LEARNING FROM FAILED HYPOTHESES: LOCAL ABDUCTION FOR NEURO-SYMBOLIC SELF-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Combining language models with symbolic solvers enables explicit and verifiable reasoning, but learning to construct symbolic solutions is hindered by the scarcity of annotations. Weakly supervised self-training addresses this challenge by learning from the model's own predictions verified against observed answers, yet new supervision still depends on generating a complete correct solution. We observe that failed solutions can retain useful structure, suggesting that supervision may be recovered before the model can solve a problem on its own. We propose Local Abductive Self-Training (LAST), which formulates this recovery as local abductive inference. Given a failed hypothesis, LAST searches its neighborhood for symbolic solutions consistent with the observed answer and uses model likelihood to select promising corrections. These corrections complement successful neural predictions and accumulate in trajectory memory to support subsequent learning. Across five geometry and numerical reasoning benchmarks, LAST improves average accuracy after six rounds by 7.84–10.69 and 4.83–12.40 percentage points over self-training baselines with 2B and 8B models, respectively. Further analyses show that LAST broadens supervision coverage while preserving solution quality, highlighting the value of failed hypotheses as a source of supervision.

1 INTRODUCTION

Large Language Models (LLMs) have demonstrated strong capabilities across many domains, with training data playing an important role. While large-scale pretraining equips LLMs with broad capabilities (Brown et al., 2020), high-quality domain-specific data help adapt these capabilities to specialized reasoning tasks (Yuan et al., 2023). In mathematical and logical reasoning, a common approach combines the model with a symbolic engine: the model formulates the problem as a symbolic solution, which is a formal representation for modeling or solving it. The engine then executes this representation to obtain the final answer (Lu et al., 2021; Amini et al., 2019; Shi et al., 2025). Such formulations capture the relationships needed to derive the answer, including geometric constraints expressed as equations (Zhang et al., 2023), quantitative dependencies represented by arithmetic expressions (Chen et al., 2021b), and premises and inference rules encoded in logical formulas (Pan et al., 2023). These neuro-symbolic systems make reasoning steps explicit, allowing them to be checked against task constraints.

However, teaching models to construct these symbolic solutions presents a data challenge. Compared with final answer labels, symbolic solution labels are scarce and often require substantial human annotation (Hong et al., 2021). This raises a natural question: how can a model learn to construct these solutions when the training data contain only problems and answers? Self-training provides a way to learn in this weakly supervised setting without requiring a stronger teacher or a reward model (Zelikman et al., 2022; Xu et al., 2025; Cheng et al., 2025). At each round, the model attempts to solve the training problems, and symbolic solutions that yield the known answers when executed are collected as supervision for the next round. The updated model then revisits these problems and may discover further solutions to learn from. However, this cycle faces a difficulty: **for a problem to provide a training target, the model must first solve it correctly.**

To address this difficulty, existing methods expand the model's exploration by approaching the same problem through different reasoning tasks. Beyond ordinary problem solving, STaR provides the known answer and prompts the model to generate a new reasoning trace (Zelikman et al., 2022),

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.

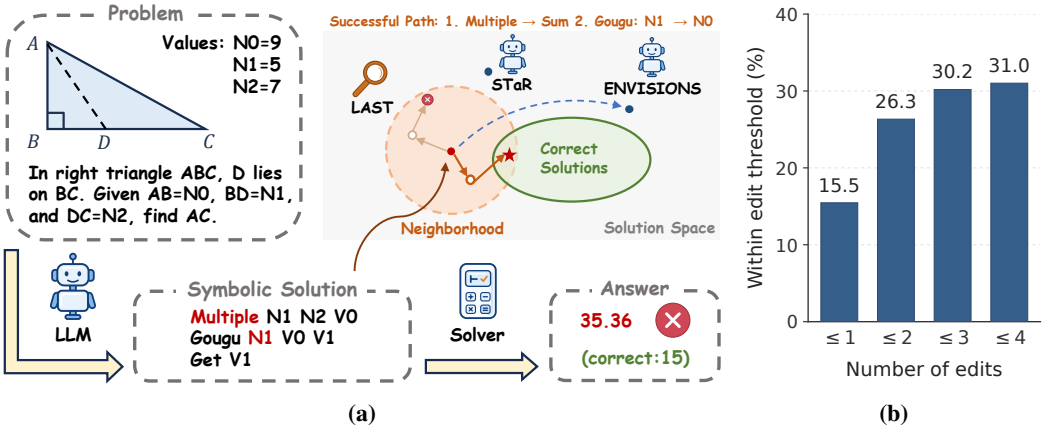


Figure 1: **(a)** Given a problem, the model produces a symbolic solution for the solver to execute. For unsolved problems, unlike previous methods that prompt the model to generate new solutions through alternative reasoning strategies, LAST starts directly from a failed solution and searches its neighborhood for an answer-consistent solution. **(b)** Each bar shows the percentage of parseable failed solutions that can be transformed into their reference solutions using at most the indicated number of edits. Experimental details are provided in Appendix C.

while ENVISIONS asks the model to refine its initial symbolic solution (Xu et al., 2025). These strategies create more opportunities for success, but discovering new labels still depends on the model generating complete correct solutions. Problems that remain unsolved are therefore difficult to turn into training examples.

We hypothesize, however, that a failed symbolic solution may retain useful structure, with errors confined to only part of the solution. Figure 1(b) provides evidence for this possibility. On the training set of the numerical reasoning benchmark FinQA (Chen et al., 2021b), 26.3% of the parseable failed solutions generated by Qwen3-VL-8B-Instruct (Bai et al., 2025) before self-training can be transformed into their reference solutions with at most two edits. Although these hypotheses fail as complete solutions, they remain structurally close to correct ones. This raises our central question: **must a model first generate a fully correct solution to provide supervision, or can supervision be recovered from the structure of its failed hypotheses?**

Building on this insight, we propose *Local Abductive Self-Training* (LAST), a framework that combines neural prediction and symbolic inference to discover supervision. We view each model-generated symbolic solution as a hypothesis about how to solve the problem. Given a failed hypothesis, LAST searches directly for a correction rather than prompting the model to generate a new solution. We formulate this recovery as local abductive inference: searching for an answer-consistent correction within a bounded neighborhood of the failed hypothesis.

To solve this abductive problem, LAST performs breadth-first beam search over the failed hypothesis's neighborhood, defined by domain-specific edit actions. The verified candidates are then ranked by their length-normalized log-likelihood under the model, and high-scoring revisions are retained as training targets. Figure 1(a) illustrates how LAST differs from prior methods. For problems with no correct solution from current predictions or trajectory memory, accepted corrections provide additional supervision. These corrections are retained alongside model-generated solutions in trajectory memory to train the next round's model, allowing failed hypotheses to contribute to subsequent learning.

Our contributions are:

- We shift the focus of self-training from expanding the model's search for complete correct solutions to recovering supervision from failed hypotheses. We hypothesize that these failures can result from local errors and provide empirical evidence for this view.
- We propose LAST, a self-training framework combining neural exploration with local abductive inference. When exploration fails, edit-based beam search finds answer-consistent

revisions near the failed hypothesis. High-likelihood revisions are retained in trajectory memory to supervise subsequent learning.

- We demonstrate substantial performance gains over self-training baselines across five geometry and numerical reasoning benchmarks with 2B and 8B models. Further analyses show that failed hypotheses guide the discovery of additional useful supervision, explaining how local abduction strengthens self-training.

2 Background and Related Work

2.1 Problem Setting

We study self-training of language models in neuro-symbolic reasoning systems. Given a problem x in domain D, a language model p_θ formulates the problem as a symbolic solution $z \leftarrow p_\theta(\cdot \mid x)$. A symbolic solver S executes z to yield $\hat{y} = S(z) = \mathcal{R}_\theta(x)$, where $\mathcal{R}_\theta$ denotes the complete system. In the weakly supervised setting, the training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ contains only problem inputs and answers, without symbolic solution labels. Our goal is to improve the model's ability to formulate problems as symbolic solutions through self-training.

Symbolic solutions are represented as sequences of instructions. Let $\mathcal{F}_D$ denote the operator set for domain D, and let $\mathcal{D}_x$ denote the available operands: quantities from the problem context x, domain constants, and variables permitted by the symbolic language. Formally, a symbolic solution is $z = (s_1, \ldots, s_m)$, where each instruction $s_j = o_j(a_{j,1}, \ldots, a_{j,k_j})$ applies an operator $o_j \in \mathcal{F}_D$ to k_j operands $a_{j,k} \in \mathcal{D}_x$ according to its signature.

Example 1. Consider the problem in Figure 1(a): "In right triangle ABC, D lies on BC. Given $AB = N_0$, $BD = N_1$, and $DC = N_2$, find AC." The symbolic solution is $z = (\mathrm{Sum}(N_1, N_2, V_0), \mathrm{Gougu}(N_0, V_0, V_1), \mathrm{Get}(V_1))$, which encodes $N_1 + N_2 = V_0$ and $N_0^2 + V_0^2 = V_1^2$. The solver returns the hypotenuse length $S(z) = \sqrt{N_0^2 + (N_1 + N_2)^2}$.

2.2 Related Work

Reasoning with symbolic solutions. Language models formulate problems as symbolic solutions that external engines execute to derive answers (Gao et al., 2023; Chen et al., 2023). In geometry, models use geometric relations and theorems to formulate problems as executable sequences of symbolic reasoning steps (Lu et al., 2021; Zhang et al., 2023; Zhao et al., 2025). For mathematical and financial reasoning, models encode quantitative relationships in text and tables as equations or arithmetic programs (Amini et al., 2019; Chen et al., 2021b). In visual question answering, models translate questions into programs for querying scene objects, attributes, and relations (Yi et al., 2018; Eiter et al., 2023). For logical reasoning, models formalize premises and requirements as logical formulas or constraints for deduction and solving (Pan et al., 2023; Shi et al., 2025).

Self-training. Self-training allows language models to learn from their own outputs, reducing the need for manually annotated reasoning. A common approach turns verified solutions into supervised training targets (Yuan et al., 2023; Lin et al., 2025), while preference learning contrasts successful and failed outputs through objectives such as Direct Preference Optimization (DPO) (Rafailov et al., 2023; Wang et al., 2024). Both types of outputs can also be used to train verifiers for candidate selection (Hosseini et al., 2024). Beyond learning from existing predictions, STaR, $\mathrm{R}^3\mathrm{V}$, and ENVISIONS elicit further attempts through answer-conditioned reasoning or self-refinement, reusing successful new attempts as supervision in subsequent rounds (Zelikman et al., 2022; Cheng et al., 2025; Xu et al., 2025).

Symbolic reasoning for weak supervision. Complementary approaches derive supervision from the constraints that solutions must satisfy. In weakly supervised mathematical reasoning, answer consistency helps infer latent expressions and equations (Hong et al., 2021; Chatterjee et al., 2022; Liu et al., 2023). For example, LBF (Hong et al., 2021) repairs predicted arithmetic expression trees by propagating desired values backward and learns from the corrected expressions. Its repair mechanism is restricted to arithmetic-tree representations of math word problems. Abductive Learning (ABL) uses background knowledge to revise neural predictions and provide supervision for further learning (Zhou, 2019; Dai et al., 2019; Hu et al., 2025a;b).

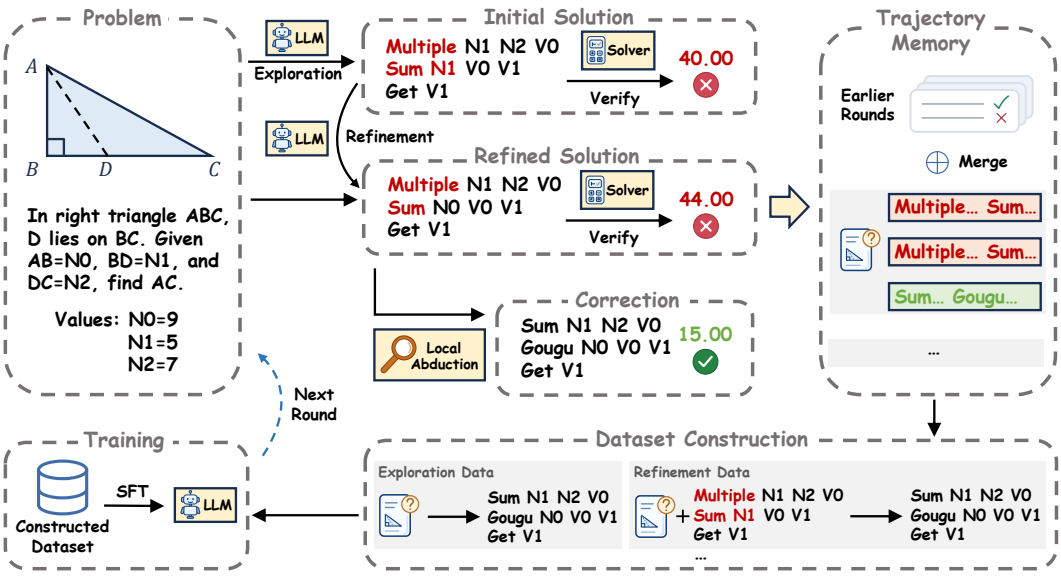


Figure 2: **Overview of LAST.** The model proposes and refines symbolic solutions, while local abduction recovers answer-consistent corrections from failed hypotheses. Current and historical trajectories are combined in memory to supervise solution generation and refinement.

Our focus is the supervision gap that remains when a model can propose useful structure but cannot yet generate a fully correct solution. LAST recovers training targets from these failed hypotheses rather than relying on successful regeneration alone. LAST formulates the recovery as local abduction in the broader neuro-symbolic setting of Section 2.1. The shared search procedure accommodates different symbolic representations without relying on a particular structure.

3 Methodology

The framework of LAST is shown in Figure 2. At each round, the model proposes and refines symbolic solutions, which the solver verifies (Section 3.1). Local abduction recovers answer-consistent corrections for unsolved problems (Section 3.2). Trajectory memory then supplies supervision for generation and refinement in subsequent rounds (Section 3.3). The trained model proposes the next round's hypotheses. A framework walkthrough of LAST and pseudocode are provided in Appendix A and Algorithm 1, respectively.

3.1 Hypothesis Generation

Exploration. At round t, the model proposes a symbolic solution $z_i \leftarrow p_{\theta_{t-1}}(\cdot \mid x_i)$ for each x_i. The solver S executes z_i to obtain $\hat{y}_i = S(z_i)$. We then compare $\hat{y}_i$ with the known answer y_i and assign binary feedback $b_i = \mathbb{I}[\hat{y}_i = y_i]$. Solutions with $b_i = 1$ provide an initial source of supervision.

Refinement. An initial solution also provides a reference for another reasoning attempt. The model is asked to reconsider the problem and refine this solution. Using this additional information, it generates a refined solution $\tilde{z}_i \leftarrow p_{\theta_{t-1}}(\cdot \mid x_i, z_i)$ for each x_i. The solver then executes $\tilde{z}_i$ to obtain $\tilde{y}_i = S(\tilde{z}_i)$, which is compared with y_i to give $\tilde{b}_i = \mathbb{I}[\tilde{y}_i = y_i]$.

3.2 Local Abduction

For problems where exploration and refinement both fail and no correct solution is available from previous rounds, LAST uses the structure of the failed predictions to recover correct solutions as additional supervision.

Modeling. For a failed symbolic solution z_i proposed by the model, we observe that parts of its structure can still be correct, and a correct solution may lie nearby. We now formalize the process of recovering a correct solution guided by this failed hypothesis.

Definition 1 (**Edit actions**). Let $\mathcal{A}_D$ denote the edit actions for symbolic solutions in domain D. Given an instruction position j, an action $e \in \mathcal{A}_D$ is a transformation that maps z to an edited solution $z' = e(z, j, x)$, such as replacing the operator or an operand of s_j, and has an edit cost $c(e)$.

Example 2. Consider the solution in Example 1, but with $s_2 = \mathrm{Sum}(N_0, V_0, V_1)$. An operator-replacement edit e_{op} changes it to $s_2' = \mathrm{Gougu}(N_0, V_0, V_1)$, leaving the other instructions unchanged. The resulting solution is $z' = e_{\mathrm{op}}(z, 2, x)$, with $c(e_{\mathrm{op}}) = 1$.

Definition 2 (**Edit distance**). For a given edit path $\pi = (e_1, \ldots, e_q)$ transforming z into z', written $z \xrightarrow{\pi} z'$, the *edit distance* is the accumulated cost along that path:

$$d_\pi(z, z') = \sum_{k=1}^{q} c(e_k). \tag{1}$$

Definition 3 (**Neighborhood**). The radius-r *neighborhood* of z contains the solutions reachable along an edit path whose edit distance is at most r:

$$\mathcal{N}_r(z) = \{z' : \exists\, \pi,\ z \xrightarrow{\pi} z',\ d_\pi(z, z') \le r\}. \tag{2}$$

Starting from the failed refinement $\tilde{z}_i$, abduction searches $\mathcal{N}_r(\tilde{z}_i)$ for solutions whose execution matches the known answer:

$$\mathcal{C}_i = \{z \in \mathcal{N}_r(\tilde{z}_i) : S(z) = y_i\}. \tag{3}$$

The answer constraint may not identify a unique solution: different structures can produce the same result, including ones that do not capture the intended reasoning. We therefore introduce a confidence score $M(z; x_i)$ to assess these candidate explanations, and select

$$z_i^\star \in \arg\max_{z \in \mathcal{C}_i} M(z; x_i). \tag{4}$$

Thus symbolic execution enforces answer consistency, while model confidence distinguishes the candidate explanations.

Solving. We solve the local abduction problem through breadth-first beam search, starting from the failed refinement $\tilde{z}_i$. Figure 5 in Appendix A illustrates this process, and Algorithm 2 provides the complete pseudocode. To expand a current solution $z = (s_1, \ldots, s_m)$, the search instantiates each edit type at different positions, drawing replacement operators from $\mathcal{F}_D$ and operands from $\mathcal{D}_{x_i}$. Let $\mathcal{S}_{e,j}(x_i, z)$ denote the set of instructions obtained by applying edit action $e \in \mathcal{A}_D$ to s_j with replacements allowed by the operator signature.

Example 3. For the solution z in Example 2, suppose $\mathcal{F}_D = \{\mathrm{Sum}, \mathrm{Gougu}, \mathrm{Multiple}\}$. Applying e_{op} at $j = 2$ gives

$$\mathcal{S}_{e_{\mathrm{op}},2}(x_i, z) = \{\mathrm{Gougu}(N_0, V_0, V_1), \mathrm{Multiple}(N_0, V_0, V_1)\}.$$

In general, replacing s_j with any $s' \in \mathcal{S}_{e,j}(x_i, z)$ preserves the rest of the solution and produces a neighbor:

$$z[j \leftarrow s'] = (s_1, \ldots, s_{j-1}, s', s_{j+1}, \ldots, s_m). \tag{5}$$

The neighbors generated by action e form $\mathcal{E}_e(z)$, and their union across actions gives the expanded neighbor set $\mathcal{E}(z)$:

$$\mathcal{E}_e(z) = \bigcup_{j=1}^{m} \{z[j \leftarrow s'] : s' \in \mathcal{S}_{e,j}(x_i, z)\}, \qquad \mathcal{E}(z) = \bigcup_{e \in \mathcal{A}_D} \mathcal{E}_e(z). \tag{6}$$

Following the fixed action order o, the solver checks whether $S(z') = y_i$ for each neighbor $z' \in \mathcal{E}(z)$ and collects successful solutions. Among candidates with $S(z') \neq y_i$, the search retains B candidates ranked by edit distance and randomly samples up to R additional candidates from the remainder. Each retained failure is expanded in the same way to form the next layer of neighbors. As the search proceeds, each edit updates the accumulated distance $g = d_\pi(\tilde{z}_i, z)$ to $g' = g + c(e)$. Only extensions with $g' \le r$ are explored, keeping the search within $\mathcal{N}_r(\tilde{z}_i)$ until the time limit.

A candidate's likelihood conditioned on x_i reflects how strongly the model supports its reasoning steps. We therefore instantiate M with the length-normalized log-likelihood:

$$\ell_t(z \mid x_i) = \frac{1}{L} \sum_{k=1}^{L} \log p_{\theta_{t-1}}(w_k \mid x_i, w_{<k}), \tag{7}$$

where $w_{1:L}$ are the tokens of z. The model evaluates these token probabilities by teacher forcing. The solution with the highest likelihood is selected: $z_i^\star \in \arg\max_{z \in \mathcal{C}_i} \ell_t(z \mid x_i)$.

3.3 Learning from Trajectories

Trajectory memory. A trajectory is represented as $T = (x_i, y_i, z, b, \ell, \sigma)$, where σ indicates its source: `Solve`, `Refine`, or `Repair`. For z_i, $\tilde{z}_i$, and $z_i^\star$, the corresponding trajectories are denoted by T_i, $\tilde{T}_i$, and $T_i^\star$, respectively. Among the per-problem abductive solutions, we retain the fraction ρ with the highest likelihoods. Let A_t denote the indices of the problems whose corrections are retained. The current trajectory collection is $\Delta_t = \bigcup_{i=1}^{n} \{T_i, \tilde{T}_i\} \cup \{T_i^\star : i \in A_t\}$.

The memory $\mathcal{M}_t$ stores the latest trajectory for each (i, σ, z) under the same generation input:

$$\mathcal{M}_t(i, \sigma, z) = \begin{cases} T, & \text{if } T = (x_i, y_i, z, b, \ell, \sigma) \in \Delta_t, \\ \mathcal{M}_{t-1}(i, \sigma, z), & \text{otherwise.} \end{cases} \tag{8}$$

Model training. For each problem with a correct trajectory in $\mathcal{M}_t$, its highest-likelihood correct solution z_i^+ supplies a generation pair (x_i, z_i^+). These pairs form the exploration set $\mathcal{T}_t^{\mathrm{E}}$. When incorrect trajectories are also available, the highest-likelihood failed solution z_i^- supplies a refinement pair $((x_i, z_i^-), z_i^+)$; all such pairs form $\mathcal{T}_t^{\mathrm{R}}$. The model is trained on their union $\mathcal{T}_t = \mathcal{T}_t^{\mathrm{E}} \cup \mathcal{T}_t^{\mathrm{R}}$ with the cross-entropy loss

$$\mathcal{L}_t(\theta) = -\frac{1}{|\mathcal{T}_t|} \sum_{(u,z) \in \mathcal{T}_t} \log p_\theta(z \mid u), \tag{9}$$

where u is x_i for generation and (x_i, z_i^-) for refinement. The log-probability sums over the tokens of the target solution.

The trained model generates the next round's hypotheses, allowing solutions discovered through abduction to inform subsequent exploration and refinement. At test time, only the problem input is provided; the model generates a symbolic solution and the solver returns its answer.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate our framework on geometry reasoning and numerical reasoning, each with its own symbolic solution language and solver. For geometry reasoning, we select three standard benchmarks covering a range of school-level geometry problems requiring multimodal reasoning: Geometry3K (Lu et al., 2021), PGPS9K (Zhang et al., 2023), and GeoQA (Chen et al., 2021a). For numerical reasoning, we evaluate on FinQA (Chen et al., 2021b), which involves financial text and tables, and MathQA (Amini et al., 2019), which contains multi-step math word problems. Dataset sizes, splits, and further details are provided in Appendix B.1.

Baselines. We compare LAST with teacher-distillation and self-training methods. Specifically, **Teacher Distillation (TD)** (Kim & Rush, 2016) fine-tunes a student on training problems paired with teacher-generated symbolic solutions, while **Filtered TD** retains only solutions whose execution matches the given answer. Among self-training baselines, **Iterative SFT** (Zelikman et al., 2022) trains on each round's answer-consistent predictions, and **STaR** (Zelikman et al., 2022) additionally provides answer hints for unsolved problems and adds verified solutions as supervision. **ENVISIONS** (Xu et al., 2025) accumulates explored and refined solutions across rounds to train both exploration and refinement. Note that LAST and the other self-training baselines generate one initial solution per problem, while ENVISIONS samples five, potentially finding more useful

Table 1: Test accuracy (%) of different methods across five geometry and numerical reasoning benchmarks using Qwen3-VL-Instruct models.

Methods	Geometry Reasoning			Numerical Reasoning		Avg.
	Geo3K	PGPS9K	GeoQA	FinQA	MathQA	
Qwen3-VL-2B-Instruct						
TD	25.30	19.20	9.95	36.50	29.53	24.09
Filtered TD	29.88	25.50	9.95	65.90	**31.73**	32.59
Iterative SFT	19.52	17.80	6.63	64.70	4.26	22.58
STaR	27.50	22.70	5.97	66.18	4.81	25.43
LAST (w/o Abduction)	25.30	21.90	7.16	66.18	5.08	25.12
LAST	**34.47**	**28.20**	**19.23**	**68.11**	16.35	**33.27**
ENVISIONS	35.82	30.60	14.06	69.49	6.59	31.31
LAST-MS	**42.95**	**35.20**	**28.51**	**70.23**	**14.56**	**38.29**
Qwen3-VL-8B-Instruct						
TD	25.98	22.60	13.00	39.17	33.52	26.85
Filtered TD	36.67	29.50	15.92	71.89	41.21	39.04
Iterative SFT	37.18	31.10	28.91	73.64	33.52	40.87
STaR	42.78	38.00	37.40	75.30	33.24	45.35
LAST (w/o Abduction)	45.50	37.90	39.26	75.58	43.96	48.44
LAST	**53.31**	**44.90**	**45.76**	**75.67**	**46.70**	**53.27**
ENVISIONS	53.14	46.90	48.01	75.21	51.24	54.90
LAST-MS	**57.72**	**50.70**	**51.72**	**76.50**	**53.43**	**58.02**

solutions at greater computational cost. To ensure a fair comparison, we implement **LAST-MS**, a Multi-Sample (MS) variant with the same exploration budget as ENVISIONS; details are provided in Appendix B.2. Additionally, **LAST (w/o Abduction)** removes only local abduction, keeping hypothesis generation, trajectory collection, and training unchanged.

Implementation details. We use Qwen3-VL-2B-Instruct and Qwen3-VL-8B-Instruct (Bai et al., 2025) as the initial models. Within each dataset, methods share the same 3-shot solution-generation prompt and refinement prompt; Appendix E illustrates their construction. ENVISIONS and LAST-MS sample five initial solutions at temperature 1 and refine each, while other self-training methods use greedy decoding for solution generation. We use GPT-4o (OpenAI, 2024) as the teacher for distillation. For TD and each self-training round, we fine-tune with LoRA (Hu et al., 2022) for 2 epochs, setting the rank to 64, $\alpha = 128$, and dropout to 0.05. We optimize with AdamW (Loshchilov & Hutter, 2019) at a learning rate of 5×10^{-5} and a batch size of 16. Following prior work (Zelikman et al., 2022; Xu et al., 2025), we train a new LoRA adapter from the same pretrained model in each round. For local abduction, we set $r = 2$, $B = 256$, and $R = 64$, with a 3-second search budget per problem, and retain the top $\rho = 25\%$ of per-problem corrections by model likelihood. We report test accuracy after 6 rounds of self-training. Unless otherwise stated, we use a fixed seed of 42. All experiments are conducted on a server equipped with AMD EPYC 9654 CPUs, with each training run using a single NVIDIA A100 80GB GPU. Edit action settings are provided in Appendix B.3.

4.2 Main Results

LAST consistently improves self-training across tasks. Across five benchmarks and both model sizes, LAST surpasses all self-training baselines that generate one initial solution per problem (Table 1). Its average improvements over Iterative SFT and STaR are 10.69–12.40 and 7.84–7.92 percentage points, respectively. The benefit extends to multi-sample exploration: LAST-MS outperforms ENVISIONS in all settings, with average gains of 6.98 points for 2B and 3.12 points for 8B. LAST also exceeds Filtered TD by 0.68 and 14.23 points on average for 2B and 8B, respectively, without requiring teacher-generated solutions. The iteration curves in Figure 3 reveal how these advantages develop over successive rounds. For example, LAST establishes a substantial lead on

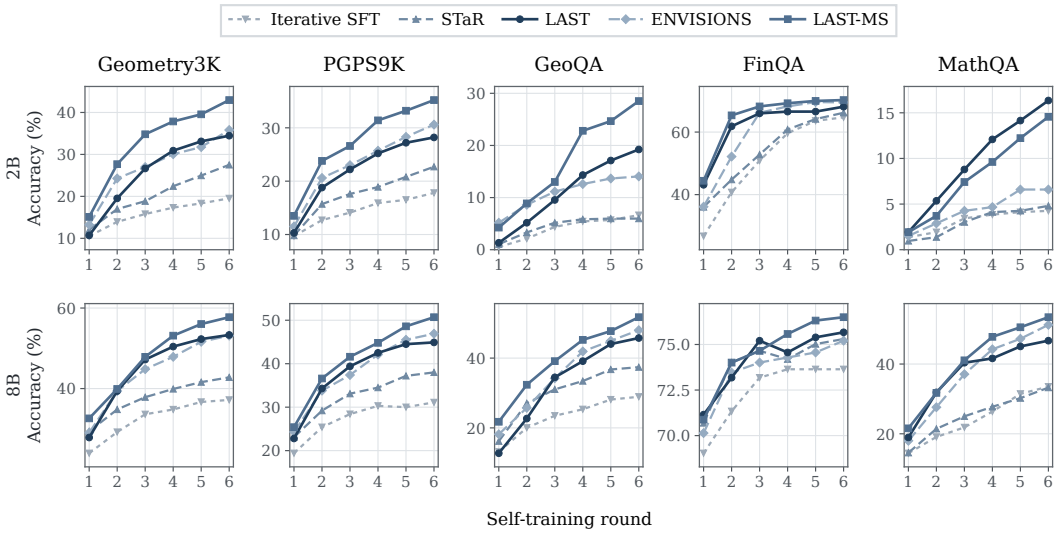


Figure 3: Test accuracy over six self-training rounds with Qwen3-VL-2B-Instruct (top) and Qwen3-VL-8B-Instruct (bottom).

GeoQA with the 2B model: STaR levels off near 6%, while LAST continues improving to 19.23%. Their gap grows from 0.27 points in the first round to 13.26 points in the sixth.

Ablation of LAST components. We ablate LAST's components on Geometry3K with Qwen3-VL-8B-Instruct to assess their individual and joint contributions, as illustrated in Figure 4(a). Removing neural refinement (w/o Refine), abduction (w/o Abd.), or both (w/o Both) reduces accuracy from 53.31% to 44.14%, 45.50%, and 38.20%, respectively. In w/o Ref. SFT, the model still refines its predictions, and successful refined solutions still supply targets for the exploration training set $\mathcal{T}_t^{\mathrm{E}}$. Only the refinement training set $\mathcal{T}_t^{\mathrm{R}}$ is removed, giving 47.37% accuracy. These results show that neural refinement and abduction provide complementary supervision, while learning to refine strengthens subsequent self-training.

4.3 Analysis

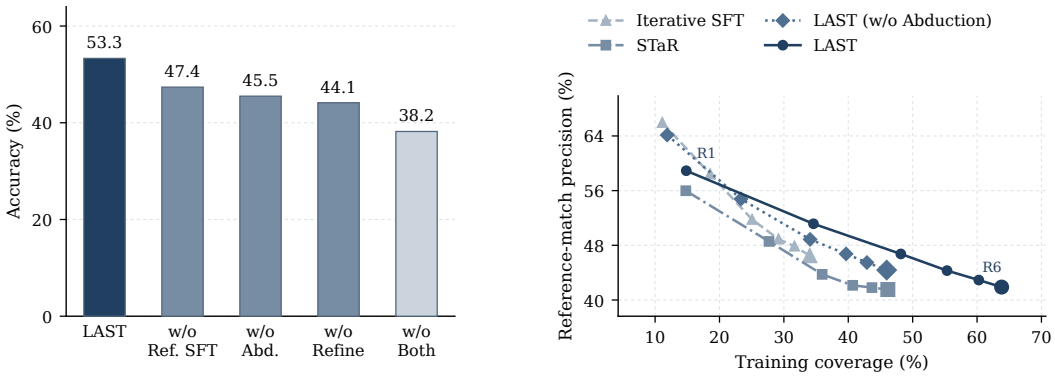
We further analyze the mechanisms underlying LAST, examining how abduction expands useful supervision and how failed hypotheses guide search. These analyses use Geometry3K with Qwen3-VL-8B-Instruct; additional experiments are presented in Appendix C.

Abduction expands useful supervision. We assess how abduction changes both the coverage and quality of supervision, comparing LAST with Iterative SFT, STaR, and LAST without abduction on our 8,432-problem Geometry3K training pool. The dataset's reference symbolic solutions provide the quality criterion for this offline analysis. Let U be the training problem set, and let $I_t \subseteq U$ denote the problems whose selected symbolic solutions provide supervision for solving the original inputs in round t. Let g_i indicate whether the selected solution for problem i exactly matches the reference solution. Coverage c_t, reference-match precision p_t, and their product are

$$c_t = \frac{|I_t|}{|U|}, \qquad p_t = \frac{\sum_{i \in I_t} g_i}{|I_t|}, \qquad c_t p_t = \frac{\sum_{i \in I_t} g_i}{|U|}. \tag{10}$$

Coverage measures the proportion of problems with an answer-consistent symbolic solution. Reference-match precision measures how often these solutions exactly match the annotated logical structure, providing a proxy for logical correctness. The product thus reflects the number of logically correct symbolic solutions recovered across the training set.

In the first round, exploration and refinement label 1,001 problems, and abduction repairs another 1,000, of which 250 are retained for training. As these additional labels accumulate, the model



(a) Module ablations

(b) Coverage and quality

Figure 4: Component and supervision analyses on Geometry3K with Qwen3-VL-8B-Instruct. (a) Round-six accuracy of module variants. (b) Coverage and reference-match precision of solution-generation targets; connected points follow rounds 1–6.

progressively learns to solve more problems directly. At the start of round six, its initial predictions correctly solve 4,772 of the 8,432 problems, versus 3,772 without abduction, before either refinement or abductive search. Figure 4(b) traces this growth from R1 to R6: LAST maintains broader coverage with comparable reference-match precision throughout training. By round six, coverage reaches 63.82%, compared with 34.01–46.15% for the baselines. Its precision is 41.89%, versus 41.53–46.48% for the baselines, and its coverage–precision product is 26.73%, versus 15.81–20.40%. Importantly, LAST achieves the highest product in every round, exceeding the strongest baseline at each round by 0.47–6.36 percentage points. Thus, local corrections expand useful supervision throughout self-training, supporting the model's growing ability to solve problems.

Failed hypotheses and local neighborhoods guide search. Does local abduction benefit from the model's failed hypotheses, or would a random starting solution work as well? We generate syntactically valid random solutions with the same number of instructions as the failed refinements and use them as alternative search starting points on the same first-round repair set. Here, coverage is the fraction of the shared repair set for which search finds an answer-consistent solution, while reference-match precision is measured among these successful problems. Failed hypotheses make search substantially more productive, raising coverage from 4.25% to 13.46%. Reference-match precision remains comparable: 16.80% for failed hypotheses versus 18.67% for random starts. This broader coverage thus yields more reference-matching supervision, increasing the coverage–precision product from 0.79% to 2.26%. This benefit also carries through to learning: replacing failed hypotheses with random starting solutions throughout a separate self-training run reduces round-six test accuracy from 53.31% to 50.93%. Together, these results show that failed hypotheses guide search toward useful supervision that improves subsequent learning.

5 Conclusion

We presented LAST for neuro-symbolic self-training, motivated by the observation that failed solutions can retain useful structure even when the model cannot solve the problem on its own. That structure provides a starting point for local abduction, which searches the failed hypothesis's neighborhood for answer-consistent corrections rather than requiring another complete solution from the model. LAST selects these corrections using model likelihood and stores them alongside successful neural predictions in trajectory memory. These corrections enable the model to learn from previously unsolved problems in subsequent rounds. Across five benchmarks and two model sizes, this additional supervision yields substantial performance gains over self-training baselines. Further analyses show that coverage expands while solution quality is preserved. These results point to a broader role for symbolic reasoning in self-training: not only verifying solutions a model can already produce, but also helping construct training targets it cannot yet generate on its own.

AI USE STATEMENT

We used large language models (LLMs) for translation, language polishing, and assistance in implementing parts of the method and experiments. All LLM-assisted content and code were reviewed and verified by the authors, who take full responsibility for this work.

REPRODUCIBILITY STATEMENT

We describe the method in Section 3 and provide pseudocode in Appendix D. Experimental settings and dataset details are documented in Section 4.1 and Appendix B, with prompt examples in Appendix E. We will publicly release our code and data upon acceptance.

REFERENCES

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 2357–2367. Association for Computational Linguistics, 2019. doi: 10.18653/V1/N19-1245. URL https://doi.org/10.18653/v1/n19-1245.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report, 2025. URL https://arxiv.org/abs/2511.21631.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Oishik Chatterjee, Isha Pandey, Aashish Waikar, Vishwajeet Kumar, and Ganesh Ramakrishnan. WARM: A weakly (+semi) supervised math word problem solver. In Nicoletta Calzolari, Chu-Ren Huang, Hansaem Kim, James Pustejovsky, Leo Wanner, Key-Sun Choi, Pum-Mo Ryu, Hsin-Hsi Chen, Lucia Donatelli, Heng Ji, Sadao Kurohashi, Patrizia Paggio, Nianwen Xue, Seokhwan Kim, Younggyun Hahm, Zhong He, Tony Kyungil Lee, Enrico Santus, Francis Bond, and Seung-Hoon Na (eds.), *Proceedings of the 29th International Conference on Computational Linguistics, COLING 2022, Gyeongju, Republic of Korea, October 12-17, 2022*, pp. 4753–4764. International Committee on Computational Linguistics, 2022. URL https://aclanthology.org/2022.coling-1.421.

Jiaqi Chen, Jianheng Tang, Jinghui Qin, Xiaodan Liang, Lingbo Liu, Eric P. Xing, and Liang Lin. Geoqa: A geometric question answering benchmark towards multimodal numerical reasoning. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Findings of*

the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021, volume ACL-IJCNLP 2021 of *Findings of ACL*, pp. 513–523. Association for Computational Linguistics, 2021a. doi: 10.18653/V1/2021.FINDINGS-ACL.46. URL https://doi.org/10.18653/v1/2021.findings-acl.46.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Trans. Mach. Learn. Res.*, 2023, 2023. URL https://openreview.net/forum?id=YfZ4ZPt8zd.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao 'Kenneth' Huang, Bryan R. Routledge, and William Yang Wang. Finqa: A dataset of numerical reasoning over financial data. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pp. 3697–3711. Association for Computational Linguistics, 2021b. doi: 10.18653/V1/2021.EMNLP-MAIN.300. URL https://doi.org/10.18653/v1/2021.emnlp-main.300.

Kanzhi Cheng, Yantao Li, Fangzhi Xu, Jianbing Zhang, Hao Zhou, and Yang Liu. Vision-language models can self-improve reasoning via reflection. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pp. 8876–8892. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.NAACL-LONG.447. URL https://doi.org/10.18653/v1/2025.naacl-long.447.

Wang-Zhou Dai, Qiu-Ling Xu, Yang Yu, and Zhi-Hua Zhou. Bridging machine learning and logical reasoning by abductive learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 2811–2822, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/9c19a2aa1d84e04b0bd4bc888792bd1e-Abstract.html.

Thomas Eiter, Tobias Geibinger, Nelson Higuera, and Johannes Oetsch. A logic-based approach to contrastive explainability for neurosymbolic visual question answering. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pp. 3668–3676. ijcai.org, 2023. doi: 10.24963/IJCAI.2023/408. URL https://doi.org/10.24963/ijcai.2023/408.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: program-aided language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Yining Hong, Qing Li, Daniel Ciao, Siyuan Huang, and Song-Chun Zhu. Learning by fixing: Solving math word problems with weak supervision. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 4959–4967. AAAI Press, 2021. doi: 10.1609/AAAI.V35I6.16629. URL https://doi.org/10.1609/aaai.v35i6.16629.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. V-star: Training verifiers for self-taught reasoners, 2024. URL https://arxiv.org/abs/2402.06457.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Wen-Chao Hu, Wang-Zhou Dai, Yuan Jiang, and Zhi-Hua Zhou. Efficient rectification of neuro-symbolic reasoning inconsistencies by abductive reflection. In Toby Walsh, Julie Shah, and Zico Kolter (eds.), *Thirty-Ninth AAAI Conference on Artificial Intelligence, Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence, Fifteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2025, Philadelphia, PA, USA, February 25 - March 4, 2025*, pp. 17333–17341. AAAI Press, 2025a. doi: 10.1609/AAAI.V39I16.33905. URL https://doi.org/10.1609/aaai.v39i16.33905.

Wen-Chao Hu, Qi-Jie Li, Lin-Han Jia, Cunjing Ge, Yufeng Li, Yuan Jiang, and Zhi-Hua Zhou. Curriculum abductive learning. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025b. URL http://papers.nips.cc/paper_files/paper/2025/hash/fed1ea8dcc2a13f3835cc854e8c8294c-Abstract-Conference.html.

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In Jian Su, Xavier Carreras, and Kevin Duh (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pp. 1317–1327. The Association for Computational Linguistics, 2016. doi: 10.18653/V1/D16-1139. URL https://doi.org/10.18653/v1/d16-1139.

Haohan Lin, Zhiqing Sun, Sean Welleck, and Yiming Yang. Lean-star: Learning to interleave thinking and proving. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=SOWZ59UyNc.

Qianying Liu, Wenyu Guan, Jianhao Shen, Fei Cheng, and Sadao Kurohashi. Comsearch: Equation searching with combinatorial strategy for solving math word problems with weak supervision. In Andreas Vlachos and Isabelle Augenstein (eds.), *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2023, Dubrovnik, Croatia, May 2-6, 2023*, pp. 2541–2554. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EACL-MAIN.186. URL https://doi.org/10.18653/v1/2023.eacl-main.186.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. Inter-gps: Interpretable geometry problem solving with formal language and symbolic reasoning. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pp. 6774–6786. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.ACL-LONG.528. URL https://doi.org/10.18653/v1/2021.acl-long.528.

OpenAI. Hello GPT-4o. https://openai.com/index/hello-gpt-4o/, 2024.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, volume EMNLP 2023 of *Findings of ACL*, pp. 3806–3824. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.248. URL https://doi.org/10.18653/v1/2023.findings-emnlp.248.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine

(eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html.

Weichun Shi, Minghao Liu, Wanting Zhang, Langchen Shi, Fuqi Jia, Feifei Ma, and Jian Zhang. Constraintllm: A neuro-symbolic framework for industrial-level constraint programming. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 15999–16019. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.809. URL https://doi.org/10.18653/v1/2025.emnlp-main.809.

Tianduo Wang, Shichen Li, and Wei Lu. Self-training with direct preference optimization improves chain-of-thought reasoning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 11917–11928. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.643. URL https://doi.org/10.18653/v1/2024.acl-long.643.

Fangzhi Xu, Qiushi Sun, Kanzhi Cheng, Jun Liu, Yu Qiao, and Zhiyong Wu. Interactive evolution: A neural-symbolic self-training framework for large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 12975–12993. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.ACL-LONG.635. URL https://doi.org/10.18653/v1/2025.acl-long.635.

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. Neural-symbolic VQA: disentangling reasoning from vision and language understanding. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 1039–1050, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/5e388103a391daabe3de1d76a6739ccd-Abstract.html.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models, 2023. URL https://arxiv.org/abs/2308.01825.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Ming-Liang Zhang, Fei Yin, and Cheng-Lin Liu. A multi-modal neural geometric solver with textual clauses parsed from diagram. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pp. 3374–3382. ijcai.org, 2023. doi: 10.24963/IJCAI.2023/376. URL https://doi.org/10.24963/ijcai.2023/376.

Junbo Zhao, Ting Zhang, Jiayu Sun, Mi Tian, and Hua Huang. Pi-gps: Enhancing geometry problem solving by unleashing the power of diagrammatic information. In *IEEE/CVF International Conference on Computer Vision, ICCV 2025, Honolulu, HI, USA, October 19-25, 2025*, pp. 1526–1536. IEEE, 2025. doi: 10.1109/ICCV51701.2025.00150. URL https://doi.org/10.1109/ICCV51701.2025.00150.

Zhi-Hua Zhou. Abductive learning: towards bridging machine learning and logical reasoning. *Sci. China Inf. Sci.*, 62(7):76101:1–76101:3, 2019. doi: 10.1007/S11432-018-9801-4. URL https://doi.org/10.1007/s11432-018-9801-4.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, Zhangwei Gao, Erfei Cui, Xuehui Wang, Yue Cao, Yangzhou Liu, Xingguang Wei, Hongjie Zhang, Haomin Wang, Weiye Xu, Hao Li, Jiahao Wang, Nianchen Deng, Songze Li, Yinan He, Tan Jiang, Jiapeng Luo, Yi Wang, Conghui He, Botian Shi, Xingcheng Zhang, Wenqi Shao, Junjun He, Yingtong Xiong, Wenwen Qu, Peng Sun, Penglong Jiao, Han Lv, Lijun Wu, Kaipeng Zhang, Huipeng Deng, Jiaye Ge, Kai Chen, Limin Wang, Min Dou, Lewei Lu, Xizhou Zhu, Tong Lu, Dahua Lin, Yu Qiao, Jifeng Dai, and Wenhai Wang. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models, 2025. URL https://arxiv.org/abs/2504.10479.

A A WALKTHROUGH OF LAST

We follow one round of LAST on the problem in Figure 2. The input x contains the question, with $N_0 = 9$, $N_1 = 5$, and $N_2 = 7$; the known answer is $y = 15$.

Exploration and refinement. The model first generates z and then produces $\tilde{z}$ by reconsidering the problem with z as a reference. The solver returns 40 and 44, respectively, so both solutions fail verification:

Stage	Symbolic solution	Output
Initial z	`Multiple N1 N2 V0 Sum N1 V0 V1 Get V1`	40
Refined $\tilde{z}$	`Multiple N1 N2 V0 Sum N0 V0 V1 Get V1`	44
Corrected $z^\star$	`Sum N1 N2 V0 Gougu N0 V0 V1 Get V1`	15

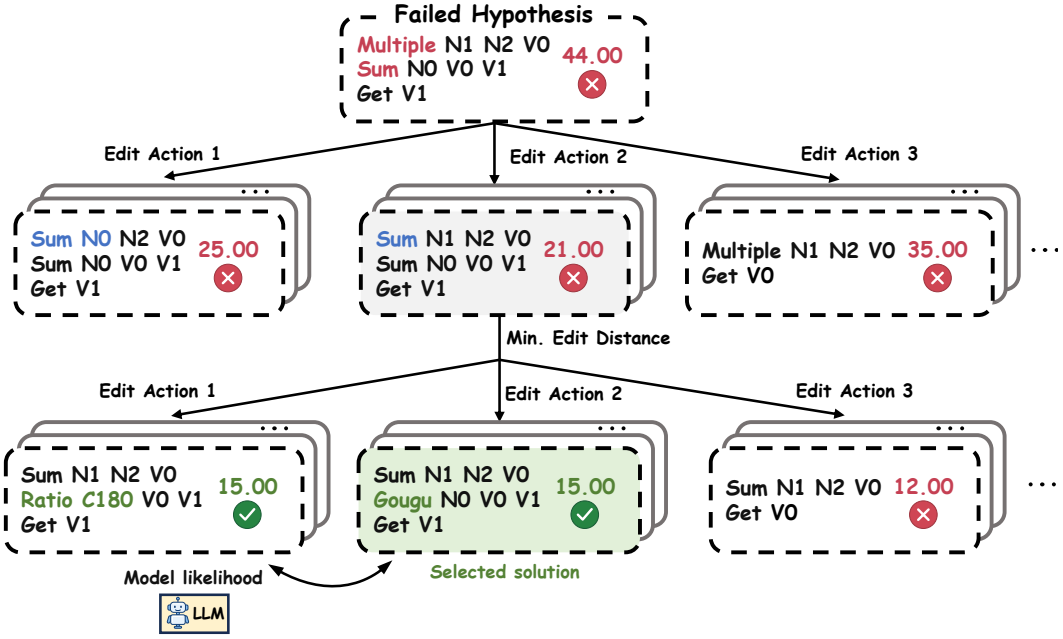


Figure 5: **An example of local abduction.** The same edit actions are applied at each layer. The failed candidate with the smallest edit distance is expanded; answer-consistent solutions are ranked by model likelihood.

Local abduction. Suppose no correct solution for this problem is available in memory. LAST therefore searches near $\tilde{z}$. For the search illustrated in Figure 5, set $B = 1$ and $R = 0$. Let e_1, e_2, e_3 denote Edit Actions 1–3. The joint edit e_1 replaces an instruction's operator and one operand, with $c(e_1) = 2$. The single edit e_2 replaces either an operator or one operand, with $c(e_2) = 1$. The deletion edit e_3 removes a computation and redirects `Get` to the last remaining intermediate, with cost $c(e_3) = 5$.

In the first expansion, e_1 changes the first instruction to `Sum N0 N2 V0`, whereas e_2 changes only `Multiple` to `Sum`. The third action deletes the second computation and changes `Get V1` to `Get V0`. These candidates return 25, 21, and 35, with edit distances 2, 1, and 5 along the displayed paths. All three fail verification. The middle candidate $z_2 = e_2(\tilde{z}, 1, x)$ has the smallest edit distance and ranks first under the beam score, so it is retained for the next expansion.

Applying the same three actions at the second instruction of z_2 gives the second layer in Figure 5. Each distance includes the first edit, giving 3, 2, and 6, respectively. The first two candidates both match the answer, but only the second captures the intended reasoning. The `Ratio` solution computes $180/(5+7) = 15$, ignoring AB and using an unjustified constant. The `Gougu` solution instead applies the Pythagorean relation, $\sqrt{9^2 + (5+7)^2} = 15$. Suppose their model scores are -0.90 and -0.50, respectively. These are the length-normalized log-likelihoods ℓ_t defined in (7); a higher value indicates greater model confidence. The `Gougu` solution is therefore selected as $z^\star$.

Trajectory memory. Suppose an earlier round stored z under the `Solve` source with recorded $\ell = -0.30$, and its current score is $\ell = -0.40$. Since the same solution reappears under the same source and input, the current trajectory replaces the earlier one rather than adding a duplicate. The refined solution $\tilde{z}$ with score -0.70 and the retained correction $z^\star$ with score -0.50 are new. They are added under the `Refine` and `Repair` sources. Memory now contains two failed trajectories and one successful correction for this problem.

Training data. The corrected solution $z^\star$ provides supervision for both solution generation and refinement. Among the failed solutions, z has the higher recorded likelihood, so it also serves as the input to refinement. This problem contributes two pairs:

$$(x, z^\star) \in \mathcal{T}_t^{\mathrm{E}}, \qquad ((x, z), z^\star) \in \mathcal{T}_t^{\mathrm{R}}.$$

The first teaches the model to generate the corrected solution from the problem alone; the second teaches it to revise a failed solution into the same target. After training on these pairs alongside those from other problems, the updated model begins the next round of exploration and refinement.

B Experimental Setup and Implementation Details

B.1 Datasets

Table 2 lists the training and test sizes. All methods use the same data splits within each dataset.

For Geometry3K, we use the split provided in the PGPS9K release (Zhang et al., 2023), whose training pool combines Geometry3K with additional textbook problems and whose test set contains 589 Geometry3K problems. PGPS9K uses the released split of 8,021 training and 1,000 test problems, while GeoQA uses 3,499 training and 754 test problems. The geometry inputs contain the diagram, question, and available quantities.

Table 2: Dataset sizes used in the experiments.

Dataset	Train	Test
Geometry3K	8,432	589
PGPS9K	8,021	1,000
GeoQA	3,499	754
FinQA	5,926	1,085
MathQA	7,394	728

For FinQA and MathQA, we preserve the original split assignments. For FinQA, we select numerical reasoning problems and use the full financial passage and a text representation of the table as input.

For MathQA, we focus on more challenging multi-step reasoning by selecting problems whose reference programs contain at least four operations contributing to the final result. Reference programs must reproduce the annotated numerical answer, and duplicate questions are removed. The model receives the question text. The development splits of both datasets remain separate from training and test.

B.2 Implementation details of LAST-MS

LAST-MS follows the ENVISIONS baseline (Xu et al., 2025) in exploration, refinement, trajectory memory, and training, using the same experimental settings. Local abduction is applied when all current initial and refined solutions for a problem are incorrect and no correct solution is available in its trajectory memory. The highest-likelihood failed refinement is selected as the search starting point. Search and candidate selection follow LAST, and retained corrections are added to trajectory memory for training.

B.3 Edit action settings

Program syntax. Geometry tasks use PGPS equation programs (Zhang et al., 2023), such as `Sum N1 N2 V0 Gougu N0 V0 V1 Get V1`. Here, `N` symbols refer to input quantities, `V` symbols denote intermediate variables, and `Get` specifies the target. FinQA (Chen et al., 2021b) and MathQA (Amini et al., 2019) retain their native syntax, with operators restricted to `add`, `subtract`, `multiply`, and `divide`. For example, FinQA writes `add(5,7), multiply(#0,const_2)`, whereas MathQA writes `add(n0,n1)|multiply(#0,const_2)|`. FinQA uses numerical literals from the input, while MathQA indexes them with `n0`, `n1`, and so on; `#0` refers to the result of the first instruction in both languages.

Edit actions. Tables 3 and 4 list the edit actions and their fixed search order o for the two task families. Replacement operators and operands are drawn from $\mathcal{F}_D$ and $\mathcal{D}_x$, respectively. For example, suppose a FinQA problem provides the quantities 5, 7, and 10 in its passage and table. Then $\mathcal{F}_D = \{\texttt{add}, \texttt{subtract}, \texttt{multiply}, \texttt{divide}\}$ and $\mathcal{D}_x = \{5, 7, 10, \#0, \texttt{const_0}, \texttt{const_1}, \texttt{const_2}, \texttt{const_100}, \ldots\}$. Here, `#0` refers to the first instruction's result and is available to subsequent instructions; 10 remains an available input quantity even though it does not appear in the example program. The search first tries operator replacements in `add(5,7)` and then in `multiply(#0,const_2)`, before moving on to operand replacements.

Table 3: Edit actions for geometry reasoning, in search order.

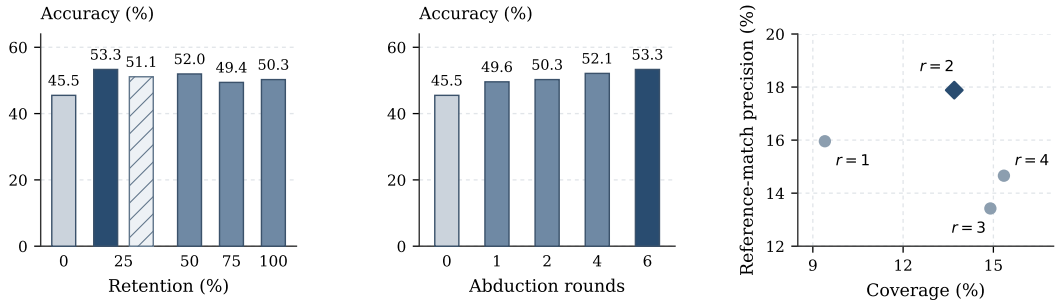
Edit action	Modification and example
Output-reference replacement	Change the target of `Get`. `Get V1` → `Get V0`
Operator replacement	Replace the operator and adapt its operands. `Sum N0 V0 V1` → `Gougu N0 V0 V1`
Operand replacement	Replace one operand. `Sum N0 V0 V1` → `Sum N1 V0 V1`

Table 4: Edit actions for numerical reasoning, in search order. Examples use FinQA syntax.

Edit action	Modification and example
Operator replacement	Replace the arithmetic operator. `subtract(5,7)` → `add(5,7)`
Operand replacement	Replace one operand. `subtract(5,7)` → `subtract(5,const_2)`
Operand swap	Exchange the two operands. `subtract(5,7)` → `subtract(7,5)`

C Additional Experiments

Reachability of reference solutions. Before self-training, Qwen3-VL-8B-Instruct predicts symbolic solutions for 5,926 FinQA training problems. From these predictions, we examine 2,180 incorrect but syntactically valid solutions. Using the same numerical-reasoning edit actions as in Appendix B.3, we compute the minimum number of edits needed to transform each failed solution into its corresponding reference. Figure 1(b) summarizes the results: 337 (15.5%), 574 (26.3%), 658 (30.2%), and 676 (31.0%) of these failures can be transformed into correct reference solutions with at most one, two, three, and four edits, respectively.



(a) Correction selection (b) Abduction duration (c) Neighborhood radius

Figure 6: Selection, duration, and neighborhood analyses on Geometry3K with Qwen3-VL-8B-Instruct. (a) Retention fraction; the hatched bar randomly retains 25% after per-problem selection. (b) Abduction is enabled only in the first h rounds. (c) Coverage and reference-match precision of per-problem abductive solutions on 2,000 fixed pre-training failures.

Analysis of likelihood-based selection. To examine the role of likelihood at the two selection stages, we randomize each stage separately. The within-problem control uniformly samples one answer-consistent candidate per problem instead of choosing the highest-likelihood candidate, then retains the top 25% by likelihood as usual. Random-25 instead keeps the highest-likelihood candidate for each problem but randomly retains 25% of these candidates rather than ranking them by likelihood. The two controls achieve 51.95% and 51.10% accuracy, respectively, both below LAST's 53.31%. Keeping likelihood-based selection and increasing the retention fraction ρ from 25% to 50%, 75%, and 100% yields 51.95%, 49.41%, and 50.25%, respectively (Figure 6(a)).

To better understand these gains, we examine how likelihood-based selection affects the quality of the supervision. In the first round on Geometry3K with Qwen3-VL-8B-Instruct, 189 of the 1,000 highest-likelihood corrections selected per problem match the reference solution (18.9%). Retaining the top 25% across problems increases this proportion to 95 out of 250 (38.0%). On these same 250 problems, uniformly sampling an answer-consistent candidate per problem gives an expected match rate of 26.43%, compared with 38.0% when choosing the highest-likelihood candidate.

Together, these results suggest that likelihood helps distinguish useful supervision among answer-consistent corrections. By favoring solutions that more often match the reference, likelihood-based selection provides more effective training targets than simply retaining more candidates.

Analysis of abduction duration. We vary the number of rounds with abduction to determine whether it only helps bootstrap learning or continues to provide useful supervision. All models complete six training rounds, with abduction enabled only in the first one, two, or four (Figure 6(b)). Accuracy reaches 49.58%, 50.25%, and 52.12%, respectively, compared with 53.31% when abduction is used throughout. The gains from later corrections show that its contribution extends beyond bootstrapping.

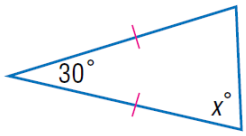
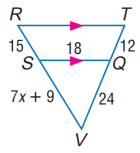
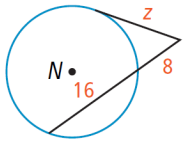
Neighborhood radius. To examine the role of locality, we randomly sample 2,000 problems from the same repair set used in the search-initialization comparison in Section 4.3. Starting from the same failed refinements, we vary $r \in \{1, 2, 3, 4\}$ while keeping other search settings fixed. Each successful search contributes its highest-likelihood abductive solution $z_i^\star$. We evaluate all these

corrections, rather than only the top ρ fraction retained for training, with coverage measured over all 2,000 sampled problems. Increasing r from 1 to 2 improves both coverage (9.40% to 13.70%) and precision (15.96% to 17.88%). Larger neighborhoods raise coverage further but reduce precision (Figure 6(c)): the products $c_t p_t$ are 1.50%, 2.45%, 2.00%, and 2.25% for $r = 1, 2, 3, 4$, respectively. Under the same search budget, allowing more distant corrections therefore does not continually increase the recovery of reference solutions. A useful neighborhood permits meaningful changes while preserving the guidance provided by the failed hypothesis.

Semantic quality of repairs. We examine whether the training targets recovered by LAST encode valid reasoning beyond producing the correct answer. For the Geometry3K run with Qwen3-VL-8B-Instruct and seed 42, we randomly sample 100 of the 1,251 corrections newly retained for training after likelihood-based selection in rounds 1–6 using proportional stratified sampling by round. The sampled solutions are manually reviewed against the problem text and diagram, variable meanings, and reference solution. The review accepts valid algebraic reformulations but classifies a solution as spurious if it reaches the annotated answer through an unjustified relation or by solving for the wrong quantity.

Among the sampled corrections, 28% match the reference solution, 55% are valid alternatives that do not match it, and 17% are spurious. Table 5 illustrates the three categories. These results indicate that local abduction provides predominantly valid reasoning targets for self-training, including correct alternative solutions.

Table 5: Examples from the retained-repair audit. The repaired solutions all produce the annotated answer, but their reasoning differs in validity.

	Reference match	Valid alternative	Spurious solution
Problem	 An isosceles triangle has vertex angle `N1` and base angle `N0`. Find x. `N0` $= x$, `N1` $= 30$.	 Given $RT \parallel SQ$, $RS =$ `N0`, $TQ =$ `N1`, $SV =$ `N2`, $SQ =$ `N3`, and $VQ =$ `N4`, find x. `N0` $= 15$, `N1` $= 12$, `N2` $= 7x + 9$, `N3` $= 18$, `N4` $= 24$.	 The tangent has length `N0`; the secant's external and internal segments have lengths `N1` and `N2`. Find z. `N0` $= z$, `N1` $= 8$, `N2` $= 16$.
Solution	`Iso_Tri_Ang N1 N0` `Get x`	`Proportion N0 N2 N1 N4` `Get x`	`Gougu N0 N1 N2` `Get N0`
Explanation	The equal base angles satisfy $30 + 2x = 180$, giving $x = 75$. The solution matches the reference.	Parallel lines give $\frac{15}{7x+9} = \frac{12}{24}$. The reference uses the reciprocal proportion; both yield $x = 3$.	Uses $z^2 = 16^2 - 8^2$ instead of the valid tangent–secant relation $z^2 = 8(8 + 16)$. Both yield $z = \sqrt{192}$.

More updates alone do not explain the gains. Abduction adds training examples and therefore also increases the number of updates. To separate the benefit of more updates from that of new supervision, we repeat the no-abduction variant's own training data to match LAST's update count on Geometry3K with Qwen3-VL-8B-Instruct and seed 42. This raises accuracy only from 45.50% to 46.18%, still 7.13 points below LAST. The gains thus come not simply from training longer on existing solutions, but from the additional supervision discovered through abduction.

Computational efficiency. Local abduction introduces additional computation for search and candidate scoring. We examine whether the resulting learning gains justify this additional cost. We analyze the six-round logs on Geometry3K with Qwen3-VL-8B-Instruct and seed 42, using one A100 80GB GPU per run on the same server. Cumulative wall-clock time includes generation, verification, search, likelihood scoring, data processing, and training. Figure 7(a) shows that LAST reaches 47.20% accuracy in 8.97 hours, exceeding the no-abduction variant's 45.50% after 16.29 hours. LAST-MS reaches 53.14% in 15.89 hours, whereas ENVISIONS takes 25.70 hours to reach the same accuracy, a 38.2% reduction in time. The breakdown in Figure 7(b) shows that LAST's search and scoring take 1.12 and 0.44 hours across six rounds, together accounting for 6.7% of its runtime. Most of the extra time relative to the no-abduction variant comes from training on the larger dataset, which accounts for 77.0% of the increase. The additional supervision thus improves the observed accuracy–time trade-off even when search and scoring are included.

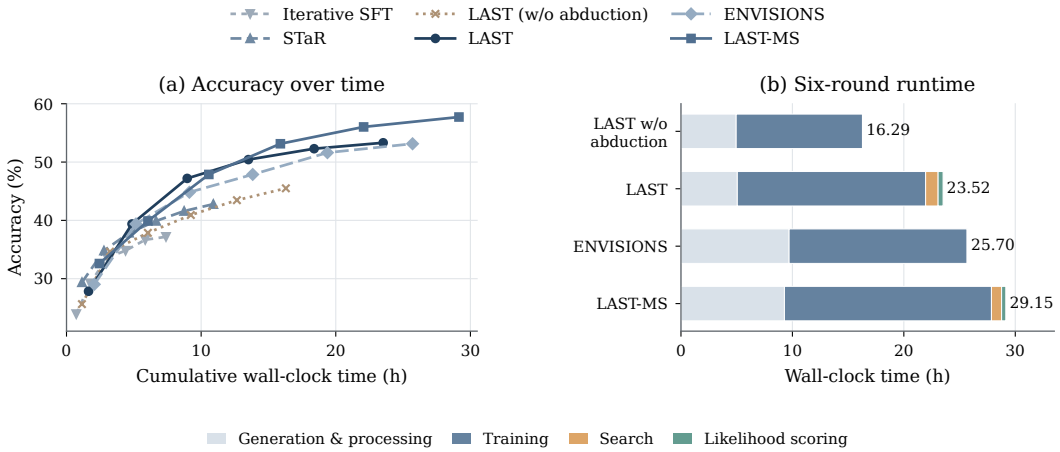


Figure 7: Computational efficiency on Geometry3K with Qwen3-VL-8B-Instruct. (a) Test accuracy against cumulative wall-clock time; points correspond to the end of each training round. (b) Runtime breakdown over six rounds. Generation and processing include verification and data preparation.

Table 6: Seed and model-family comparisons on Geometry3K. Accuracy is in percent.

Model / Seed	Iterative SFT	STaR	LAST (w/o Abduction)	LAST
Qwen3-VL-8B-Instruct / 42	37.18	42.78	45.50	**53.31**
Qwen3-VL-8B-Instruct / 43	38.37	44.14	46.01	**51.44**
Qwen3-VL-8B-Instruct / 44	37.01	40.24	44.65	**51.10**
Mean $\pm$ std.	37.52 ± 0.74	42.39 ± 1.98	45.39 ± 0.69	51.95 ± 1.19
InternVL3-8B / 42	34.30	37.86	38.71	**46.52**

The benefits persist across seeds and model families. A useful source of supervision should remain beneficial when the training seed or backbone changes. Across seeds 42, 43, and 44, LAST improves over self-training baselines in every run, averaging $51.95 \pm 1.19\%$ versus $45.39 \pm 0.69\%$ (Table 6). Replacing Qwen3-VL-8B-Instruct with InternVL3-8B (Zhu et al., 2025) gives the same pattern: abduction raises accuracy from 38.71% to 46.52%. These results extend its benefit beyond a single training seed or model family.

D ALGORITHMS

Algorithm 1 gives the pseudocode for LAST. Algorithm 2 gives the pseudocode for local abduction. Each state (z, g) stores the edit distance $g = d_\pi(\tilde{z}_i, z)$ along its path from $\tilde{z}_i$; applying an edit e gives $g' = g + c(e)$.

Algorithm 1 Local Abductive Self-Training (LAST)

Require: Training set $\mathcal{D}$, base model p_{θ_0}, rounds T, retention fraction ρ
Ensure: Trained model p_{θ_T}

$\mathcal{M}_0 \leftarrow \varnothing$; $\mathcal{T}_0^{\mathrm{E}} \leftarrow \varnothing$
for $t = 1, \ldots, T$ **do**
  $\Delta_t \leftarrow \varnothing$
  for each $(x_i, y_i) \in \mathcal{D}$ **do**
    $z_i \leftarrow p_{\theta_{t-1}}(\cdot \mid x_i)$; $\tilde{z}_i \leftarrow p_{\theta_{t-1}}(\cdot \mid x_i, z_i)$
    Add `Solve` and `Refine` trajectories with feedback $b_i, \tilde{b}_i$ and likelihoods to Δ_t as in Section 3.3
    if $b_i = \tilde{b}_i = 0$ and x_i has no target in $\mathcal{T}_{t-1}^{\mathrm{E}}$ **then**
      $z_i^\star \leftarrow \textsc{LocalAbduction}(x_i, y_i, \tilde{z}_i, p_{\theta_{t-1}})$
    end if
  end for
  Collect the problems with an abductive solution into J_t
  $A_t \leftarrow$ the top $\lceil \rho |J_t| \rceil$ problems in J_t ranked by $\ell_t(z_i^\star \mid x_i)$
  Add `Repair` trajectories for $\{z_i^\star : i \in A_t\}$ to Δ_t
  Update the trajectories in $\mathcal{M}_t$ with Δ_t using (8)
  Construct $\mathcal{T}_t^{\mathrm{E}}$ and $\mathcal{T}_t^{\mathrm{R}}$ from $\mathcal{M}_t$ as in Section 3.3
  Fit p_{θ_t} from the base model on $\mathcal{T}_t^{\mathrm{E}} \cup \mathcal{T}_t^{\mathrm{R}}$ with loss $\mathcal{L}_t$
end for
return p_{θ_T}

Algorithm 2 Local abduction

Require: (x_i, y_i), failed refinement $\tilde{z}_i$, model $p_{\theta_{t-1}}$; edit actions $\mathcal{A}_D$, action order o, settings r, B, R and a time limit
Ensure: A selected solution $z_i^\star$

$F \leftarrow \{(\tilde{z}_i, 0)\}$; $C \leftarrow \varnothing$
while $F \neq \varnothing$ and time remains **do**
  $G \leftarrow \varnothing$
  for each $(z, g) \in F$ **do**
    for each $e \in \mathcal{A}_D$ with $g + c(e) \leq r$, in order o **do**
      $g' \leftarrow g + c(e)$
      Instantiate $\mathcal{S}_{e,j}(x_i, z)$ at each instruction position j
      $\mathcal{E}_e(z) \leftarrow \bigcup_{j=1}^{m} \{z[j \leftarrow s'] : s' \in \mathcal{S}_{e,j}(x_i, z)\}$
      for each $z' \in \mathcal{E}_e(z)$, while time remains **do**
        if $S(z') = y_i$ **then**
          $C \leftarrow C \cup \{z'\}$
        else
          $G \leftarrow G \cup \{(z', g')\}$
        end if
      end for
    end for
  end for
  $F \leftarrow$ the B states in G with the smallest edit distances g
  Add up to R randomly sampled states from $G \setminus F$ to F
end while
Evaluate $\ell_t(z \mid x_i)$ for each $z \in C$ using (7)
return $\arg\max_{z \in C} \ell_t(z \mid x_i)$

E PROMPTS

E.1 GEOMETRY REASONING

Geometry3K, PGPS9K, and GeoQA share these templates. The token `<image>` represents the target diagram.

Geometry reasoning: Solution generation

<image>

Solve the geometry problem by producing a PGPS program. Use the diagram, problem text, and known values together.

Program structure:

- Each step starts with an operator; every following token up to the next operator is one operand of that step.
- Operands may be N0,N1,... (the listed expressions), V0,...,V6 (intermediates), lowercase a,...,z (unknowns), or C0.5,C2,C3,C4,C5,C6,C8,C60,C90,C180,C360.
- End with exactly 'Get target'. All earlier steps must be needed to determine that target.

Operators:

Get(x): return x.

Sum(a,b,c): a+b=c.

Gougu(a,b,c): a^2+b^2=c^2.

Para_Area(b,h,S): b*h=S.

...

Example:

Problem: AC is perpendicular to EB at B. Given ED = AC = N0, AE = DC = N1, and AB = N2, find the area of the figure.

Known values: N0 = 15; N1 = 13; N2 = 7

Program:

```
Gougu V0 N2 N1 Para_Area N0 V0 V1 Get V1
```

...

Target problem:

{question}

Known values:

{indexed quantities}

Geometry reasoning: Solution refinement

<image>

Review the candidate PGPS program using the diagram, problem text, and known values. Correct it if needed, or retain it if it is already correct.

Geometry reasoning: Solution refinement (continued)

Program structure:

- Each step starts with an operator; every following token up to the next operator is one operand of that step.
- Operands may be N0,N1,... (the listed expressions), V0,...,V6 (intermediates), lowercase a,...,z (unknowns), or C0.5,C2,C3,C4,C5,C6,C8,C60,C90,C180,C360.
- End with exactly 'Get target'. All earlier steps must be needed to determine that target.

Operators:

Get(x): return x.

Sum(a,b,c): a+b=c.

Gougu(a,b,c): a^2+b^2=c^2.

Para_Area(b,h,S): b*h=S.

...

Example:

Problem: AC is perpendicular to EB at B. Given ED = AC = N0, AE = DC = N1, and AB = N2, find the area of the figure.

Known values: N0 = 15; N1 = 13; N2 = 7

Program:

```
Gougu V0 N2 N1 Para_Area N0 V0 V1 Get V1
```

...

Target problem:

{question}

Known values:

{indexed quantities}

Candidate program:

{candidate solution}

E.2 FINQA

FinQA inputs contain the passage, table, and question, with tables presented as text.

FinQA: Solution generation

Answer the financial question using the passage and table.

Program structure:

- Write calls op(argument1, argument2), separated by commas. The last call gives the answer.
- Arguments are values from the passage, table or question, fixed constants (const_1, const_100, ...), or earlier results #0, #1, ... (zero-indexed).

Operators:

add(a,b)=a+b; subtract(a,b)=a-b; multiply(a,b)=a*b; divide(a,b)=a/b.

FinQA: Solution generation (continued)

Example:

Table excerpt:

Year	Booking Holdings	Nasdaq Composite
2013	100.00	100.00
2018	148.18	165.84

Question: what was the difference in percentage change in booking holding inc . and nasdaq composite index for the five years ended 2018?

Program:

```
subtract(148.18, const_100), divide(#0, const_100),
subtract(165.84, const_100), divide(#2, const_100),
subtract(#1, #3)
```

...

Target problem:

Passage:

{passage}

Table:

{table}

Question:

{question}

FinQA: Solution refinement

Review the candidate program against the financial passage, table, and question. Correct it if needed, or retain it if it is already correct.

Program structure:

- Write calls op(argument1, argument2), separated by commas. The last call gives the answer.
- Arguments are values from the passage, table or question, fixed constants (const_1, const_100, ...), or earlier results #0, #1, ... (zero-indexed).

Operators:

add(a,b)=a+b; subtract(a,b)=a-b; multiply(a,b)=a*b; divide(a,b)=a/b.

Example:

Table excerpt:

Year	Booking Holdings	Nasdaq Composite
2013	100.00	100.00
2018	148.18	165.84

Question: what was the difference in percentage change in booking holding inc . and nasdaq composite index for the five years ended 2018?

FinQA: Solution refinement (continued)

Program:

```
subtract(148.18, const_100), divide(#0, const_100),
subtract(165.84, const_100), divide(#2, const_100),
subtract(#1, #3)
```

...

Target problem:

Passage:

{passage}

Table:

{table}

Question:

{question}

Candidate program:

{candidate solution}

E.3 MATHQA

MathQA inputs contain the question and its numerical quantities, indexed in order of occurrence.

MathQA: Solution generation

Solve the math word problem by writing a MathQA arithmetic program.

Program structure:

- Write steps op(argument1,argument2), separated by |. The last step gives the answer.
- Arguments are n0,n1,... from the listed question numbers, fixed constants (const_1, const_100, ...), or earlier results #0,#1,... (zero-indexed).

Operators:

add(a,b)=a+b; subtract(a,b)=a-b; multiply(a,b)=a*b; divide(a,b)=a/b.

Example:

Question: 6 workers should finish a job in 8 days . after 3 days came 4 workers join them . how many days m do they need to finish the same job ?

Number references: n0=6; n1=8; n2=3; n3=4

Program:

```
add(n0,n3)|multiply(n0,n1)|multiply(n0,n2)|subtract(#1,#2)|
divide(#3,#0)|
```

...

Target problem:

Question: {question}

Number references: {number references}

MathQA: Solution refinement

Review the candidate program against the question and its listed numbers. Correct it if needed, or retain it if it is already correct.

Program structure:

- Write steps op(argument1,argument2), separated by |. The last step gives the answer.
- Arguments are n0,n1,... from the listed question numbers, fixed constants (const_1, const_100, ...), or earlier results #0,#1,... (zero-indexed).

Operators:

add(a,b)=a+b; subtract(a,b)=a-b; multiply(a,b)=a*b; divide(a,b)=a/b.

Example:

Question: 6 workers should finish a job in 8 days . after 3 days came 4 workers join them . how many days m do they need to finish the same job ?

Number references: n0=6; n1=8; n2=3; n3=4

Program:

```
add(n0,n3)|multiply(n0,n1)|multiply(n0,n2)|subtract(#1,#2)|
divide(#3,#0)|
```

...

Target problem:

Question: {question}

Number references: {number references}

Candidate program:

{candidate solution}