# Late Breaking Results: A Multi-Agent System for Adaptive Optimization of Large-Scale Conjunctive Normal Form

Zhiyuan He
Chinese University of Hong Kong

Rongliang Fu*
Chinese University of Hong Kong

Pin-Yu Chen
IBM Research AI

Tsung-Yi Ho
Chinese University of Hong Kong

## Abstract

Large-scale CNFs exceed LLM context windows, and direct rewriting can break equivalence. SAT-Helper is a zero-shot multi-agent CNF optimizer that samples sliding windows, selects small high-impact blocks and local strategies, rewrites only those blocks, and accepts updates only after SAT-based equivalence checking. This jointly addresses scalability, adaptivity, and correctness. On CNFgen and SAT Competition 2025, SAT-Helper cuts solving time by **68.0% and 80.2%** while reducing memory by **18.7–25.7%**. The results show that verified input-side optimization is a practical direction for LLM-assisted SAT solving.

## 1 Introduction

SAT is a foundational problem in computer science with broad applications in verification, testing, theorem proving, and combinatorial optimization [2]. In practice, solver performance depends not only on the search algorithm itself, but also on the structural quality of the input CNF. Classical preprocessors therefore play an important role in reducing redundancy and improving solver efficiency before search begins [4].

Meanwhile, recent LLM-based SAT research has mostly focused on improving solver implementations rather than optimizing the input formula. Representative efforts use LLMs for solver code generation, heuristic synthesis, debugging, or structure extraction [6–8, 10]. These studies suggest that LLMs can reason about SAT-related artifacts, but they leave open a complementary question: can an LLM improve solving by directly rewriting the CNF itself?

This direction is attractive, but direct CNF rewriting faces three obstacles. First, industrial CNFs can contain hundreds of thousands of clauses, far beyond the effective context window of current LLMs such as Llama and Qwen [11]. Second, different clause regions often benefit from different simplification strategies, so uniform rewriting is usually inefficient. Third, any non-equivalent modification may change satisfiability and invalidate the solver result.

**SAT-Helper** addresses these issues with a zero-shot multi-agent pipeline for input-side optimization. It samples sliding windows from large CNFs, lets a CNF Analyzer select small high-impact blocks and recommend a local strategy, rewrites only those blocks with an LLM-based Optimizer, and merges a candidate only if a SAT-based Equivalence Checker proves it correct. Compared with prior preprocessing pipelines that rely on intermediate representations such as CNF-to-AIG conversion [9], SAT-Helper works directly on CNF and is easier to deploy as a plug-and-play tool.

**Table 1. Performance comparison on CNFgen benchmarks (averaged over 50 instances). Best results in bold.**

| Method | Coverage | Time (s) | Mem. (MB) |
|---|---|---|---|
| Original CNF | 100% | 306.7 | 97.6 |
| EDA4SAT | 100% | 867.4 | 94.8 |
| Direct LLM (Llama3 *) | 12.8% | 563.8 | 103.3 |
| Direct LLM (Qwen3 *) | 16.5% | 426.7 | 101.7 |
| SAT-Helper (Llama3) | **12.8%** | 103.7 | 75.3 |
| SAT-Helper (Qwen3) | 16.5% | **98.3** | **72.5** |

* May contain non-equivalent outputs.

Our key contributions are: **(1)** a practical multi-agent pipeline for large-scale CNF input optimization; **(2)** adaptive block and strategy selection instead of uniform rewriting; and **(3)** formal verification that guarantees accepted optimizations preserve satisfiability. Across CNFgen and SAT Competition 2025 benchmarks, SAT-Helper consistently reduces solving time and memory while touching only a small fraction of clauses.

## 2 Method

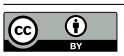
SAT-Helper follows the workflow in Fig. 1. Instead of exposing the whole CNF to the LLM, the system repeatedly samples a window from the formula and optimizes only a small block inside that window. This keeps each prompt within context limits while allowing the system to cover large formulas over multiple iterations. The design is motivated by the observation that many useful simplifications are local: *if a candidate local rewrite is proven equivalent, it can be merged without changing the satisfiability of the formula.*

The **CNF Analyzer** is the key decision module. Given a sampled window, it identifies clause blocks that are likely to matter for downstream solving and selects a local strategy such as subsumption, elimination, resolution-style simplification, or long-clause decomposition. This step is what makes SAT-Helper adaptive: different regions of the same CNF may exhibit very different syntactic patterns, so the system avoids the one-strategy-fits-all behavior that limits both classical fixed-rule pipelines and naive direct prompting.

The **CNF Optimizer** then rewrites only the selected block, which sharply reduces unnecessary edits and makes optimization more targeted than end-to-end LLM rewriting. In our experiments, this selective behavior is important because large performance gains come from modifying a small portion of clauses rather than from aggressively rewriting the entire formula.

Correctness is enforced by the **Equivalence Checker**. Every candidate rewrite is validated with SAT-based checking before it is merged back into the original formula; unsafe or non-equivalent candidates are simply discarded. This verifier is crucial because LLM outputs are probabilistic and may otherwise introduce subtle semantic errors. The combination of *sliding-window partitioning*, *adaptive strategy selection*, and *formal verification* is the core innovation that

*Corresponding author: rlfu@cse.cuhk.edu.hk.




*DAC '26, Long Beach, CA, USA*

ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3807722

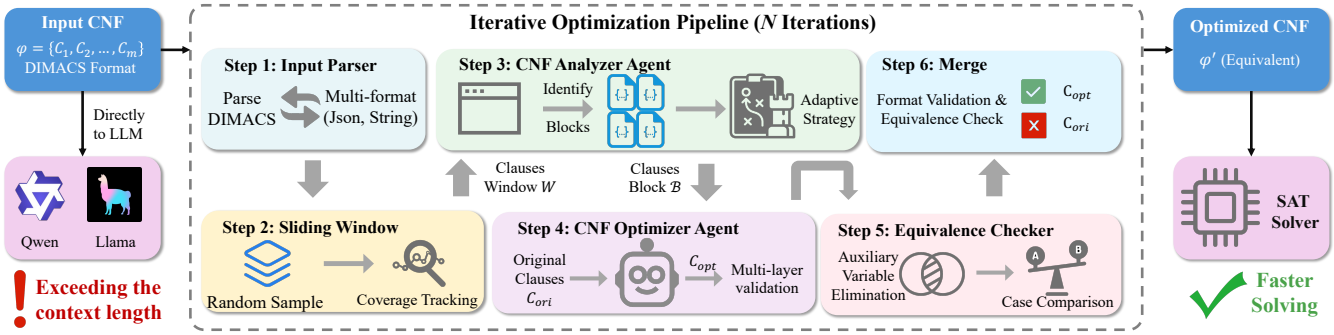



**Figure 1. SAT-Helper pipeline. A large CNF is partitioned by sliding windows; the Analyzer selects optimization blocks and strategies; the Optimizer rewrites them; the Equivalence Checker accepts only proven-equivalent candidates; and the Merger updates the global CNF.**

**Table 2. Detailed performance comparison on SAT Competition 2025 benchmarks (5 representative cases). SAT-Helper (Qwen3-30B) vs. Kissat baseline.**

| Case | CNF | | Kissat | | SAT-Helper | |
|---|---|---|---|---|---|---|
| | Vars | Clauses | Time | Mem. | Time | Mem. |
| C1 | 120036 | 704717 | 535.1 | 103 | 79.2 | 97 |
| C2 | 29934 | 247657 | 1066.3 | 100 | 238.0 | 74 |
| C3 | 31294 | 802297 | 742.8 | 98 | 156.3 | 71 |
| C4 | 82269 | 271217 | 598.6 | 112 | 108.5 | 102 |
| C5 | 24624 | 212761 | 893.4 | 107 | 204.7 | 80 |
| Ave. ratio | 57631 | 447730 | 1 | 1 | 0.20 | 0.81 |

distinguishes SAT-Helper from both fixed-rule preprocessors and direct LLM rewriting.

## 3 Experimental Results

We evaluate SAT-Helper on CNFgen [5] and SAT Competition 2025 [1] instances using downstream SAT solving time and memory as the main metrics. Kissat [3] is the backend solver for all comparisons. We also compare against direct LLM rewriting and EDA4SAT [9]. The goal is to test whether local, verified optimization can improve performance on large CNFs without rewriting the entire formula.

**Results on CNFgen Benchmarks:** Table 1 shows that SAT-Helper achieves a **68.0%** time reduction and **25.7%** memory reduction on CNFgen while touching only **16.5%** of clauses. Direct LLM rewriting is much slower and unreliable, and EDA4SAT degrades on these large instances, highlighting the value of targeted optimization plus verification. The result also suggests that input-side optimization is effective even when only a small part of the formula is changed.

**Case Studies on SAT Competition 2025:** On SAT Competition 2025, SAT-Helper reduces average solving time from 767.2s to 157.3s (**80.2%**) and memory from 104.0MB to 84.8MB (**18.7%**). The gains are consistent across all five large instances, showing that the method scales beyond synthetic benchmarks. Since these instances contain roughly 200k–800k clauses, the result directly validates the need for sliding-window partitioning.

**Ablation Study** Figure 2 (left) confirms that all SAT-Helper components are essential. Removing the CNF Analyzer or Optimizer pushes solving times beyond 600s, highlighting the need for accurate block selection and targeted rewriting. Omitting the Equivalence Checker risks logically incorrect rewrites, while removing the sliding window makes processing large instances impossible due to LLM context limits. The right panel demonstrates that adaptive strategy selection drastically outperforms any single fixed strategy, including the best alternative (long-clause decomposition). This confirms that effective CNF optimization requires tailoring transformations to local

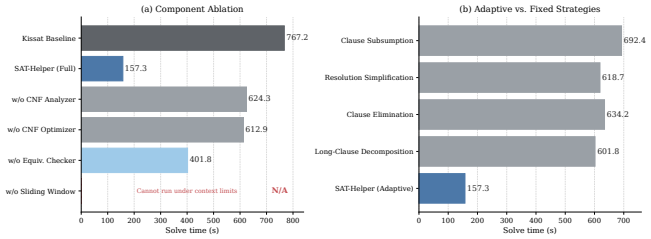



**Figure 2. The ablation and strategy-comparison results on SAT Competition 2025. Left: component ablation. Right: adaptive strategy selection versus fixed single strategies. Lower is better in both panels.**

clause patterns rather than applying uniform rules. In conclusion, SAT-Helper succeeds because it chooses *where* to optimize, *how* to optimize, and *when* to reject an unsafe rewrite.

## 4 Conclusion

We presented SAT-Helper, a zero-shot multi-agent system that tackles large-scale CNF optimization. By targeting 16.5% of clauses, it accelerates solving speeds by 68.0–80.2% and cuts memory usage by 18.7–25.7%. Operating as a plug-and-play framework with no need for fine-tuning or data preprocessing, SAT-Helper can be integrated with diverse LLMs. This work highlights input-side optimization as a practical direction for LLM-assisted SAT solving.

## Acknowledgments

The research work described in this paper was conducted in the JC STEM Lab of Intelligent Design Automation funded by The Hong Kong Jockey Club Charities Trust.

## References

[1] [n. d.]. SAT Competition 2025. https://satcompetition.github.io/2025.
[2] Jeremias Berg et al. (Eds.). 2024. *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions.*
[3] Armin Biere et al. 2024. CaDiCaL, Gimsatul, IsaSAT and Kissat Entering the SAT Competition 2024. (2024).
[4] Niklas Eén and Armin Biere. 2005. Effective preprocessing in SAT through variable and clause elimination. In *SAT.*
[5] Massimo Lauria. [n. d.]. CNFgen: A Generator of Combinatorial Benchmarks.
[6] Kyla H Levin et al. 2025. ChatDBG: Augmenting Debugging with Large Language Models. *ACM on Software Engineering* 2, FSE (2025), 1892–1913.
[7] André Schidler and Stefan Szeider. 2025. Extracting problem structure with LLMs for optimized SAT local search. *arXiv:2501.14630* (2025).
[8] Junjie Sheng et al. 2025. SolSearch: An LLM-Driven Framework for Efficient SAT-Solving Code Generation. In *ICSE-NIER.*
[9] Zhengyuan Shi et al. 2024. EDA-driven preprocessing for SAT solving. (2024).
[10] Yiwen Sun et al. 2024. AutoSAT: Automatically optimize SAT solvers via large language models. (2024).
[11] An Yang et al. 2025. Qwen3 Technical Report. *arXiv:2505.09388* (2025).