PRELIMINARY PREPRINT VERSION – DO NOT CITE
This is not the version of record. The final version will appear in the Proceedings of IJCAI-ECAI 2026.

# Bridging LLMs and SAT Solving: Automated Evolution of High-Performance Heuristics

**Mao Luo[1], Hang Ding[1], Chu-Min Li[2,3,*], Junjie Zhang[4], Zhiwei Ye[1,*], Zhipeng Lü[4], Caiquan Xiong[1], Xinyun Wu[1]**

[1]Hubei University of Technology, Wuhan, China
[2]Université de Picardie Jules Verne, Amiens, France
[3]Aix-Marseille Université, Marseille, France
[4]Huazhong University of Science and Technology, Wuhan, China
luomao@hbut.edu.cn, dinghang@hbut.edu.cn, chu-min.li@u-picardie.fr,
junjie_zhang@hust.edu.cn, hgcsyzw@hbut.edu.cn, zhipeng.lv@hust.edu.cn,
xiongcq@hbut.edu.cn, xinyun@hbut.edu.cn

IJCAI-ECAI 2026 PREPRINT

## Abstract

Despite decades of intensive research and optimization, modern Boolean Satisfiability (SAT) solvers have reached a plateau where significant performance gains are increasingly difficult to achieve. While Large Language Models (LLMs) have demonstrated remarkable capabilities in pattern recognition and code generation for combinatorial optimization, their direct application to highly optimized SAT solvers remains a formidable challenge due to the extreme complexity and sensitivity of solver heuristics. In this paper, we introduce AESAT (Auto-Evolving SAT solving), a novel neuro-symbolic framework designed to automatically evolve and optimize the heuristic functions of SAT solvers. AESAT employs a memetic-inspired approach, synergizing LLM-guided individual optimization with evolutionary exploration. By leveraging self-optimized prompting techniques, our framework enables LLMs to iteratively discover and refine sophisticated heuristics that bypass the limitations of human-engineered designs. The efficacy of AESAT is demonstrated by its flagship derivative, AE-Kissat-MAB, which won the main track of the 2025 International SAT Competition by a wide margin. This result represents the first time an LLM-enhanced solver has dominated the world's premier SAT competition, marking a paradigm shift in the automated design of reasoning algorithms.

## 1 Introduction

The Boolean Satisfiability Problem (SAT) was the first problem proven to be NP-complete and remains a cornerstone of computer science and logic [Cook, 1971]. Beyond its theoretical significance, SAT solving has become a critical industrial technology with applications spanning hardware and software verification, artificial intelligence, automated planning, and bioinformatics [Vizel *et al.*, 2015; Marques-Silva, 2008], thanks to the continuous development of SAT solvers over several decades, making the SAT formalism a highly competitive generic approach for solving complex real-world problems [Biere *et al.*, 2021].

However, this success has led to a performance plateau: Modern solvers have become so sophisticated and finely tuned that achieving further significant improvements is increasingly difficult. Historically, manual SAT solver optimization has been a grueling process, requiring deep domain expertise, years of trial-and-error, and exhaustive experimental validation. For a human research team, designing an optimization to solve a few additional instances in a SAT competition benchmark often represents years of dedicated effort.

The emergence of Large Language Models (LLMs) offers a potential escape from this human-expert bottleneck. Unlike traditional heuristic design, which relies on intuition, LLMs can leverage complex patterns learned from vast datasets to generate novel, and sometimes counter-intuitive, algorithmic strategies [Naveed *et al.*, 2023]. While AutoSAT [Sun *et al.*, 2024] was the first framework to utilize LLMs for SAT policy optimization, its scope was limited. By targeting a relatively simple solver (easySAT [Chen and Zhang, 2022]) and utilizing unorganized requests for function improvements, AutoSAT lacked the deep optimization and structural diversification necessary to improve highly optimized solvers like Kissat_MAB [Biere *et al.*, 2020; Cherif *et al.*, 2021].

In this paper, we propose AESAT (Auto-Evolving SAT solving), a novel memetic framework designed to automatically evolve SAT solver heuristics using LLMs. AESAT bridges the gap between neural generation and symbolic reasoning by:

1. Synergizing Individual and Evolutionary Search: It organizes LLM requests through a combination of local refinement and global evolutionary exploration.
2. Self-Optimizing Prompts: It employs a prompting mechanism that evolves based on historical performance data to maximize the LLM's generative potential.

The efficacy of our approach is demonstrated by AE-Kissat-MAB, a solver optimized via AESAT that won the

PRELIMINARY PREPRINT VERSION – DO NOT CITE
This is not the version of record. The final version will appear in the Proceedings of IJCAI-ECAI 2026.

main track of the 2025 International SAT Competition by a wide margin. This victory marks a historic milestone: the first time an LLM-enhanced solver has dominated this prestigious and rigorous competition. By providing a low-cost, automated alternative to manual engineering, AESAT offers a scalable path for optimizing solvers for SAT and other hard combinatorial problems, representing a paradigm shift in how we design reasoning algorithms.

The remainder of this paper is organized as follows: Section 2 reviews related works. Section 3 details the AESAT framework and its five core functions. Section 4 presents the application of AESAT to Kissat_MAB, including a comparative analysis between LLM-generated and human-expert heuristics. Section 5 provides a comprehensive empirical analysis, and Section 6 concludes the paper.

## 2 Related Works

As LLMs demonstrate increasing proficiency in structural reasoning and code generation, they are being used for combinatorial optimization as heuristic generators [Da Ros *et al.*, 2025]. This shift allows researchers to treat strategy design as a language modeling task [Ye *et al.*, 2024]. Current research in this domain can be broadly categorized into general optimization frameworks and SAT-specific applications.

### 2.1 LLM-Based Evolutionary Optimization

Several frameworks have pioneered the integration of LLMs into evolutionary loops. FunSearch [Romera-Paredes *et al.*, 2023] demonstrated that LLMs can generate and evaluate code within an evolutionary framework to achieve breakthrough discoveries in combinatorial mathematics. Similarly, EvoPrompting [Guo *et al.*, 2024] utilizes evolutionary algorithms to optimize the prompt templates, ensuring precise and structured LLM outputs. In the domain of traditional combinatorial tasks, HeurEvo [Liu *et al.*, 2024b] couples an LLM with a genetic algorithm to serve as an evolution engine, automatically discovering efficient rules for the Traveling Salesman Problem. These works establish the feasibility of LLMs in structured search tasks but often focus on problems with simpler constraints than modern SAT solving.

### 2.2 LLMs in SAT Solving

Due to the extreme complexity and delicate internal dependencies of modern SAT solvers, LLM-driven improvements have only emerged very recently. AutoSAT [Sun *et al.*, 2024] was the first framework to apply LLMs to SAT policy optimization. Using the EasySAT solver [Chen and Zhang, 2022] as a baseline, AutoSAT employed two strategies: Greedy Hill Climbing (GHC), which modifies functions sequentially, and an Evolutionary Algorithm (EA) that modifies multiple functions simultaneously. However, AutoSAT's approach has two primary limitations. First, if an LLM-generated modification fails initially, it is discarded without the fine-tuning necessary to unlock its potential—a critical flaw when dealing with highly optimized solvers where raw ideas require precise calibration. Second, its EA approach lacks a crossover operation due to its single-parent structure, limiting diversification.

Following the 2025 SAT Competition, SATLUTION [Yu *et al.*, 2025] introduced an evolution framework that utilizes an LLM agent to combine the strengths of five state-of-the-art baseline solvers. While SATLUTION reports impressive speedups over 2025 competition winners, it remains a human-in-the-loop system, requiring manual guidance for high-level search strategies. Furthermore, its resource intensity is substantial, requiring over 20,000 USD in computational costs and resulting in nearly 65,000 lines of code changes.

## 3 The AESAT Framework

We introduce AESAT (Auto-Evolving SAT solving), a lightweight framework designed to automatically refine and optimize a target heuristic function $F$ within a modern SAT solver $A$. AESAT addresses the conventional "fine-tuning" bottleneck through a self-optimized prompting mechanism, and introduces a novel multi-parent evolutionary structure that fosters robust algorithmic diversification. To effectively coordinate and exploit these diverse strategies, the framework is built upon a Mixture-of-Experts (MoE) architecture [Wu *et al.*, 2024; Chan *et al.*, 2023]. Crucially, AESAT achieves these capabilities while maintaining significantly lower computational overhead and greater architectural simplicity than existing LLM-agent architectures.

This section details the underlying MoE architecture alongside the five core algorithms that govern AESAT.

### 3.1 Mixture-of-Experts (MoE) Architecture

The integration of the MoE architecture is grounded in the insight that no single large language model (LLM) excels uniformly across all optimization tasks. By deploying specialized models at distinct phases of the pipeline, AESAT maximizes the comparative advantages of diverse LLMs at distinct phases of the pipeline. Specifically, the framework orchestrates the heuristic optimization process by decomposing it into three specialized, synergistic roles:

**Idea Generation (GPT-4.5):** It leverages cross-domain knowledge to propose novel algorithmic concepts and diversify the search space.

**Code Implementation in local optimization (Claude 3.7):** Known for high fidelity in structured code generation, it ensures that natural language concepts are translated into robust, compilable, and convention-adherent $C$ code.

**Idea Analysis and Code Implementation (DeepSeek-R1):** It utilizes superior logical reasoning and chain-of-thought capabilities to perform feasibility assessments and pattern recognition on failed or successful heuristics. It also generates $C$ code for combining the strengths of two functions in a crossover operation.

### 3.2 Algorithmic Overview

AESAT explores the space of heuristic variants to optimize the target function $F$ within a solver $A$. This orchestration is formalized in Algorithm 1. The framework takes as input the baseline function $F$, the solver $A$, a set of adaptive guidelines ($tips$), and a training set of SAT instances $I$. It outputs a synthesized function $t^*$ designed to replace $F$ in $A$, maximizing solving performance as quantified by the Penalized

IJCAI-ECAI 2026 PREPRINT

**Algorithm 1** The AESAT Framework

**Input**: heuristic function $F$ requiring optimization, SAT solver $A$, a set of $tips$, a set of training SAT instances $I$
**Parameters**: $populationSize$
**Output**: an improved function $t^*$ and a set of adaptive $tips$

Call GPT with prompt: You are an expert SAT Solver Researcher. Analyze solver $A$ and the existing heuristic $F$. Generate $populationSize$ novel heuristic functions to replace $F$ in $A$. Each generation must strictly adhere to the constraints defined in $tips$. The final output, set $T$, must contain $populationSize$ individuals, each consisting of a formal natural language description
**for** each function $t \in T$ **do**
  $t' \leftarrow$ LocalOptimization$(t, A, tips, I)$
  $T \leftarrow (T \setminus \{t\}) \cup \{t'\}$
**end for**
$(t^*, tips) \leftarrow$ EvolutionaryUpdate$(T, A, F, tips, I)$
**return** $(t^*, tips)$

Average Runtime (PAR2) score[1]. We denote the PAR2 score of solver $A$ executing a modified heuristic $t$ on the dataset as $\mathrm{PAR2}(A, t)$, where a lower score indicates superior performance.

AESAT begins by calling GPT to generate $\mathrm{populationSize}$ functions to replace $F$ in $A$, each represented in an individual. At this stage, the functions are described in formal natural language and no code is generated. Then AESAT performs local optimization for each function, and evolutionary exploration for diversification, described in the following subsections. Note that since each individual should be evaluated on the training set, an iteration in local optimization or evolutionary exploration takes much more time than in a classical memetic algorithm.

### 3.3 Local Optimization and Individual Refinement

Unlike classical memetic algorithms that rely on stochastic local search, AESAT employs Algorithm 2 (LocalOptimization) for individual refinement, which is more costly, because each individual needs to be evaluated on the training set. For an input function $t$, AESAT generates $nbNeighbors$ variants through slight modifications, using the ancestral lineage and historical performance data to guide these modifications. The best variant is kept for the next iteration. This ensures that the search does not discard promising ideas prematurely, but instead fine-tunes them toward local optima.

### 3.4 Evolutionary Exploration and Selection

After individual refinement, Algorithm 3 (EvolutionaryUpdate) performs global exploration through a structured crossover. The DeepSeek-R1 identifies the most promising pairs $(s_1, s_2)$ and synthesizes their strengths into a hybrid function $t$. To manage population health, AESAT employs

[1]Consistent with International SAT Competitions, the PAR2 score of a solver on a set of instances is defined as the sum of runtimes for all solved instances plus twice the timeout threshold ($2 \times$ timeout) for any unsolved instances

**Algorithm 2** LocalOptimization

**Input**: heuristic function $t$, solver $A$, set of tips $tips$, set of SAT instances $I$
**Parameters**: $localOptimLimit$, $nbNeighbors$, $timeOut$
**Output**: an improved heuristic function $t$

**for** $i = 1$ to $localOptimLimit$ **do**
  **for** $j = 1$ to $nbNeighbors$ **do**
    Call GPT with prompt: As a SAT solver researcher, read solver $A$ and its base heuristic function $t$. Considering the ancestral lineage of $t$ and adhering to guidelines in $tips$, generate a variant $t_j$ by applying slight modifications
    Call Claude to generate the complete compliant $C$ code implementation for $t_j$ according to its formal language description created by GPT
    Run solver $A$ with $t_j$ for each instance in $I$, stopped by $timeOut$
    Compute $par2(A, t_j)$ and record $t$ as $parent$ of $t_j$
  **end for**
  $t \leftarrow argmin_{s \in \{t_1, \ldots, t_{nbNeighbors}\}} par2(A, s)$
  $t \leftarrow$ UpdateHistory$(t)$
**end for**
**return** $t$

a heuristic fitness function $q(t)$ that balances performance, innovation, and stochastic diversity:

$$q(t) = \alpha \times \frac{\mathrm{PAR2}(A, F)}{\mathrm{PAR2}(A, t)} + \beta \times inn(t) + \gamma \times r \quad (1)$$

where $inn(t)$ is an innovation score assigned by the DeepSeek-R1 expert and $r \in [0, 1)$ is a random perturbation to prevent premature convergence. The population is pruned by removing individuals below the median $q(t)$ score while strictly preserving the global best in terms of PAR2.

This evolutionary process is repeated as long as $T$ contains more than two individuals. When $T$ converges to only two individuals, it returns the best function found so far.

### 3.5 Adaptive Meta-Learning

Whenever a function either enters or is generated by the evolutionary exploration phase (Algorithm 3), AESAT invokes DeepSeek-R1 to evaluate its performance against the Kissat4.0.2_MAB baseline via Algorithm 4. Guided by these performance insights, the framework triggers the PromptSelfOptimize mechanism (Algorithm 5) every $roundSize$ iterations within the EvolutionaryUpdate loop. This meta-analysis of the ancestral lineage identifies recurring failure modes and suboptimal logic patterns, enabling the framework to refine its own $tips$ (system prompt).

Through this self-correction, the system masters the meta-process of "learning how to learn," progressively steering the evolution of the heuristics toward more promising directions.

## 4 Case Study: Optimizing the Branching Heuristic Selection for Kissat 4.0.2

Kissat is a highly optimized SAT solver [Biere *et al.*, 2020] that constitutes the basis of most SAT solvers in recent SAT

IJCAI-ECAI 2026 PREPRINT

**Algorithm 3** EvolutionaryUpdate

**Input**: set of heuristic functions $T$, solver $A$, original function $F$ requiring optimization, set of tips $tips$, set of SAT instances $I$
**Parameters**: $roundSize$, $0 < \alpha, \beta, \gamma \leq 1$ such that $\alpha + \beta + \gamma = 1$, *timeOut* in seconds
**Output**: An improved heuristic function $t$ and $tips$

$i \leftarrow 0$
**while** $T$ contains more than two functions **do**
  Call DeepSeek-R1 with prompt: As a SAT solver researcher, perform crossover operations on the set $T$. For every pair of $(s_1, s_2)$ in $T$ deemed most promising based on their characteristics, analysis, and ancestral lineage, generate a new function $t$. The goal is to combine strengths while mitigating weaknesses. The output $t$ must include complete compliant C code, strictly adhering to $tips$. Avoid exhaustive pairwise combinations.
  **for** each new function $t$ combining $s_1$ and $s_2$ given by DeepSeek-R1 **do**
    Run $A$ with $t$ for each instance in $I$, stopped by *timeOut*
    Compute $par2(A, t)$, and record $s_1$ and $s_2$ as $parent1$ and $parent2$ of $t$, respectively
    $T \leftarrow T \cup \{t\}$
    $t \leftarrow$ UpdateHistory$(t)$
  **end for**
  Call DeepSeek-R1 with prompt: For every heuristic $t \in T$, calculate its innovation score $\mathrm{inn(t)}$ by analyzing its description, C code and differences with other functions in $T$, and calculate the fitness score $q(t)$ using Equation (1).
  $t_{best} \leftarrow argmin_{t \in T}\, par2(A, t)$
  Remove all $t \in T$ such that $q(t)$ is smaller than or equal to the median $q$ in $T$
  $T \leftarrow T \cup \{t_{best}\}$
  $i \leftarrow i + 1$
  **if** $i \bmod roundSize = 0$ **then**
    $tips \leftarrow$ PromptSelfOptimize$(t_{best}, tips)$
  **end if**
**end while**
$t_{best} \leftarrow argmin_{t \in T}\, par2(A, t)$
**return** $(t_{best}, tips)$

**Algorithm 4** UpdateHistory

**Input**: heuristic function $t$
**Output**: heuristic function $t$

Call DeepSeek-R1 with prompt: Perform a detailed comparative analysis of heuristic $t$ against its baseline. This analysis must encompass differences in their natural language descriptions, $C$ code, and $\mathrm{par2}$ scores. Based on this comparison, record a concise explanation in $t$, detailing the reason for $t$'s observed performance improvement or degradation relative to Kissat4.0.2_MAB.
**return** $t$

**Algorithm 5** PromptSelfOptimize

**Input**: heuristic function $t$, set of tips $tips$
**Output**: set of optimized $tips$

Call DeepSeek-R1 with prompt: As a system feedback mechanism, analyze heuristic $t$ and perform a detailed comparison against its entire ancestral lineage across four criteria: natural language description, C code, $\mathrm{par2}$ score, and existing performance analysis. Based on identified failures, errors, or suboptimal design choices, update the document $tips$ to prevent the recurrence of the same issues or incorrect design directions in all future generations.
**return** $tips$

competitions. It frequently restarts its search. Each restart develops a binary search tree and can use a different branching heuristic for this. Cherif et al. [2021] integrated a function called restart_MAB (Multi-Armed Bandits) into Kissat to select a branching heuristic among VSIDS [Moskewicz *et al.*, 2001] and CHB [Liang *et al.*, 2016] in the stable mode. The resulted solver is called Kissat_MAB and won the 2021 SAT competition with a large margin.

This section first presents the restart_MAB function, and then describes in details a real evolutionary trajectory produced by AESAT when optimizing it, so that the reader can visualize how AESAT's operations parallel or diverge from those a human expert would perform to achieve similar evolutions of heuristics.

## 4.1 Function Restart_MAB

Consider a restart $R$ using the branching heuristic $h$. The restart_MAB function assigns it a reward. Let $\#\mathrm{decisions}$ denote the number of internal nodes in the tree developed by $R$, and $\#\mathrm{conflicts}$ denote the number of leaves. We use below a reward for $R$ defined in [Liu *et al.*, 2024a]:

$$r(h) = \frac{\log_2(\#\mathrm{decisions})}{\log_2(\#\mathrm{conflicts})} \tag{2}$$

For each heuristic $h \in \{\mathrm{VSIDS}, \mathrm{CHB}\}$, let $n_h$ denote the number of restarts using $h$, and $\bar{r}(h)$ denote the average reward obtained across the $n_h$ restarts using $h$. After $n_{total} = n_{VSIDS} + n_{CHB}$ restarts, the restart_MAB function defines the score of $h$ using the Upper Confidence Bound (UCB) strategy as follows:

$$\mathrm{UCB}(h) = \bar{r}(h) + \sqrt{\frac{4 \times \ln(n_{\mathrm{total}})}{n_h}} \tag{3}$$

$\mathrm{UCB}(h)$ contains two terms: the exploitation term $\bar{r}(h)$ and the exploration term $\sqrt{\frac{4 \times \ln(n_{\mathrm{total}})}{n_h}}$, called $\mathrm{xpr1}(h)$ in the sequel. The restart_MAB function returns the heuristic with the highest UCB score, that will be used in the next restart. So, optimizing restart_MAB is equivalent to optimizing the UCB function. Note that $\mathrm{xpr1}(h)$ penalizes the heuristic with high $n_h$.

IJCAI-ECAI 2026 PREPRINT

We integrate the above restart_MAB function into Kissat 4.0.2[2] to create our baseline Kissat4.0.2_MAB, where Kissat 4.0.2 is a successor of Kissat 4.0.0 [Biere *et al.*, 2024], the winner of the 2024 SAT competition with a wide margin. Then, we task AESAT with evolving the UCB function to improve the performance of Kissat4.0.2_MAB on a benchmark set $I$ of 60 instances randomly selected from the 2024 SAT Competition.

In the sequel, each new UCB function, as well as intermediate terms, will be defined in an equation, and an equation can use the functions defined in previous equations. The performance of a new UCB function is defined to be the performance of Kissat4.0.2_MAB using this UCB function instead of Equation (3), measured by the PAR2 score.

## 4.2 Evolutionary Trajectory of Heuristic Discovery

As previously described, AESAT evolves heuristics through a local optimization stage followed by an evolutionary stage. This subsection illustrates this two-stage process as applied to the restart_MAB function, followed by an adaptive meta-learning.

**Local Refinement**

To begin, AESAT feeds the source code of Kissat4.0.2_MAB and specially restart_MAB to LLMs, and calls GPT to generate 20 ($\mathrm{populationSize}$) restart_MAB functions, each with a different UCB definition. After this, Algorithm 2 is called to optimize locally each of the 20 functions.

For example, the seventh function $t_7$ introduces hysteresis by temporarily boosting the UCB score of a newly selected heuristic $h$ to prevent rapid and unstable switching. The first cycle of local optimization of $t_7$ in Algorithm 2 begins by calling GPT and Claude to generate three variants of $t_7$, of which $t_{7.1}$ boosts UCB($h$) by a factor of 1.2:

$$\mathrm{UCB}_{7.1}(h) = \begin{cases} \mathrm{UCB}(h) \times 1.2 & \text{if } h \text{ was just switched on} \\ & \text{in the previous restart} \\ \mathrm{UCB}(h) & \text{otherwise} \end{cases} \tag{4}$$

In variant $t_{7.3}$, the boost is not a factor, but a fixed bonus $a = 0.05 \times n_{total}$:

$$\mathrm{UCB}_{7.3}(h) = \begin{cases} \mathrm{UCB}(h) + a & \text{if } h \text{ was just switched on} \\ & \text{in the previous restart} \\ \mathrm{UCB}(h) & \text{otherwise} \end{cases} \tag{5}$$

All the three variants are evaluated on $I$ and $t_{7.1}$ is found to be the best. The next cycle of local optimization begins by calling LLMs to generate three variants of $t_{7.1}$, among which $t_{7.1.2}$ boosts UCB($h$) for the next three successive restarts after $h$ is switched on, its value being multiplied by 1.2, 1.1 and 1.05, respectively, similar to Equation (4). After evaluation, it is found to be the best variant among the three variants.

The next cycle identifies $t_{7.1.2.2}$ to be the best among three variants of $t_{7.1.2}$, which boosts UCB($h$) by a factor $f = 1.5 \times 0.85^k$, if $h$ is just used for $k$ consecutive restarts. This is the final function returned by optimizing $t_7$.

$$\mathrm{UCB}_{7.1.2.2}(h) = \begin{cases} \mathrm{UCB}(h) \times f & \text{if } h \text{ was just used for } k \\ & \text{consecutive restarts} \\ \mathrm{UCB}(h) & \text{otherwise} \end{cases} \tag{6}$$

The same local optimization process is applied to functions $t_{11}$ and $t_{13}$, where $t_{11}$ divides the raw reward of heuristic $h$ by a value depending on $n_h$, and $t_{13}$ uses an entropy-guided selection mechanism that prefers the heuristic with higher reward variance when its UCB score is not too bad, favoring uncertain but high-potential options.

After several cycles, an optimized variant $t_{11.3.3.2}$ for $t_{11}$ is identified. It uses the same second term $\mathrm{xpr1}$ in the right as in Equation (3), but a normalized reward $r_n$:

$$\mathrm{UCB}_{11.3.3.2}(h) = \bar{r}_n(h) + xpr1(h) \tag{7}$$

where $r_n$ is normalized by considering the average numbers of internal tree nodes and leaves, $\mathrm{avg10_h}(\#\mathrm{decisions})$ and $\mathrm{avg10_h}(\#\mathrm{conflicts})$, respectively, in the 10 previous restarts using $h$. Concretely:

$$r_n(h) = r(h) \times \frac{\mathrm{avg10_h}(\#\mathrm{conflicts})}{\mathrm{avg10_h}(\#\mathrm{decisions})} \tag{8}$$

And $t_{13.1.2.1}$, a locally optimized variant of $t_{13}$, is also obtained:

$$\mathrm{UCB}_{13.1.2.1}(h) = \mathrm{UCB}(h) + 0.5 \times e^{-0.1 \times n_{total}} \times var(r(h)) \tag{9}$$

where $var(r(h))$ is the variance of $r(h)$ (see Section 4.3).

**Evolutionary Exploration**

After locally optimizing each individual, AESAT performs evolutionary update by calling Algorithm 3 to combine promising individuals. The combining of $\mathrm{UCB}_{7.1.2.2}$ and $\mathrm{UCB}_{13.1.2.1}$ gives

$$\mathrm{UCB}_{7+13}(h) = \begin{cases} \bar{r}_{wf}(h) + \mathrm{xpr1}(h) & h \text{ was just used for } k \\ & \text{consecutive restarts,} \\ \bar{r}_w(h) + \mathrm{xpr1}(h) & \text{otherwise} \end{cases} \tag{10}$$

where $\bar{r}_w(h) = \bar{r}(h) \times w$ with $w = 1.0 + e^{-0.01 \times n_{total}} \times var(r(h))$ borrowed from $\mathrm{UCB}_{13.1.2.1}$, $\bar{r}_{wf}(h) = \bar{r}_w(h) \times f$, with $f = 1.5 \times 0.85^k$ borrowed from $\mathrm{UCB}_{7.1.2.2}(h)$.

With a $q$ score 0.63, $\mathrm{UCB}_{7+13}$ yields a 6.83% improvement of the PAR2 score relative to the baseline Kissat4.0.2_MAB. According to DeepSeek-R1, this improvement is achieved by combining variance-aware reward adjustment with decay-based exploration control in the UCB selection, enhancing the balance between exploitation and exploration for heuristic switching.

In this case study, $\mathrm{UCB}_{7+13}$ is the first function generated by LLMs that achieves an improvement relative to Kissat4.0.2_MAB, all previous functions being worse. For example, $\mathrm{UCB}_{13.1.2.1}$ results in a 30.46% degradation in the PAR2 score relative to the baseline. According to

[2]https://github.com/arminbiere/kissat

IJCAI-ECAI 2026 PREPRINT

DeepSeek-R1, this degradation occurs because the entropy-guided mechanism excessively prioritizes exploration by incorporating variance into UCB scores, causing suboptimal heuristic selection when scores are close and disrupting the exploitation-exploration balance.

$\mathrm{UCB}_{7+13}$ is then combined with $\mathrm{UCB}_{11.3.3.2}$ to give $\mathrm{UCB}_{7+13+11}$.

$$\mathrm{UCB}_{7+13+11}(h) = \begin{cases} \bar{r}_{npwbfs}(h) + \mathrm{xpr1}(h) & h \text{ was just used for } k \\ & \text{consecutive restarts,} \\ \bar{r}_{npwb}(h) + \mathrm{xpr1}(h) & \text{otherwise} \end{cases} \quad (11)$$

The computation of $\bar{r}_{npwb}(h)$ and $\bar{r}_{npwbfs}(h)$ is done in several steps. In the first step, let $\mathrm{recentAVG5_h}$ denote the average of $r_n(h)$ in the last 5 restarts using $h$ and $\mathrm{oldAVG5_h}$ the average of $r_n(h)$ in the 5 restarts using $h$ just before the last 5 restarts using $h$. A trend factor $p_h = 1.0 + 0.3 \times (\mathrm{recentAVG5_h} - \mathrm{oldAVG5_h})/\mathrm{oldAVG5_h}$ is added:

$$\bar{r}_{np}(h) = \bar{r}_n(h) \times p_h \quad (12)$$

Let $w_n(h) = (1.0 + e^{-0.01 \times n_{total}} \times var(r_n(h)))$, similar as in $\mathrm{UCB}_{7+13}$.

$$\bar{r}_{npw}(h) = \bar{r}_{np}(h) \times w_n(h) \quad (13)$$

Let $\mathrm{recentAVG3}_h$ denote the average of $r_n(h)$ in the last three restarts using $h$ and $b_h = 1.0 + 0.2 \times (\mathrm{recentAVG3}_h - \bar{r}_{npw}(h))/\bar{r}_{npw}(h)$ be a bonus computed using the last three restarts using $h$. We have

$$\bar{r}_{npwb}(h) = \bar{r}_{npw}(h) \times b_h \quad (14)$$

Finally, let $f = 1.5 \times 0.85^k$ as in $\mathrm{UCB}_{7,1.2.2}(h)$ and $\mathrm{UCB}_{7+13}$, and $s = 1.0 + 0.1 \times ln(k+1)$. We have

$$\bar{r}_{npwbfs}(h) = \bar{r}_{npwb}(h) \times f \times s \quad (15)$$

With a $q$ score 0.73, $\mathrm{UCB}_{7+13+11}$, although complex, yields a 12.26% improvement w.r.t. Kissat4.0.2_MAB on $I$.

**Adaptive Meta-Learning**

After a round, AESAT calls DeepSeek-R1 to optimize the tips by analyzing the performance reasons of all individuals so far. For example, the following tips are added:

- Do not attempt to excessively adjust exploration rate parameters to avoid performance instability
- Deeply attempt dynamic reward function optimizations, especially adjustments based on the number of conflicts
- Avoid complex implementations; focus on simple, effective modifications
- Thoroughly test strategies that combine learning rates and rewards

## 4.3 Observations: LLM Logic vs. Human Intuition

The most significant finding of this case study is that AESAT discovered algorithmic strategies that diverge fundamentally from human expert intuitions. We highlight three counter-intuitive implementations:

**The Average-Subtraction Rolling Window:** $\mathrm{UCB}_{7+13+11}$ needs to compute the average of $k$ last rewards $r$. For this, it computes a rolling sum. A human usually computes this rolling sum by removing the oldest reward and adding the new one ($S_{new} = S_{old} - r_{n-k} + r_{new}$). However, $\mathrm{UCB}_{7+13+11}$ subtracts the average of the current window before adding the new value. This sounds as a self-correcting normalization factor that prevents reward saturation in long-running solver instances.

**Unorthodox Variance Computation:** When computing the variance $\mathrm{var(r)}$ of a reward $r$, a human would use the Welford method. However, the computation of $\mathrm{var(r)}$ by LLMs is very hard to understand and interpret in $\mathrm{UCB}_{7+11+13}$. Let $d$ be an intermediate value initialized to the first reward $r_1$, and $m$ denote the number of rewards so far. When a new reward $r_{m+1}$ arrives, LLMs update $d$ to

$$d + r_{m+1} + fabs(r_{m+1} - \frac{d}{m}) \times (\frac{d + r_{m+1}}{m+1} - \frac{d}{m}) \quad (16)$$

Let $\bar{r}$ denote the average of rewards so far, LLMs calls Equation (17) the variance $\mathrm{var(r)}$ of $r$:

$$\frac{d}{m+1} - \bar{r} \times \bar{r} \quad (17)$$

**Multi-Tiered Trend Analysis:** Human experts rarely implement nested trend factors $(p_h, b_h)$ due to the extreme difficulty of manual tuning. AESAT successfully balanced these overlapping feedback loops, which LLMs analyzes as "optimizing the balance between historical stability and rapid adaptation."

Note that LLMs use standard identifiers such as $\mathrm{variance}$ or $\mathrm{rollingSum}$ within the generated $C$ code, even when the underlying logic is fundamentally different. Without the granular focus provided by AESAT, these subtle but powerful deviations from classical arithmetic would likely remain hidden. Indeed, such insights would be significantly more difficult to uncover if AESAT were tasked with optimizing a solver's global architecture rather than its localized functions. By constraining the LLM's creative search to discrete components, AESAT allows human experts to audit, identify, and eventually adopt these novel machine-logic strategies.

## 4.4 Discussion: AESAT vs AutoSAT

In frameworks like AutoSAT [Sun *et al.*, 2024], the 20 initial functions generated in this study would have been discarded because they performed worse than the baseline. However, through its memetic architecture, AESAT recognized the latent potential in these worse functions. By applying LocalOptimization to refine the factors (like the boost factor $f$) and EvolutionaryUpdate to cross-pollinate ideas (like combining

IJCAI-ECAI 2026 PREPRINT

$t_7$ and $t_{13}$), AESAT navigated a fitness landscape that is inaccessible to simple hill-climbing or single-parent approaches.

## 5 Experiments

This section provides a rigorous quantitative evaluation of AESAT. We first demonstrate its ability to discover high performance SAT solvers across multiple competition datasets, by evaluating four functions it generated. Then, we present an ablation study to isolate the impact of different components of AESAT and the impact of the *counter-intuitive heuristics* discovered by the LLMs.

**Benchmark Datasets**: We evaluate AESAT using the official datasets from the SAT Competitions 2022–2025. Each year's dataset comprises 400 instances originating from diverse industrial and theoretical domains.

**Experimental Setup**: All experiments were conducted on a server equipped with an AMD EPYC 9654 processor unless otherwise stated. The training set comprise 60 instances randomly selected from the SAT Competition 2024 dataset. The parameters were initialized as follows: $populationSize = 20$, $roundSize = 5$, $localOptimLimit = 5$, $nbNeighbors = 3$, $timeOut = 5000s$ as in the SAT competitions, $\alpha = 0.4$, $\beta = 0.4$, $\gamma = 0.2$. The source code of AESAT and Kissat4.0.2_MAB with all tested UCB functions, as well as the set of 60 training instances, are available in the **supplementary materials**[3].

### 5.1 Four Functions Generated by AESAT

Each call to AESAT returns a new function in 30–40 hours for roughly 0.50 USD, representing a significant cost reduction over the 20,000 USD SATLUTION [Yu *et al.*, 2025]. Furthermore, 46% of the UCB functions surpassing the baseline on training set $I$ maintained their superiority on the full datasets.

To demonstrate the framework's versatility, we evaluate the following four functions returned by AESAT. The first three provide optimized implementations of the UCB selection logic in `restart_MAB`, as illustrated in Section 4.2. The fourth replaces `kissat_bump_score_increment`, demonstrating that AESAT's capabilities extend to various components of the solver.

**$\mathbf{UCB}_{7+13+11}$:** Defined in Equation (11).

**$\mathbf{UCB}_{sc2025}$:** Defined in Equation (18), where $r(h)$ is defined by Equation (2), $m$ is a momentum coefficient initialized to 1. Let $\{R_1, \ldots, R_{10}\}$ be set of the 10 last restarts, $h_i$ ($1 \leq i \leq 10$) be the heuristic used in restart $R_i$, $\text{avg10}(\bar{r}) = \frac{1}{10}\sum_{i=1}^{10} \bar{r}(h_i)$. After a restart using $h$, if $\bar{r}(h) > \text{avg10}(\bar{r})$, $m$ is updated to $m \times 1.1$, otherwise $m$ is updated to $m \times 0.9$.

**$\mathbf{UCB}_{xpr2}$:** Defined in Equation (19), where xpr2 is defined in Equation (20).

**BumpInc:** In the VSIDS heuristic, each variable's score is initialized to 0. Following every conflict, the scores of involved variables are incremented by a value, *scinc*, which starts at 1 and is progressively scaled by a factor of $\frac{1}{1-decay}$, where $decay$ is a fixed constant in the baseline **Kissat 4.0.2**. The **kissat_bump_score_increment** function governs the evolution of this $scinc$ value.

AESAT was applied to optimize this component in **Kissat 4.0.2**, yielding a new implementation of **kissat_bump_score_increment** defined in Equations (21)–(23). The resulting solver is called **BumpInc**.

$$\text{UCB}_{sc2025}(h) = \bar{r}(h) + \sqrt{\frac{4 \times \ln(n_{total})}{m \times n_{total} \times n_h}} \tag{18}$$

$$\text{UCB}_{xpr2}(h) = \bar{r}(h) + \text{xpr2}(h) \tag{19}$$

$$\text{xpr2}(h) = \sqrt{\frac{4 \times sqrt(\ln(n_{\text{total}})) \times pow(n_{total}, 0.25)}{n_h}} \tag{20}$$

Let scinc denote the current score increment value also starts at 1, $d_0$ denote the original decay of Kissat4.0.2, MAX_SCORE the upper bound of the VSIDS score, and ratio = scinc/MAX_SCORE. The dynamic decay $d_{dyn}$ is defined as:

$$d_{dyn} = \begin{cases} d_0 \times (0.5 + \text{ratio}) & \text{if ratio} < 0.5 \\ d_0 \times (0.5 + 0.3 \times \tanh(\text{ratio})) & \text{otherwise} \end{cases} \tag{21}$$

Then, a compensation-adjusted growth factor $g$ is defined in Eq. (22) to replace the original constant factor $\frac{1}{1-decay}$.

$$g = \frac{1}{1 - d_{dyn} \times (\ln(2 - d_{dyn}) + 1 + 0.15 \times d_{dyn})} \tag{22}$$

Finally, the new score increment $\text{scinc}_{new}$ is calculated in Eq. (23) using $g$ after every conflict.

$$\text{scinc}_{new} = \text{scinc} \times g \tag{23}$$

Note that the new factor $g$ is adaptive, allowing to increase $scinc$ more rapidly when it is small, while the original factor $\frac{1}{1-decay}$ is a fixed constant.

Table 1 shows the standard performance metrics as in SAT competitions: the number of (ALL, SAT, UNSAT) instances solved by **Kissat4.0.2_MAB** with $\mathbf{UCB}_{7+13+11}$, $\mathbf{UCB}_{sc2025}$, $\mathbf{UCB}_{xpr2}$, and **Kissat4.0.2** with **BumpInc**, respectively, instead of the original heuristic functions, and their mean PAR2 score, against the baselines **Kissat4.0.2_MAB** (**Base_MAB** in the table) and **Kissat4.0.2**. The first three functions globally perform better than the baselines. The fourth function **BumpInc** solves more unsatisfiable instances than the baselines. Especially, **Kissat4.0.2_MAB with $\mathbf{UCB}_{sc2025}$** is the solver **AE-Kissat-MAB** that won the main track and the SAT track of the 2025 SAT competition with a wide margin. Additionally, **Kissat4.0.2** with **BumpInc** is the solver **AE-Kissat-Bump** that won the bronze medal in the UNSAT track of the 2025 SAT Competition, solving more unsatisfiable instances than Kissat-public, the newest version of Kissat participating in the competition and winning the silver medal in the main track (SAT+UNSAT).

[3]https://github.com/FSQH-dh/AESAT

IJCAI-ECAI 2026 PREPRINT

PRELIMINARY PREPRINT VERSION – DO NOT CITE
This is not the version of record. The final version will appear in the Proceedings of IJCAI-ECAI 2026.

| Solver | 2025 | | | | 2024 | | | | 2023 | | | | 2022 | | | | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Total | PAR2 | SAT | UNSAT | Total | PAR2 | SAT | UNSAT | Total | PAR2 | SAT | UNSAT | Total | PAR2 | SAT | UNSAT | |
| $UCB_{sc2025}$ | **341** | **1883** | **176** | 165 | 330 | 2056 | 165 | 165 | 293 | 2952 | 123 | 170 | **323** | **2324** | **165** | 158 | 1287 |
| $UCB_{xpr2}$ | 340 | 1932 | 173 | **167** | **335** | **1963** | **170** | 165 | **296** | 2914 | 124 | **172** | 320 | 2375 | 164 | 156 | **1291** |
| $UCB_{7+13+11}$ | 340 | 1884 | 173 | **167** | 329 | 1986 | 164 | 165 | 290 | 2952 | 120 | 170 | 322 | 2352 | **165** | 157 | 1281 |
| BumpInc | 330 | 2151 | 163 | **167** | 330 | 2066 | 160 | **170** | **296** | 2846 | 124 | **172** | 316 | 2475 | 159 | 157 | 1272 |
| Base_MAB | 335 | 2025 | 171 | 164 | 326 | 2139 | 164 | 162 | 291 | 2995 | 122 | 169 | 317 | 2480 | **165** | 152 | 1269 |
| Kissat4.0.2 | 330 | 2229 | 165 | 165 | 328 | 2119 | 159 | 169 | **296** | **2818** | **125** | 171 | 318 | 2451 | 159 | **159** | 1272 |

Table 1: The number of (ALL, SAT, UNSAT) instances solved and PAR2 score on SAT Competition Benchmarks (2022–2025)

## 5.2 Ablation Study

We first present ablation studies to isolate the performance impact of each AESAT component. Recognizing that AESAT frequently identifies strategies at odds with expert intuition, we also perform a comparative analysis to characterize the differences between LLM-synthesized search logic and conventional human logic.

**Note:** The experiment in Figure 1 was conducted on the complete SAT Competition 2024 dataset (400 instances) using a different server configuration from Table 1. The experiments in Figure 2 and in Figure 3 were conducted on the training set $I$ of 60 instances described in the Experimental Setup. All experiments in this ablation study are performed by optimizing the MAB function.

**Effectiveness of PromptSelfOptimize.** To evaluate the contribution of the PromptSelfOptimize mechanism (Algorithm 5), we track the performance of 93 UCB functions returned across successive runs of AESAT. Figure 1 shows the number of instances solved by each returned function on the 2024 dataset, with the baseline Kissat4.0.2_MAB solving 318 instances. A linear regression line is fitted using ordinary least squares (OLS), and it exhibits a positive slope, indicating that the later runs of AESAT tend to discover better performing heuristic functions. This upward trend is attributed to the PromptSelfOptimize mechanism, which periodically refines the prompt guidelines (*tips*) based on historical successes and failures, effectively steering the LLM's search toward more fruitful algorithmic directions over time.

**Effectiveness of EvolutionaryUpdate.** To isolate the impact of the crossover operation within EvolutionaryUpdate (Algorithm 3), we analyze how locally refined functions are transformed into hybrid variants. Figure 2 contrasts the best-performing locally optimized function against the top hybrid function for each execution of AESAT on the training set $I$, illustrating their relative improvements over the baseline. Notably, 82% of the best locally optimized functions already surpass the baseline, validating the standalone efficacy of LocalOptimization. Crucially, in 64% of the runs, the best hybrid function further improves upon its locally optimized counterpart, with the peak global hybrid yielding a 36.6% improvement relative to the baseline. This trend confirms that EvolutionaryUpdate introduces vital optimization and diversification beyond what local refinement alone can achieve.

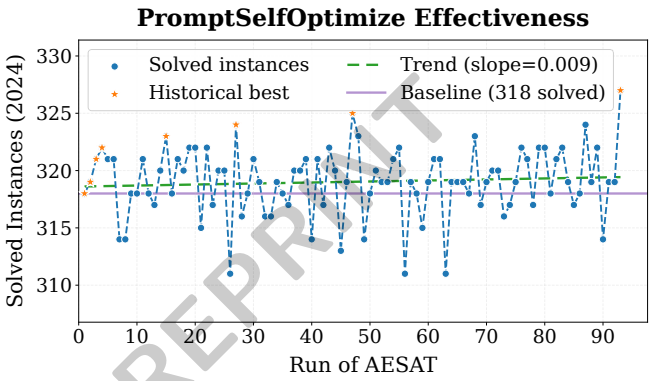


Figure 1: Number of instances solved (y-axis) by each of 93 UCB functions returned across successive runs of AESAT (x-axis) on the 2024 dataset. The dashed green line is the linear regression with a positive slope. Orange stars mark historical best solutions. The purple line indicates the baseline (318 instances solved by Kissat4.0.2_MAB).

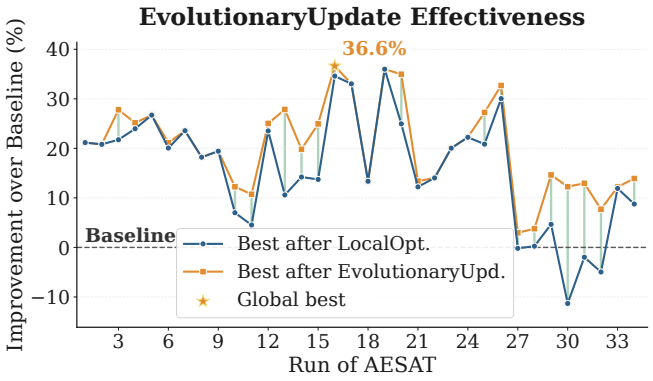


Figure 2: Relative improvement over baseline (%, y-axis) of the best locally optimized function (blue) and the best hybrid function after crossover (orange) in each run of AESAT (x-axis) on training set $I$. Green vertical bars indicate the improvement gained by crossover. The gold star marks the global best hybrid at 36.6%.

**Effectiveness of LocalOptimization.** To isolate the contribution of LocalOptimization (Algorithm 2), we evaluate an ablated variant of AESAT that bypasses the local refinement phase entirely: 20 initial ideas are generated and, post-evaluation, immediately passed to the EvolutionaryUpdate stage. Figure 3 illustrates the results across 35 independent

IJCAI-ECAI 2026 PREPRINT

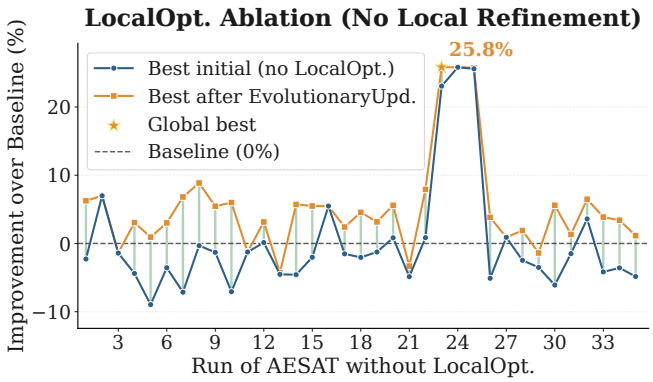


Figure 3: Relative improvement over baseline (%, y-axis) of the best initial idea without local refinement (blue) and the best hybrid function after crossover (orange) in each run of AESAT (x-axis) without LocalOptimization on training set $I$. Green vertical bars indicate the improvement gained by crossover. The gold star marks the global best hybrid.

runs of this variant on the training set $I$. Without LocalOptimization, only 28.6% of the best initial ideas outperform the baseline—a stark contrast to the 82% success rate observed in the full pipeline (Figure 2). Despite this initial deficit, EvolutionaryUpdate successfully generates hybrid functions that improve upon the best initial seed in 82.9% of the runs, underscoring its robust complementary efficacy. However, omitting local refinement caps the maximum global improvement relative to the baseline at just 25.8%, compared to the 36.6% global improvement achieved when LocalOptimization primes the evolutionary search (Figure 2).

**Comparative Ablation: LLM Logic vs. Human Intuition.** As the case study in Section 4 suggests that AESAT discovers strategies fundamentally different from human intuition, we conduct a comparative ablation study on $\text{UCB}_{7+13+11}$. By replacing the machine logic observed in Section 4.3 with usual human-expert implementations, we aim to determine whether these counter-intuitive discoveries provide a measurable performance advantage over traditional heuristics.

**UCBrolling**$_{7+13+11}$ : The rolling sum of rewards in the last $k$ restarts is done as usually, i.e., when a new reward arrives, subtract the oldest reward and add the new one.

**UCBvar**$_{7+13+11}$ : The variance of rewards is computed using the Welford method.

**UCBrecency**$_{7+13+11}$ : The last recent performance bonus $b_h$ is discarded.

The results shown in Table 2 on the 2025 dataset are striking: reverting to human practices for variance, rolling sums and trend analysis degrades performance significantly. In fact, using standard human implementation methods for these specific components caused the solver to perform worse than the LLM version (e.g., PAR2 increasing from 1884 to 2379). This suggests that the LLM-discovered heuristics are not just faster, but capture subtle search-space dynamics that standard arithmetic fails to address.

| 2025 | Total | PAR-2 | SAT | UNSAT |
|---|---|---|---|---|
| $\text{UCB}_{7+13+11}$ | **340** | **1884** | **173** | **167** |
| $\text{UCBrolling}_{7+13+11}$ | 324 | 2344 | 172 | 152 |
| $\text{UCBvar}_{7+13+11}$ | 323 | 2379 | 168 | 155 |
| $\text{UCBrecency}_{7+13+11}$ | 335 | 2003 | 172 | 163 |

Table 2: $\text{UCB}_{7+13+11}$ vs Human Variants

## 6 Conclusion

In this paper, we introduced AESAT, a multi-agent, auto-evolving framework designed to optimize the delicate and complex heuristics of modern SAT solvers. By a memetic evolutionary loop, AESAT moves beyond traditional black-box algorithm configuration, enabling the discovery of entirely new and efficient algorithmic logic. The ablation study shows that every component of AESAT contributes to the performance of AESAT.

Our empirical evaluation demonstrates that AESAT can achieve quickly significant performance gains that could require years of effort from human teams to obtain. More crucially, our analysis reveals that AESAT achieves these results by evolving machine logic that often contradicts human intuition. From the average-subtraction rolling sum to unorthodox variance estimations, AESAT successfully explores a design space that has remained largely untapped by human experts. The success of **UCB**$_{sc2025}$ and **BumpInc** in the 2025 SAT Competition serves as a definitive validation of this approach.

AESAT represents a paradigm shift from manual tuning to automated discovery, ensuring that even complex, multi-layered ideas—which are traditionally prone to failure during initial implementation—can be refined into competitive heuristics. This reduces the barrier to entry for developing high-performance solvers.

IJCAI-ECAI 2026 PREPRINT

## Acknowledgments

This work was partially supported by National Natural Science Foundation of China (NSFC) under Grant 62402164 and by the French National Research Agency (ANR) under the Chair MASSAL'IA (ANR-19-CHIA-0013-01), co-funded by the French electricity distribution network operator Enedis. It was granted access to HPC resources of "Plateforme MatriCS" within University of Picardie Jules Verne, co-funded by the European Union with the European Regional Development Fund (FEDER) and the Hauts-De-France Regional Council. And it was supported by the Hubei provincial key laboratory of green intelligent computing power network, China.

## References

[Biere *et al.*, 2020] Armin Biere, Amélie Fleury, L. Heisinger, and B. Kiesl. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020. In *SAT Competition 2020-Solver and Benchmark Descriptions*, volume B-2020-1, pages 16–19, Helsinki, Finland, 2020. Department of Computer Science, University of Helsinki.

[Biere *et al.*, 2021] Armin Biere, Marijn Heule, Hans Van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability (Second Edition)*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, 2021.

[Biere *et al.*, 2024] Armin Biere, , Tobias Faller, Katalin Fazekas†, Mathias Fleury, Nils Froleyks, and Florian Pollitt. Cadical, gimsatul, isasat and kissat entering the sat competition 2024. In *Proceedings of SAT Competition 2024 : Solver, Benchmark and Proof Checker Descriptions*, pages 8–10, 2024.

[Chan *et al.*, 2023] Chi-Min Chan, Wei-Sheng Lee, Jie-Jhih Hsiang, Ting-Hao Chen, Yi-Fang Chen, Chien-Yi Lee, and Hung-yi Lee. ChatEval: Towards Better LLM-as-a-Judge through Role-Playing and Multi-Agent Debate. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11330–11350, Singapore, 2023. Association for Computational Linguistics.

[Chen and Zhang, 2022] Zhihan Chen and X. Zhang. Easysat: A simple cdcl sat solver. https://github.com/shaowei-cai-group/EasySAT, 2022. Accessed: 2025-10-12.

[Cherif *et al.*, 2021] Sami Cherif Cherif, Djamal Habet, and Terrioux Cyril. Combining VSIDS and CHB Using Restarts in SAT. In *CP 2021*, pages 20:1—-20:19. LIPICS, 2021.

[Cook, 1971] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 1971. ACM.

[Da Ros *et al.*, 2025] Francesca Da Ros, Michael Soprano, Luca Di Gaspero, and Kevin Roitero. Large Language Models for Combinatorial Optimization: A Systematic Review. arXiv preprint arXiv:2507.03637, 2025.

[Guo *et al.*, 2024] Qingyan Guo, Rui Wang, Junliang Guo, Boyu Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. EvoPrompt: Connecting LLMs with Evolutionary Algorithms Yields Powerful Prompt Optimizers. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*, Vienna, Austria, 2024.

[Liang *et al.*, 2016] Jia Hui Liang, Chung-Kuan Oh, Zhihao Wu, Hongce Li, Satrajit Mitra, and Sharad Malik. Learning rate based branching heuristic for sat solvers. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT 2016)*, volume 9710 of *Lecture Notes in Computer Science*, pages 123–140. Springer, 2016.

[Liu *et al.*, 2024a] J. Liu, J. Zhang, and Y. Sun. Kissat_mabdc in sat competition 2024. In *Proceedings of SAT Competition 2024 : Solver, Benchmark and Proof Checker Descriptions*, page page 20, 2024.

[Liu *et al.*, 2024b] Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. Large language models as evolutionary optimizers. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2024.

[Marques-Silva, 2008] Joao P. Marques-Silva. Practical Applications of Boolean Satisfiability. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 301–306, San Jose, CA, 2008. IEEE Press.

[Moskewicz *et al.*, 2001] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th Design Automation Conference (DAC 2001)*, pages 530–535, Las Vegas, NV, USA, 2001. ACM.

[Naveed *et al.*, 2023] Hina Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, et al. A Comprehensive Overview of Large Language Models. *Artificial Intelligence Review*, pages 1–69, 2023.

[Romera-Paredes *et al.*, 2023] Bernardino Romera-Paredes, Alhussein Fawzi, et al. Mathematical Discoveries from Program Search with Large Language Models. *Nature*, 624(7990):303–308, 2023.

[Sun *et al.*, 2024] Yisong Sun, Xing Zhang, Shaowei Huang, Shaofei Cai, Bei-Zhen Zhang, and Ke Wei. AutoSAT: Automatically Optimize SAT Solvers via Large Language Models. arXiv preprint arXiv:2402.10705, 2024.

[Vizel *et al.*, 2015] Yakir Vizel, Georg Weissenbacher, and Sharad Malik. Boolean Satisfiability Solvers and Their Applications in Model Checking. *Proceedings of the IEEE*, 103(11):2021–2035, 2015.

[Wu *et al.*, 2024] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

[Ye *et al.*, 2024] Haoran Ye, Jiyuan Wang, Zonggan Cao, Francesco Berto, Chen Hua, H. Kim, J. Park, and G. Song. ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada, 2024.

[Yu *et al.*, 2025] Cunxi Yu, Rongjian Liang, Chia-Tung Ho, and Haoxing Ren. Autonomous code evolution meets np-completeness. *arXiv preprint arXiv:2509.07367*, 2025.

IJCAI-ECAI 2026 PREPRINT