &= \prod_{s=1}^{t} \left(\sum_{k=0}^{K_{s-1}} q_s^{\text{dep}}(k) q_s^{\text{old}}(k) \right) && (126)\\ &= \prod_{s=1}^{t} \left(q_s^{\text{dep}}(0) q_s^{\text{old}}(0) + \sum_{k=1}^{K_{s-1}} q_s^{\text{dep}}(k) q_s^{\text{old}}(k) \right) && (127)\\ &\geq \prod_{s=1}^{t} q_s^{\text{dep}}(0) q_s^{\text{old}}(0) && (128)\\ &= \prod_{s=1}^{t} q_s^{\text{dep}}(0) && (129)\\ &= \prod_{s=1}^{t} \exp\left(-\gamma(s-1)\right) && (130)\\ &= \exp\left(-\gamma \sum_{s=1}^{t} (s-1) \right) && (131)\\ &= \exp\left(-\gamma \frac{t(t-1)}{2} \right). && (132) \end{aligned}$$

From this, in the refinement process from time 1 to time t, we obtain

$$\mathcal{V}_t \subseteq \mathcal{V}_{t-1} \subseteq \cdots \subseteq \mathcal{V}_0 \tag{133}$$

with probability at least

$$1 - \delta_{\text{SFS}} = \exp\left(-\gamma \frac{t(t-1)}{2} \right). \tag{134}$$

2. We prove this by mathematical induction.
First, consider the case $t = t_c$.

(i) Case $t_c = 0$.
Since for any $c \in C_{\text{init}}$

$$H_c^\star \subseteq H_c, \tag{135}$$

we have

$$\mathcal{T}_0 = \bigcup_{c \in C_{\text{init}}} H_c^\star \subseteq \bigcup_{c \in C_{\text{init}}} H_c = \mathcal{V}_0, \tag{136}$$

so the claim holds.

(ii) Case $t_c \geq 1$.
Since the true hypothesis space first becomes non-empty at time t_c,

$$\mathcal{T}_{t_c} = H^\star_{c_{t_c}} (\neq \varnothing) \tag{137}$$

holds. Also, since program c_{t_c} should be unknown,

$$H_{c_{t_c}}(e_{t_c}) \subseteq \mathcal{V}_{t_c} \tag{138}$$

holds. Here, taking any $h^\star = (c_{t_c}, r^\star) \in H^\star_{c_{t_c}}$, by Assumption 1,

$$h^\star \in H_{c_{t_c}}(e_{t_c}) \tag{139}$$

holds, and so

$$H^\star_{c_{t_c}} \subseteq H_{c_{t_c}}(e_{t_c}). \tag{140}$$

Therefore,

$$\mathcal{T}_{t_c} = H^\star_{c_{t_c}} \subseteq H_{c_{t_c}}(e_{t_c}) \subseteq \mathcal{V}_{t_c} \tag{141}$$

holds.

Next, suppose that preservation of the true hypothesis holds at $t = s$ $(s \geq t_c)$; that is,

$$\mathcal{T}_s \subseteq \mathcal{V}_s. \tag{142}$$

(i) Case where the refinement target c_{s+1} at time $s+1$ is known.
$\mathcal{T}_{s+1} = \mathcal{T}_s$. Take any $h^\star = (c^\star, r^\star) \in \mathcal{T}_{s+1}$. If $c^\star = c_{s+1}$, then by Assumption 1,

$$h^\star \in H_{c_{s+1}}(e_{s+1}) \tag{143}$$

holds. Also, by the induction hypothesis,

$$h^\star \in \mathcal{V}_s \tag{144}$$

holds as well, so

$$h^\star \in \mathcal{V}_s \cap H_{c_{s+1}}(e_{s+1}) \subseteq \mathcal{V}_{s+1}. \tag{145}$$

If $c^\star \neq c_{s+1}$, then

$$h^\star \notin H_{c_{s+1}}. \tag{146}$$

On the other hand, by the induction hypothesis,

$$h^\star \in \mathcal{V}_s, \tag{147}$$

so

$$h^\star \in \mathcal{V}_s \setminus H_{c_{s+1}} \subseteq \mathcal{V}_{s+1}. \tag{148}$$

Thus, for any $h^\star \in \mathcal{T}_{s+1}$,

$$h^\star \in \mathcal{V}_{s+1} \tag{149}$$

holds, and hence

$$\mathcal{T}_{s+1} = \mathcal{T}_s \subseteq \mathcal{V}_{s+1} \tag{150}$$

holds.

(ii) Case where the refinement target c_{s+1} at time $s+1$ is unknown.
$\mathcal{T}_{s+1} = \mathcal{T}_s \cup H^\star_{c_{s+1}}$. Taking any $h^\star = (c_{s+1}, r^\star) \in H^\star_{c_{s+1}}$, by Assumption 1,

$$h^\star \in H_{c_{s+1}}(e_{s+1}) \tag{151}$$

holds, and so

$$H^\star_{c_{s+1}} \subseteq H_{c_{s+1}}(e_{s+1}) \tag{152}$$

holds. Also, by the induction hypothesis,

$$\mathcal{T}_s \subseteq \mathcal{V}_s. \tag{153}$$

Thus,

$$\mathcal{T}_{s+1} = \mathcal{T}_s \cup H^\star_{c_{s+1}} \subseteq \mathcal{V}_s \cup H_{c_{s+1}}(e_{s+1}) = \mathcal{V}_{s+1}. \tag{154}$$

Therefore, the claim also holds for $t = s+1$. By the above, preservation of the true hypothesis holds.

3. By Property 2, as events, $\{t \geq t_c\} = \{\mathcal{T}_t \neq \varnothing\} \subseteq \{\mathcal{V}_t \neq \varnothing\}$ holds. Therefore, by Lemma 3 and monotonicity of probability,

$$\Pr[\mathcal{V}_t \neq \varnothing] \geq \Pr[\mathcal{T}_t \neq \varnothing] = \Pr[t \geq t_c] \geq 1 - (1 - q^{\text{SFS}})^N \exp\left(-\pi^{\text{SFS}}\theta^{\text{SFS}}\left(t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}}\right)\right) \tag{155}$$

follows. □

E Further analysis and discussion

E.1 Sequential self-refinement method Line)

From Theorem 1, the hypothesis space $\mathcal{V}_t$ in Line can be non-decreasing or non-increasing depending on whether the refinement target c_t at time t is an unknown or known program. Furthermore, over the refinement process from time 1 to time t, all refinement targets are known, and consequently the hypothesis space becomes monotonically non-increasing with probability at most

$$\exp\left(-A_\tau\left(1 - e^{-\tau(t-1)}\right)\right). \tag{156}$$

This indicates that there is a limit to the extent to which Line can properly narrow down the hypothesis space. Moreover, the upper bound is monotonically decreasing in t and approaches

$$\exp\left(-A_\tau\right) \tag{157}$$

as $t \to \infty$. Therefore, the probability that all refinement targets are known, and consequently the hypothesis space is monotonically non-increasing, in Line decreases as t increases.

Theorem 1 also states that Line satisfies non-emptiness with probability at least a certain threshold. Here, we discuss the condition on time t required for the probability $\Pr[\mathcal{V}_t \neq \varnothing]$ to be at least $1 - \xi_{\text{Line}}$. Before doing so, we first organize the condition that ξ_{Line} must satisfy. The lower bound on the probability $\Pr[\mathcal{V}_t \neq \varnothing]$,

$$1 - (1 - q^{\text{Line}}) \exp\left(-\pi^{\text{Line}} A_\tau\left(1 - e^{-\tau(t-1)}\right)\right), \tag{158}$$

is monotonically increasing in t and approaches

$$1 - (1 - q^{\text{Line}}) \exp\left(-\pi^{\text{Line}} A_\tau\right) \tag{159}$$

as $t \to \infty$. From this, we must have

$$\xi_{\text{Line}} \geq (1 - q^{\text{Line}}) \exp\left(-\pi^{\text{Line}} A_\tau\right). \tag{160}$$

This means that no matter how good Line behaves, the lower bound on the probability $\Pr[\mathcal{V}_t \neq \varnothing]$ never reaches 1. Next, considering the condition on time t,

$$t \geq 1 - \frac{1}{\tau}\log\left(1 + \frac{1}{\pi^{\text{Line}} A_\tau}\log\frac{\xi_{\text{Line}}}{1 - q^{\text{Line}}}\right). \tag{161}$$

Thus, the minimum time $t_{\min}$ at which the probability $\Pr[\mathcal{V}_t \neq \varnothing]$ becomes at least $1 - \xi_{\text{Line}}$ is obtained as

$$t_{\min} = \left\lceil 1 - \frac{1}{\tau} \log \left(1 + \frac{1}{\pi^{\text{Line}} A_\tau} \log \frac{\xi_{\text{Line}}}{1 - q^{\text{Line}}} \right) \right\rceil . \tag{162}$$

Under the condition imposed on ξ_{Line}, as the value of ξ_{Line} decreases, the right-hand side of the above expression increases. This shows that, in order for non-emptiness to hold with sufficiently high probability, the time $t_{\min}$ needs to be relatively large, requiring more correction steps.

E.2 Parameter and upper-bound estimation in Line

Recall that we modeled the probability that the refinement target c_t is a previously known program as

$$q_t^{\text{old}} = 1 - \exp(-\tau t) \quad (t \geq 2). \tag{163}$$

We estimated the parameter τ appearing in this model empirically by applying Line to real datasets. Specifically, we used gpt-4o-mini as the underlying model and applied Line to four datasets: HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), APPS (Hendrycks et al., 2021) and CodeContests (Li et al., 2022). For each task, Line generates a single initial program and then performs up to 15 rounds of revision.

At each time step $t \in \{2, \cdots, 15\}$, we determined whether the program being revised had appeared previously and used these observations to compute the empirical probability $\hat{q}_t^{\text{old}}$ that the refinement target c_t is known. We then fit the model $q_t^{\text{old}} = 1 - \exp(-\tau t)$ to the resulting data points $\{(t, \hat{q}_t^{\text{old}})\}_{t=2}^{15}$ via non-linear least squares to obtain an estimate of τ. Using this estimated value of τ, we computed the upper bound on the probability that the hypothesis space is monotonically non-increasing up to time $t = 15$.

The results are shown in Tables 4 and 5. The extremely small estimated upper bound shows that, in Line, all refinement targets are almost never known, and consequently the hypothesis space is almost never monotonically non-increasing.

Table 4: **Estimated values of τ.**

Dataset	τ
HumanEval	0.1497 ± 0.0168
MBPP	0.1490 ± 0.0113
APPS	0.1109 ± 0.0057
CodeContests	0.1011 ± 0.0062

Table 5: **Estimated values of $1 - \delta_{\text{Line}}$.**

Dataset	$1 - \delta_{\text{Line}} \quad (t = 15)$
HumanEval	9.325×10^{-3}
MBPP	9.134×10^{-3}
APPS	2.442×10^{-3}
CodeContests	1.607×10^{-3}

E.3 Scattered Forest Search (SFS)

From Theorem 3, the hypothesis space $\mathcal{V}_t$ in SFS can be non-decreasing or non-increasing depending on whether the refinement target c_t at time t is an unknown or known program. Furthermore, over the refinement process from time 1 to time t, it becomes monotonically non-increasing with probability at least

$$\exp\left(-\gamma \frac{t(t-1)}{2} \right). \tag{164}$$

The condition for this probability to be at least $1 - \epsilon_{\text{SFS}}$ is

$$t \leq \frac{1 + \sqrt{1 + \frac{8}{\gamma} \log \frac{1}{1 - \epsilon_{\text{SFS}}}}}{2}. \tag{165}$$

This implies that, for a relatively short refinement process satisfying the above equation, the hypothesis space may approximately exhibit monotonic non-increasing behavior. In other words, this suggests that SFS can properly narrow down the hypothesis space within a small number of revision steps.

Moreover, when the refinement process is relatively short, MCTS prioritizes breadth-wise exploration, so the refinement targets are drawn from the set of initial programs with high probability. This means that SFS exhibits refinement behavior similar to that of IRRI.

Theorem 3 also states that SFS satisfies non-emptiness with probability at least a certain threshold. Let

$$G_t(N) = 1 - (1 - q^{\text{SFS}})^N \exp\left(-\pi^{\text{SFS}}\theta^{\text{SFS}}\left(t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}}\right)\right). \tag{166}$$

Since $q^{\text{SFS}} \in (0, 1)$, we have $0 < 1 - q^{\text{SFS}} < 1$. Therefore, $G_t(N)$is monotonically increasing in N and satisfies

$$\lim_{N\to\infty} G_t(N) = 1 - \lim_{N\to\infty}(1 - q^{\text{SFS}})^N \exp\left(-\pi^{\text{SFS}}\theta^{\text{SFS}}\left(t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}}\right)\right) = 1. \tag{167}$$

This indicates that increasing the number of initial programs generated has the benefit of increasing the probabilities that $\mathcal{T}_t \neq \varnothing$ and $\mathcal{V}_t \neq \varnothing$.

Compared with Theorem 2, an additional factor

$$\exp\left(-\pi^{\text{SFS}}\theta^{\text{SFS}}\left(t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}}\right)\right) \tag{168}$$

is multiplied onto the second term in SFS. Here, we consider the function

$$J(t) = t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}}, \tag{169}$$

noting that $\gamma > 0$. From

$$J'(t) = 1 - \frac{\gamma e^{-\gamma t}}{1 - e^{-\gamma}}, \tag{170}$$

$$J''(t) = \frac{\gamma^2 e^{-\gamma t}}{1 - e^{-\gamma}} > 0, \tag{171}$$

the function $J(t)$ is convex. Since $J(t) = 0$ at $t = 0, 1$, it follows that for $t > 1$,

$$t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}} > 0 \tag{172}$$

holds. Taking into account that $\pi^{\text{SFS}} > 0$ and $\theta^{\text{SFS}} > 0$, we obtain

$$\exp\left(-\pi^{\text{SFS}}\theta^{\text{SFS}}\left(t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}}\right)\right) < 1, \tag{173}$$

and hence

$$1 - (1 - q^{\text{SFS}})^N \exp\left(-\pi^{\text{SFS}}\theta^{\text{SFS}}\left(t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}}\right)\right) > 1 - (1 - q^{\text{SFS}})^N. \tag{174}$$

This reflects the exploratory capability of SFS in the refinement steps and demonstrates its effectiveness. However, unless t is sufficiently large, no clear difference emerges between the two.

Here, we discuss the condition on time t required for the probability $\Pr[\mathcal{V}_t \neq \varnothing]$ to be at least $1 - \xi_{\text{SFS}}$. Before doing so, we first organize the condition that ξ_{SFS} must satisfy. From the preceding discussion, we see that the function $J(t)$ is monotonically increasing for $t \geq 1$ and satisfies $J(t) \to \infty$ as $t \to \infty$. It follows that the lower bound on the probability $\Pr[\mathcal{V}_t \neq \varnothing]$ is monotonically increasing for $t \geq 1$ and converges to 1 as $t \to \infty$. This means that ξ_{SFS} can be taken to be arbitrarily small. Next, considering the condition on time t,

$$t - \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}} \geq -\frac{1}{\pi^{\text{SFS}}\theta^{\text{SFS}}} \log \frac{\xi_{\text{SFS}}}{(1 - q^{\text{SFS}})^N}. \tag{175}$$

Let

$$L = \frac{1}{1-e^{-\gamma}}, \quad \kappa = -\frac{1}{\pi^{\text{SFS}}\theta^{\text{SFS}}} \log \frac{\xi_{\text{SFS}}}{(1-q^{\text{SFS}})^N}, \tag{176}$$

and consider solving the equation

$$t - L(1-e^{-\gamma t}) = \kappa \iff t = \kappa + L - Le^{-\gamma t}. \tag{177}$$

Since we expect $\xi_{\text{SFS}} \leq (1-q^{\text{SFS}})^N$ with respect to ξ_{SFS}, we consider the case $\kappa \geq 0$. In addition, since $t \geq 0$, we seek only the larger of the two solutions. Applying the change of variables $\hat{t} = \gamma t - \gamma(\kappa + L)$, we obtain

$$\frac{\hat{t}}{\gamma} = -Le^{-\hat{t}}e^{-\gamma(\kappa+L)} \iff \hat{t}e^{\hat{t}} = -\gamma Le^{-\gamma(\kappa+L)}. \tag{178}$$

Hence, using the Lambert W function, this can be expressed as

$$\hat{t} = W_0\left(-\gamma Le^{-\gamma(\kappa+L)}\right) \tag{179}$$

$$= W_0\left(-\frac{\gamma}{1-e^{-\gamma}} \exp\left(-\gamma\left(\kappa + \frac{1}{1-e^{-\gamma}}\right)\right)\right), \tag{180}$$

so that the solution to the equation is given by

$$t = \kappa + \frac{1}{1-e^{-\gamma}} + \frac{1}{\gamma} W_0\left(-\frac{\gamma}{1-e^{-\gamma}} \exp\left(-\gamma\left(\kappa + \frac{1}{1-e^{-\gamma}}\right)\right)\right). \tag{181}$$

Given that the function $J(t)$ is monotonically increasing for $t \geq 1$, the condition for the probability $\Pr[\mathcal{V}_t \neq \varnothing]$ to be at least $1 - \xi_{\text{SFS}}$ is given by

$$t \geq \kappa + \frac{1}{1-e^{-\gamma}} + \frac{1}{\gamma} W_0\left(-\frac{\gamma}{1-e^{-\gamma}} \exp\left(-\gamma\left(\kappa + \frac{1}{1-e^{-\gamma}}\right)\right)\right). \tag{182}$$

Thus, the minimum time $t_{\min}$ at which the probability $\Pr[\mathcal{V}_t \neq \varnothing]$ becomes at least $1 - \xi_{\text{SFS}}$ is obtained as

$$t_{\min} = \left\lceil \kappa + \frac{1}{1-e^{-\gamma}} + \frac{1}{\gamma} W_0\left(-\frac{\gamma}{1-e^{-\gamma}} \exp\left(-\gamma\left(\kappa + \frac{1}{1-e^{-\gamma}}\right)\right)\right) \right\rceil, \tag{183}$$

where

$$\kappa = -\frac{1}{\pi^{\text{SFS}}\theta^{\text{SFS}}} \log \frac{\xi_{\text{SFS}}}{(1-q^{\text{SFS}})^N}. \tag{184}$$

As the value of ξ_{SFS}decreases, the value of κ increases, and the value of $t_{\min}$ also increases. This shows that, in order for non-emptiness to hold with sufficiently high probability, the time $t_{\min}$ needs to be relatively large, requiring more correction steps. However, SFS allows the value of κ and $t_{\min}$ to be decreased by increasing the number of initial programs generated, unlike Line. This means that, by enlarging the set of initial programs, the probability that non-emptiness is satisfied becomes higher even within a relatively small number of correction steps.

E.4 Parameter and lower-bound estimation in SFS

Recall that we modeled the probability that the refinement target c_t is a node at depth 0 as

$$q_t^{\text{dep}}(0) = \exp\left(-\gamma(t-1)\right) \quad (t \geq 1). \tag{185}$$

We estimated the parameter γ appearing in this model empirically by applying SFS to real datasets. Specifically, we used gpt-4o-mini as the underlying model and applied Line to four datasets: HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), APPS (Hendrycks et al., 2021) and CodeContests (Li et al., 2022). For each task, SFS generates five initial programs and then performs up to 11 rounds of revision.

At each time step $t \in \{1, \cdots, 11\}$, we determined whether the program being revised was a node at depth 0 and used these observations to compute the empirical probability $\hat{q}_t^{\text{dep}}(0)$ that the refinement target c_t is a node at depth 0. We then fit the model $q_t^{\text{dep}}(0) = \exp(-\gamma(t-1))$ to the resulting data points $\{(t, \hat{q}_t^{\text{old}})\}_{t=1}^{11}$ via non-linear least squares to obtain an estimate of γ. Using this estimated value of γ, we computed the lower bound on the probability that the hypothesis space is monotonically non-increasing up to time $t = 11$.

The results are shown in Tables 6 and 7. Since the estimated lower bound is non-negligible, we find that the hypothesis space in SFS is monotonically non-increasing with a certain probability. This suggests that SFS is more amenable to maintaining a monotonically non-increasing hypothesis space.

Table 6: **Estimated values of γ.**

Dataset	γ
HumanEval	0.01338 ± 0.00149
MBPP	0.03989 ± 0.00178
APPS	0.01832 ± 0.00231
CodeContests	0.01084 ± 0.00209

Table 7: **Estimated values of $1 - \delta_{\text{SFS}}$.**

Dataset	$1 - \delta_{\text{SFS}} \quad (t = 11)$
HumanEval	0.4791
MBPP	0.1115
APPS	0.3651
CodeContests	0.5508

F ADDITIONAL RESULTS FOR INFERENCE SCALING

The results are shown in Figures 6- 9.

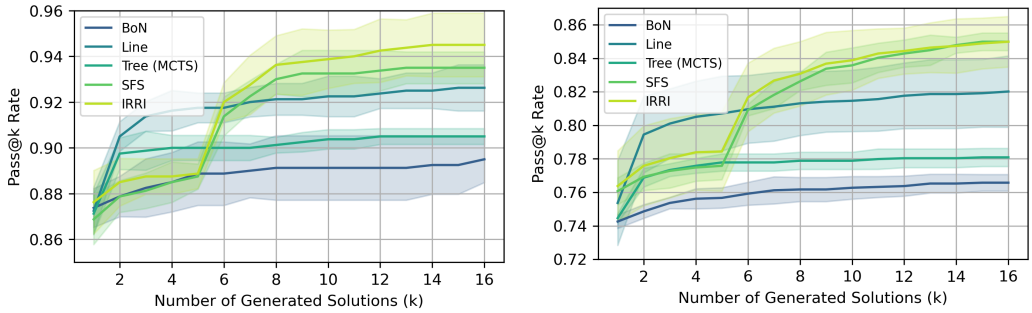


Figure 6: **Scaling curves for different methods on HumanEval using gpt-4o-mini (left) and on MBPP using gpt-4o-mini (right).**

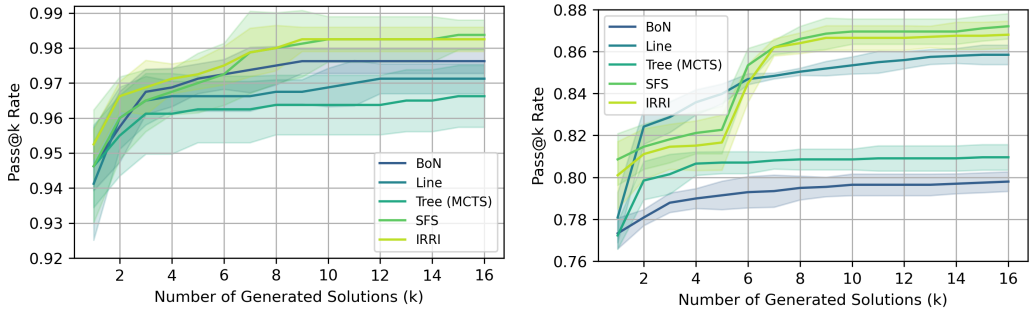


Figure 7: **Scaling curves for different methods on HumanEval using gpt-4.1-mini (left) and on MBPP using gpt-4.1-mini (right).**

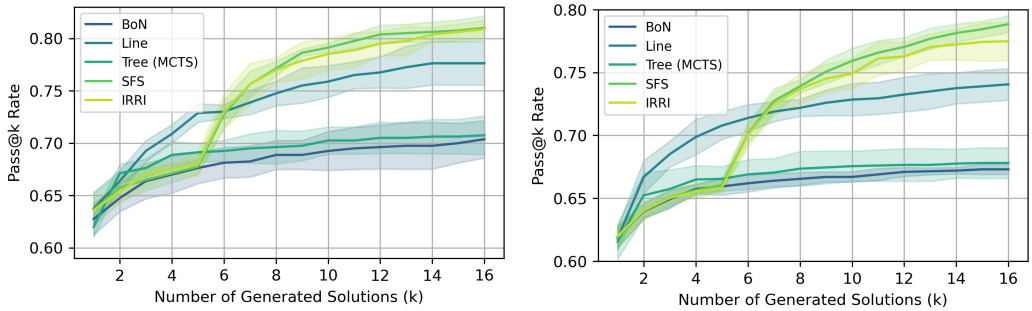


Figure 8: **Scaling curves for different methods on HumanEval using Llama-3.1-8B-Instruct (left) and on MBPP using Llama-3.1-8B-Instruct (right).**

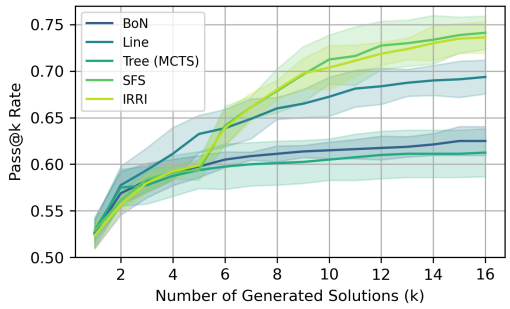


Figure 9: **Scaling curves for different methods on HumanEval using Llama-3.2-3B-Instruct.**

G Ablation Studies

G.1 Other benchmarks

We compared the inference scaling performance of SFS and IRRI on CodeContests (Li et al., 2022) with gpt-4o-mini as the base model. We also compared the inference scaling performance of SFS and IRRI on Leetcode (Guo et al., 2024) with Llama-3.1-8B-Instruct as the base model. Each curve represents the mean of three runs, and each shaded area indicates 95% confidence intervals based on the t-distribution. From Figure 10, we found that IRRI achieves inference scaling performance comparable to SFS.

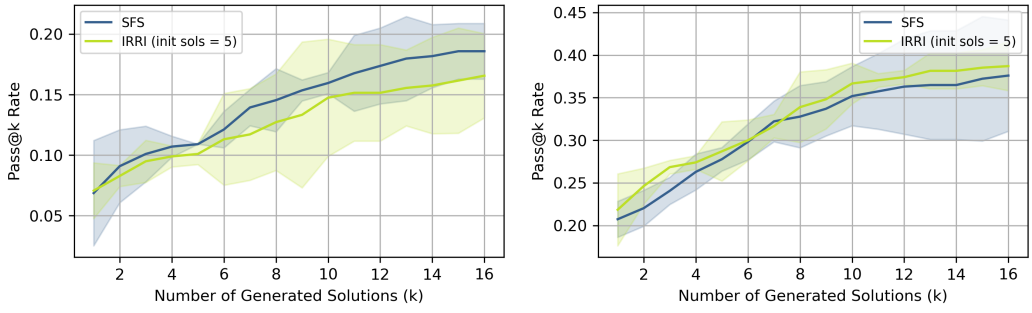


Figure 10: **Scaling curves for different methods on CodeContests using gpt-4o-mini (left) and on Leetcode using Llama-3.1-8B-Instruct (right).**

G.2 COMPUTATIONAL COST COMPARISON

To reveal the efficiency of IRRI, we compared SFS and IRRI in terms of the number of LLM calls, the number of output tokens, and the total number of tokens, reporting the average over five runs (Tables 8- 12). We found that IRRI requires fewer LLM calls and consumes fewer tokens than SFS, highlighting the advantages of IRRI.

Table 8: **Computational cost comparison on APPS using gpt-4o-mini.**

Metric	SFS	IRRI
Average LLM calls	30.95	29.64
Average output tokens	8228	7792
Average total tokens	56348	52900

Table 9: **Computational cost comparison on HumanEval using gpt-4o-mini (left) and on MBPP using gpt-4o-mini (right).**

Metric	SFS	IRRI
Average LLM calls	15.47	14.04
Average output tokens	3356	3009
Average total tokens	20714	18290

Metric	SFS	IRRI
Average LLM calls	14.23	13.11
Average output tokens	2725	2463
Average total tokens	16339	14574

Table 10: **Computational cost comparison on HumanEval using gpt-4.1-mini (left) and on MBPP using gpt-4.1-mini (right).**

Metric	SFS	IRRI
Average LLM calls	9.01	8.11
Average output tokens	2384	2118
Average total tokens	12396	10753

Metric	SFS	IRRI
Average LLM calls	9.51	8.24
Average output tokens	2186	1826
Average total tokens	11078	8986

Table 11: **Computational cost comparison on HumanEval using Llama-3.1-8B-Instruct (left) and on MBPP using Llama-3.1-8B-Instruct (right).**

Metric	SFS	IRRI
Average LLM calls	19.79	18.02
Average output tokens	5140	4907
Average total tokens	27037	23286

Metric	SFS	IRRI
Average LLM calls	20.97	19.37
Average output tokens	5109	4621
Average total tokens	26069	23286

Table 12: **Computational cost comparison on HumanEval using Llama-3.2-3B-Instruct (left) and on MBPP using Llama-3.2-3B-Instruct (right).**

Metric	SFS	IRRI
Average LLM calls	15.47	14.04
Average output tokens	3356	3009
Average total tokens	20714	18290

Metric	SFS	IRRI
Average LLM calls	14.23	13.11
Average output tokens	2725	2463
Average total tokens	16339	14574