Project Aletheia: Verifier-Guided Distillation of Backtracking for Small Language Models

Aradhya Dixit[1]*, Tianxi Liang[2], Jai Telang[3]
[1]Wake Technical Community College, [2]Cornell University, [3]Algoverse
adixit1@my.waketech.edu, tianxi730@gmail.com, jptelang17@gmail.com

Abstract

Small Language Models (SLMs, under 10B parameters) are attractive for private, on-device deployment, yet they frequently fail on strict constraint-satisfaction problems due to linear, overconfident reasoning traces that do not recover from early mistakes. We introduce *Verifier-Guided Distillation*, a training protocol that transfers the *process of error repair*—explicit conflict detection and backtracking—rather than only correct final answers. By training a 7B model on verified reasoning traces that include mistakes and self-corrections, we show that latent verification behavior can emerge in small models, enabling them to occasionally stop, detect contradictions, and revise earlier assumptions.

Introduction

Small Language Models (SLMs, ¡10B parameters) are attractive for on-device deployment but often fail on strict constraint-satisfaction tasks. For Boolean satisfiability (SAT), early assignment errors propagate, producing invalid solutions, since even a single falsified clause invalidates the entire assignment (Davis and Putnam 1960).

Frontier models achieve strong reasoning performance by producing intermediate hypotheses and iteratively revising them before committing to a final answer (OpenAI 2024; DeepMind 2025), akin to "slow thinking" in cognitive science (Kahneman 2011). Standard supervised training, however, emphasizes monotonic solution paths, encouraging overconfident continuation even after errors (Wei et al. 2022; Kojima et al. 2022; Wang et al. 2022).

We argue that SLM failures are not just capacity-limited but also reflect a *training-data bias*: most corpora overrepresent flawless reasoning and underrepresent recoverable errors. We propose **Project Aletheia**, which distills the *process of error repair*—detecting conflicts, revising state, and continuing—into small models. By supervising SLMs with traces including both mistakes and corrective backtracking, we show latent verification behavior can emerge even under resource constraints.

Our contributions are: (i) introducing *Verifier-Guided Distillation* for transferring backtracking and self-correction into SLMs; (ii) constructing a verified SAT training dataset with explicit conflict and repair steps, alongside a linearized control dataset; and (iii) demonstrating that even a single-epoch LoRA fine-tune can induce observable backtracking behavior in previously monotonic models.

*These authors contributed equally.

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Method: Verifier-Guided Distillation

We aim to distill a *procedure*—conflict detection and state revision—into a small model, rather than distilling only correct final assignments. Our pipeline has three stages: (i) synthesizing backtracking-rich teacher traces, (ii) symbolic verification of trace correctness, and (iii) controlled distillation under a matched control/treatment design.

Problem Setup (SAT as Constraint Satisfaction)

We consider Boolean satisfiability (SAT) instances given in conjunctive normal form (CNF):

$$F = \bigwedge_{i=1}^{m} C_i, \tag{1}$$

where each clause C_i is a disjunction of literals, $C_i = \bigvee_{j=1}^{k_i} \ell_{ij}$, and each literal ℓ_{ij} is either a variable x_p or its negation $\neg x_p$. A (partial) assignment is a mapping $a : \{x_1, \ldots, x_n\} \to \{0, 1, \perp\}$ where $\perp$ denotes unassigned. A *conflict* occurs when some clause is falsified under the current assignment:

$$\exists i \text{ s.t. } C_i(a_t) = 0, \tag{2}$$

i.e., every literal in C_i evaluates to false under a_t (with no remaining unassigned literal that could make it true). We represent solver state as $s_t = (a_t, H_t)$, where a_t is the current (partial) assignment at step t and H_t is a trace of emitted reasoning steps.

Verification and Dataset Construction

Teacher traces may hallucinate intermediate steps; we enforce correctness only on final assignments using PySAT (Ignatiev, Morgado, and Marques-Silva 2018). From 500 generation attempts, 496 traces (99.2%) passed verification. **Caveat:** The dataset is small and may not cover all types of conflicts (single- vs multi-variable, early vs delayed), and traces reflect the heuristics of the teacher model. These factors may limit generalization but are sufficient for initial proof-of-concept distillation.

Negative Constraint Injection (Backtracking-Supervision Trace Synthesis)

To generate supervision that explicitly encodes self-correction, we prompt Gemini 2.5 Pro (DeepMind 2025) to act as a backtracking engine rather than a direct-answer oracle. At each decision step, the teacher proposes an assignment, runs an explicit constraint check, and, upon detecting a conflict, emits a standardized token `[CONFLICT]` followed by a revision routine.

Teacher policy. We treat the teacher trace as a sequence of actions π_T producing tokens and assignment operations. At step t, the teacher chooses either:

- **Decide:** select an unassigned variable x_p and set $a_t(x_p) \leftarrow v$ for $v \in \{0, 1\}$.
- **Verify:** evaluate all clauses under a_t; if any clause is falsified, emit `[CONFLICT]` and identify a violated clause.
- **Backtrack:** revert some prior decision(s), e.g. undo the last assignment or a short window of recent assignments:

$$a_{t+1} \;=\; \text{Revert}(a_t;\, \Delta), \tag{3}$$

  where Δ encodes which variable(s) to unassign (or flip and then re-check).

Why "negative" constraints help. Standard chain-of-thought distillation overrepresents *successful* partial assignments, providing weak supervision for what to do when a clause becomes falsified (Wei et al. 2022). Negative Constraint Injection forces the teacher to surface failure states and repair moves, creating training signals aligned with the desired behavior:

$$\text{If } \exists i:\; C_i(a_t) = 0 \;\Rightarrow\; \text{emit } \texttt{[CONFLICT]} \text{ and revise } a_t. \tag{4}$$

Trace format. Each teacher trace is serialized as a structured natural-language log containing: (i) the current assignment state, (ii) clause-check outcomes, (iii) an explicit conflict marker, and (iv) corrective backtracking steps. This yields trajectories that encode both incorrect hypotheses and their resolution.

Symbolic Verification and "Golden" Dataset Construction

Because teacher traces can hallucinate satisfying assignments or misreport clause evaluations, we enforce correctness with a symbolic judge. For each SAT instance F and final assignment $a^\star$ produced by a teacher trace, we verify:

$$\forall i \in \{1, \ldots, m\},\;\; C_i(a^\star) = 1. \tag{5}$$

We implement this judge using the PySAT toolkit (Ignatiev, Morgado, and Marques-Silva 2018). Only traces whose final assignments satisfy the CNF under exact evaluation are retained. From 500 generation attempts, 496 pass verification (99.2%), forming our "golden" dataset.

Control vs. Treatment: Matched Ablation of Backtracking Signal

To isolate whether backtracking supervision (not just exposure to SAT solutions) induces verification loops, we derive two training corpora from the same verified traces.

Algorithm 1: Symbolic verification of a teacher trace (PySAT).

Require: CNF formula F; teacher-proposed final assignment $a^\star$; PySAT solver
Ensure: `verified` $\in \{0, 1\}$
1: Encode F as PySAT CNF object
2: Translate $a^\star$ into a list of signed literals (e.g., $x = 1 \mapsto +x$, $x = 0 \mapsto -x$)
3: `verified` $\leftarrow$ `Solve`$(F \wedge a^\star)$
4: **return** `verified`

Listing 1: Pseudo-code for trace linearization (control construction).

```
def linearize(trace_lines):
    out = []
    skip = False
    for line in trace_lines:
        if "[CONFLICT]" in line:
            skip = True
            continue
        if skip:
            # skip backtracking
                narration + reverted
                steps
            if "New Proposal" in line or
                 "Revised" in line:
                skip = False
                out.append(line)
            continue
        out.append(line)
    return out
```

Treatment (Aletheia traces). We retain the full sequence including incorrect proposals, clause-check commentary, explicit `[CONFLICT]` markers, and corrective backtracking.

Control (Linearized traces). We remove conflict/backtracking spans and stitch the remaining steps into a monotonic reasoning path that ends in the same verified satisfying assignment. If a trace contains:

$$\cdots \rightarrow \texttt{[CONFLICT]} \rightarrow \text{Backtrack} \rightarrow \text{Revised proposal} \rightarrow \cdots$$

we drop the `[CONFLICT]` block and any reverted steps, keeping the revised proposal and subsequent successful steps. Thus, control and treatment share (i) the same task distribution, (ii) the same final answers, and (iii) similar token budgets, differing primarily in whether the student sees explicit self-correction patterns.

Student Training (LoRA Distillation Under Edge Constraints)

We fine-tune Qwen-2.5-7B-Instruct (Qwen Team 2025) via supervised imitation of teacher traces, using parameter-efficient LoRA adapters (Hu et al. 2021) and 4-bit quantization to approximate edge deployment constraints (Dettmers et al. 2023; Face 2023).

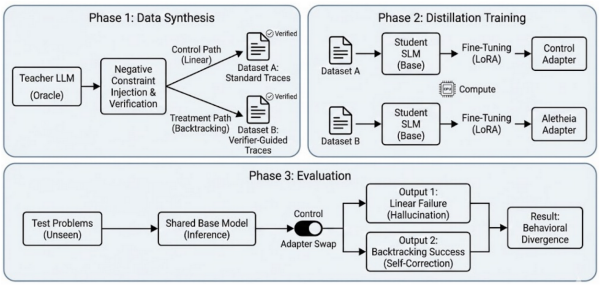

Figure 1: End-to-end Verifier-Guided Distillation pipeline. Teacher generates backtracking traces under Negative Constraint Injection; PySAT verifies final assignments; control vs. treatment adapters are trained and evaluated via adapter hot-swapping.

Objective. Let $y = (y_1, \ldots, y_T)$ be the target trace tokens and x be the SAT instance prompt. We minimize autoregressive cross-entropy:

$$\mathcal{L}(\theta) \;=\; -\sum_{t=1}^{T} \log p_\theta(y_t \mid x, y_{<t}). \tag{6}$$

In the treatment condition, y includes explicit `[CONFLICT]` and backtracking tokens; in the control condition, these tokens are absent.

LoRA parameterization. For a linear projection $W \in \mathbb{R}^{d\times k}$, LoRA learns low-rank updates:

$$W' \;=\; W{+}\Delta W, \quad \Delta W = BA, \quad A \in \mathbb{R}^{r\times k},\; B \in \mathbb{R}^{d\times r}, \tag{7}$$

with rank $r = 16$ and scaling $\alpha = 16$ (Hu et al. 2021). We apply LoRA to all major projection layers $(Q, K, V, O, \text{Gate}, \text{Up}, \text{Down})$.

Compute and tooling. Training is performed on a single NVIDIA A100-SXM4-40GB GPU using Unsloth (AI 2025). Control and treatment adapters are trained with identical hyperparameters and training budgets.

Hyperparameters. Learning rate 2×10^{-4}, epochs=1, LoRA rank $r = 16$, LoRA scaling $\alpha = 16$.

Evaluation

We evaluate on 40 held-out SATBench problems (Wei et al. 2025) using a shared base model and adapter hot-swapping to compare control vs. treatment under identical decoding. We use temperature 0.6 to encourage exploration, making verification behavior observable rather than suppressed by greedy decoding (Holtzman et al. 2020).

Backtracking Event Rate (Primary Behavioral Metric)

Our primary metric measures whether the model triggers an explicit verification loop. Define an event indicator for a generated response R:

$$\mathbf{1}_{\text{bt}}(R) \;=\; \begin{cases} 1, & \text{if } R \text{ contains } \texttt{[CONFLICT]} \\ & \text{and an explicit revision step,} \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

We report the backtracking event rate over $N = 40$ instances:

$$\text{BER} \;=\; \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{\text{bt}}(R_i). \tag{9}$$

We operationalize "explicit revision" as (i) a backtracking phrase (e.g., "Backtracking", "Reverting state") and (ii) a changed assignment proposal in subsequent lines.

Correctness (Secondary Metric)

Separately from behavior, we optionally verify the final assignment produced by the model with the same PySAT judge (Ignatiev, Morgado, and Marques-Silva 2018), yielding a correctness indicator $\mathbf{1}_{\text{sat}}(a^\star)$. In this 4-page version, we emphasize BER as the key evidence of distilled self-correction.

Results

We evaluate two adapters trained on identical base models and data distributions: a *Control* adapter trained on linearized traces, and an *Aletheia* adapter trained on full backtracking traces. Both are tested under identical decoding settings on 40 held-out SATBench instances, using adapter hot-swapping to isolate the effect of supervision.

Backtracking Event Rate

The primary metric is BER: the fraction of inference traces that contain (i) `[CONFLICT]` and (ii) a subsequent state revision step.

The Control model exhibits no observable verification behavior. Across all 40 test instances, it generates monotonic traces without emitting conflict markers or revising earlier assignments:

$$\text{BER}_{\text{control}} = \frac{0}{40} = 0.$$

In contrast, the Aletheia-trained model triggers explicit verification behavior in 2 out of 40 cases:

$$\text{BER}_{\text{aletheia}} = \frac{2}{40} = 0.05.$$

In these instances, the model emits a `[CONFLICT]` signal, retracts a prior assignment, and proposes an alternative value. Since the two adapters differ only in exposure to failure-and-repair supervision, this behavior is attributable to the distillation signal.

Qualitative Trace Analysis

Figure 2 incorporates an excerpted qualitative trace (same structure as Listing 2), showing (i) a hypothesis proposal, (ii) explicit conflict detection, and (iii) localized revision via backtracking.

Listing 2: Representative Aletheia inference trace exhibiting explicit conflict detection and backtracking.

```
1. Initial State:
   All variables unassigned.
   Assignment set = {}.

2. Propose Assignment:
   Based on Constraint 1 (x3 = false),
   assign x3 = False.
   -> Current State: {x3: False}

3. Check Assignment:
   Verifying against constraints...
   - Constraint 1 satisfied.
   - Constraint 2 violated.

   [CONFLICT] Constraint violation
       detected.
   Backtracking...

   Reverting assignment x3 = False.

4. New Proposal:
   Assign x3 = True.
   -> Current State: {x3: True}
   ...
```

Summary of Empirical Findings

Empirically, we observe a clear behavioral contrast: linear supervision yields zero backtracking behavior, while failure-aware supervision yields non-zero, structured verification behavior. Although the absolute frequency is low, the appearance of any explicit self-correction after a single-epoch LoRA fine-tune supports the claim that the *mechanism* of backtracking can be transferred through supervised distillation.

Discussion and Future Work

Interpretation

These results support a data-topology hypothesis: SLM reasoning failures are partly due to limited exposure to *recoverable errors* during training. Conventional reasoning supervision can implicitly teach that "reasoning is monotonic" (Wei et al. 2022), whereas Aletheia supervision teaches a conditional control rule: *when a contradiction is detected, pause and revise state*. The exclusive appearance of conflict-and-repair behavior in the Aletheia condition suggests that self-correction is a learnable routine, not only an emergent artifact of scale or test-time compute (OpenAI 2024; DeepMind 2025).

Limitations

This study demonstrates feasibility rather than robustness. First, the Backtracking Event Rate (BER) is low (5%), so self-correction is rare. Second, the evaluation is limited to SAT; generalization to other domains is untested. Third, supervised imitation does not directly incentivize the model to choose backtracking when next-token likelihood is low. Fourth, the current behavioral metric is string-based; parser-based detection and end-to-end satisfiability checks would yield richer evaluation.

Data limitations: Our training dataset is small (496 verified traces) and reflects the heuristics of the teacher model, which may bias backtracking patterns. Conflict types are not systematically controlled, and only final assignments are verified—intermediate steps may contain minor inconsistencies. Control traces are linearized from treatment traces, sharing final assignments, which could affect comparisons. These factors constrain generalization but suffice for a proof-of-concept demonstration.

Future Work

RL over repairs (from syntax to strategy). Once the model has learned the *syntax* of backtracking via distillation, reinforcement learning can optimize its *usage*. A simple reward can be issued when a backtracking step resolves a contradiction and yields a satisfying assignment (verified by PySAT), turning rare self-correction into a deliberate strategy (Ignatiev, Morgado, and Marques-Silva 2018). This mirrors a broader shift toward training models to reason with verifiable intermediate checks rather than purely imitating fluent traces (Wang et al. 2022).

Curriculum over failure modes (deeper search). We can progressively introduce (i) delayed conflicts (contradictions only visible after multiple decisions), (ii) multi-variable rollbacks, and (iii) adversarial "tempting" assignments that appear locally consistent but fail globally. Curricula of increasing repair depth would push beyond one-step revisions toward systematic search-like behavior, more aligned with classical SAT procedures (Davis and Putnam 1960).

Stronger behavioral instrumentation. Replace string heuristics with a finite-state detector that tracks explicit assignment diffs (e.g., extracting $\{x_i \mapsto v\}$ from text) and counts (a) retractions, (b) flips, and (c) verified satisfiable completions. This would allow reporting richer metrics such as "conflict precision" (fraction of emitted conflicts that correspond to an actually falsified clause under the model-stated assignment).

Generalization beyond SAT. Verifier-guided distillation is domain-agnostic: any task with an external judge can provide the same "detect $\rightarrow$ repair" supervision. Natural next targets include arithmetic proof checking, program synthesis (unit tests as a verifier), and symbolic planning (state transition checkers), where backtracking is also fundamental.

Edge deployment implications. Because the student runs under strict parameter and quantization constraints, this approach offers a complementary path to logic-capable, privacy-preserving agents without proprietary inference-time scaling (Hu et al. 2021; Dettmers et al. 2023). Distilling *how to recover* (not just what to answer) may be a practical route to robust on-device reasoning.

References

AI, U. 2025. unslothai/unsloth. https://github.com/unslothai/unsloth. Accessed: 2025-12-12.

Davis, M.; and Putnam, H. 1960. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7(3): 201–215.

DeepMind, G. 2025. Gemini 2.5: Our most intelligent AI model. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/. Accessed: 2025-12-12.

Dettmers, T.; Pagnoni, A.; Holtzman, A.; and Zettlemoyer, L. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. *arXiv preprint arXiv:2305.14314*.

Face, H. 2023. Making LLMs even more accessible with bitsandbytes, 4-bit quantization and QLoRA. https://huggingface.co/blog/4bit-transformers-bitsandbytes. Accessed: 2025-12-12.

Holtzman, A.; Buys, J.; Du, L.; Forbes, M.; and Choi, Y. 2020. The Curious Case of Neural Text Degeneration. In *International Conference on Learning Representations (ICLR)*. Originally released as arXiv:1904.09751 (2019).

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2021. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*. Springer.

Kahneman, D. 2011. *Thinking, Fast and Slow*. Farrar, Straus and Giroux.

Kojima, T.; Gu, S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2022. Large Language Models are Zero-Shot Reasoners. *arXiv preprint arXiv:2205.11916*.

OpenAI. 2024. OpenAI o1 System Card. https://openai.com/index/openai-o1-system-card/. Accessed: 2025-12-12.

Qwen Team. 2025. Qwen/Qwen2.5-7B-Instruct. https://huggingface.co/Qwen/Qwen2.5-7B-Instruct. Accessed: 2025-12-12.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; and Zhou, D. 2022. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *arXiv preprint arXiv:2203.11171*.

Wei, A.; Wu, Y.; Wan, Y.; Suresh, T.; Tan, H.; Zhou, Z.; Koyejo, S.; Wang, K.; and Aiken, A. 2025. SATBench: Benchmarking LLMs' Logical Reasoning via Automated Puzzle Generation from SAT Formulas. https://arxiv.org/abs/2505.14615. Accessed: 2025-12-12.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; and Zhou, D. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *arXiv preprint arXiv:2201.11903*.