

DiLA: Enhancing LLM Tool Learning with Differential Logic Layer

Anonymous submission

Abstract

Logical reasoning remains a significant challenge for large language models (LLMs), particularly in tasks involving complex constraint satisfaction such as Boolean satisfiability (SAT) and graph coloring. Existing approaches—ranging from pure prompting-based reasoning to solver-aided frameworks—either suffer from unfaithful reasoning or face scalability bottlenecks due to exponential search spaces in symbolic solvers. In this paper, we present **DiLA** (Differential Logic Layer-Aided Language Modeling), a novel framework that integrates a differentiable logic layer into LLMs to jointly leverage linguistic understanding and gradient-based logical refinement. DiLA first translates natural language problems into SAT specifications and generates an initial LLM-informed variable assignment, then iteratively refines it through a logic layer implementing differentiable MaxSAT optimization. This synergy enables efficient reasoning grounded in formal logic while maintaining semantic awareness. Comprehensive experiments across logical deduction, SAT, and graph coloring benchmarks demonstrate that DiLA achieves 100% accuracy with up to $65\times$ runtime speedup over solver-aided methods such as SATLM. On industrial-scale benchmarks where state-of-the-art solvers (Z3, Kissat) fail within 10,000 seconds, DiLA successfully converges in under 300 seconds, illustrating its robustness in large and highly constrained settings. Furthermore, on the Natural Language Constraint Reasoning benchmark, DiLA reaches 87% end-to-end success rate, outperforming both SATLM and pure LLM baselines by large margins. Our code is available at <https://anonymous.4open.science/r/DiLA-33DE/>.

Introduction

Recently, a significant research thrust has been on leveraging large language models (LLMs) for reasoning and planning, with numerous efforts aimed at augmenting their reasoning capabilities. These endeavors include text-based temporal reasoning (Xiong et al. 2024), logic feedback-enhanced alignment methods (Nguyen, Fungwacharakorn, and Satoh 2023), and prompt-based methods such as the chain of thoughts (CoT) (Wang et al. 2023a) or a simple directive like “Let’s think step by step” (Lightman et al. 2023). To further bolster the reasoning ability of LLMs, an AlphaZero-like tree-search learning framework has been introduced. This framework demonstrates how tree-search, coupled with a learned value function, can guide LLMs’ decoding and enhance their reasoning abilities (Feng et al. 2023b). Despite the strides made by LLMs in achieving human-like reasoning abilities, they still encounter challenges when confronted with complex plannings (Valmeekam et al. 2022). Occasionally, LLMs exhibit unfaithful reasoning, leading to derived conclusions that do not consistently follow the

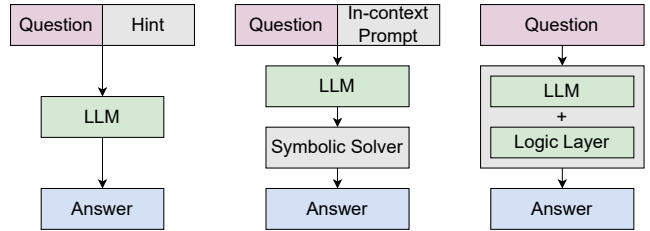


Figure 1: Illustration of CoT (left), solver-aided approach (middle), and our logic layer-aided language modeling approach (right).

previously generated reasoning chain in practical applications (Pan et al. 2023).

To this end, a growing trend in this area involves augmenting LLMs with access to external engines, i.e., utilizing LLMs to first parse natural language logical questions into symbolic representations and subsequently employ external solvers to generate answers based on these representations. To enhance parsing accuracy, LoGiPT (Feng et al. 2023a) has been proposed to directly emulate the reasoning processes and mitigate the parsing errors by learning to strict adherence to solver syntax and grammar. It is fine-tuned on a constructed instruction-tuning dataset derived from revealing and refining the invisible reasoning process of deductive solvers. An alternative approach for improving reasoning capabilities involves SATLM, a new satisfiability-aided modeling approach (Ye X 2023), in which an LLM is used to generate a declarative task specification with few-shot prompting and offload the actual reasoning task to an off-the-shelf theorem prover, i.e., Z3 solver (De Moura and Bjørner 2008). This method has demonstrated superiority over program-aided Language Models (Gao et al. 2023) by achieving a 23% improvement on the GSM arithmetic reasoning dataset, establishing a new state-of-the-art.

Despite their impressive performance on various benchmark tests, SATLM (Ye X 2023) and its solver-augmented successors can not deal with reasoning and planning problems in practical scenarios. Particularly, these real problems often state a set of premises and complex constraints and require a sophisticated search process to find the optimal solution, which is still challenging even for modern solvers. More specifically, even the state-of-the-art symbolic solvers, such as Z3 (De Moura and Bjørner 2008) and Kissat (Biere and Fleury 2022), encounter significant bottlenecks when addressing formal verification problems such as electronic circuit verification involving tree/cyclic circuit structures (Shi et al. 2023) and job-shop scheduling prob-

lems (Li et al. 2022) with complex graph connections. For instance, the Kissat solver requires days or even weeks to process a vanilla circuit verification problem with arithmetic circuit modules including multipliers or multiply-add circuits, yet fails to produce a solution. This highlights the limitations of solver-augmented LLM approaches which solely rely on symbolic solvers to tackle logical reasoning problems in reality.

In this paper, we propose DiLA, a novel tool-learning approach for Large Language Models, designed to enhance their logical reasoning capabilities through the integration of an additional logic layer. Unlike existing methods that either rely on in-context prompting for step-by-step reasoning (see Figure 1(left)) or entirely offload reasoning to external solvers (see Figure 1(middle)), our approach offers a third option: enhancing LLMs’ reasoning ability by incorporating a differential logic layer (see Figure 1(right)). Specifically, DiLA leverages the LLM to parse and comprehend the problem description, generate an initial solution based on its language understanding, and then iteratively refine this solution through forward and backward passes of a network layer that embeds first-order logic constraints into its architecture. In this way, DiLA overcomes the limitations of traditional solvers by directly performing reasoning within the framework of layer-augmented LLMs. Our contributions are summarized as follows:

- We introduce a novel tool-learning approach for LLMs, DiLA, which synergistically integrates a differential logic layer into LLMs, effectively bridging the gap between natural language understanding and symbolic reasoning capabilities.
- Leveraging SAT encoding as a bridge, DiLA successfully translates natural language reasoning problems into satisfiability problems, enabling it to tackle a range of reasoning problems, such as SAT and GCP.
- We propose a novel neural network layer, termed the logic layer, which differentiates symbolic problems and iteratively searches for solutions through forward and backward propagation of a network layer, thereby circumventing the limitations of off-the-shelf solvers.

We evaluate the performance on three constraint satisfaction problems: logical deduction, Boolean satisfiability, and graph coloring. Our analysis yields two key findings: firstly, on simple problem instances, DiLA boosts the inference accuracy of LLMs to 100% and consistently outperforms solver-aided approaches with improved runtime. Secondly, for real-world problems that current solvers struggle with, DiLA showcases robustness and remarkable efficiency in handling these complex test cases, thereby opening up opportunities for further real-world applications.

Preliminary and Related Works

Preliminary

In general, Boolean formulae are represented in Conjunctive Normal Form (CNF) as a conjunction of clauses, where a clause is a disjunction of literals and a literal denotes either a variable or its negation. Each variable can be assigned a logic value, either 0 or 1. Any general Boolean problem can be represented as a CNF formula model. An SAT solver either finds an assignment such that CNF is satisfied or proves that no such assignment exists, i.e., UNSAT. Modern SAT solvers are based on the conflict-driven-clause-learning (CDCL) algorithm, and they work as a basic engine for many applications. SAT plays an important role in data

mining applications, e.g., Maximal Frequent Subgraph Mining (Liu and Li 2022), Graph Coloring (Velev 2007), Sequence Mining (Jabbour, Sais, and Salhi 2013).

Related Works

Logic Reasoning in LLMs. Prior approaches to NL-based reasoning with LLMs can be broadly categorized into two groups. One is *in-context learning* approaches that design special prompts to elicit LLMs’ step-by-step reasoning capabilities. Typical methods include chain-of-thought prompting (Wang et al. 2023a) that generates a sequence of reasoning steps before the final answer and the least-to-most prompting (Zhou et al. 2022) that breaks the problem down into simpler components that can be solved sequentially. Both the above approaches perform reasoning directly over natural language (NL), providing greater flexibility than symbolic-based reasoning. However, the intrinsic complexity and ambiguity of NL also bring undesired issues such as unfaithful reasoning and hallucinations. The other is *tool-augmented* approaches that only require LLMs to parse the problem specification out of the language description accurately and leverage off-the-shelf automated tools to derive the final answer, as exemplified by SATLM (Ye X 2023) and LOGIC-LM (Pan et al. 2023). The tool-augmented approaches guarantee the correctness of the answer with respect to the parsed specification and avoid planning errors in the solving process. However, the performance of such tool-augmented methods highly relies on external tools and can not deal with many real problems due to the deficiency of backbone solvers.

Neural Symbolic Reasoning. Deep neural networks have made remarkable progress in various domains, but their ability to handle logical reasoning tasks remains uncertain. One potential approach is to encode logical constraints that are essential for certain inference tasks symbolically, making them available to the model either during training or inference. SATNet (Wang et al. 2019), for instance, introduces a differentiable MaxSAT solver layer into its network architecture to learn the logical structure of problems through data-driven learning. Their approximation is based on a coordinate descent approach for solving the semidefinite program (SDP) relaxation of the MaxSAT problem. SATNet does not assume that the logical structure of the problem is given, and instead attempts to learn it. While SATNet achieves high accuracy in solving Sudoku problems, it cannot be generalized to regular SAT problems as it relies on a specific number of accurate assignments as inputs and is only applicable to solving logic problems with constant constraints after training. In the pursuit of neuro-symbolic learning, SMTLayer (Wang et al. 2023b) incorporates a satisfiability modulo theories (SMT) solver into a DNN layer, thereby embedding problem-specific theories into DNN architectures through data-driven training. Compared to SATNet, SMTLayer demonstrates superior accuracy in logical reasoning tasks and a reduced need for training data. However, SMTLayer still requires a certain amount of training data and relies on off-the-shelf SMT solvers for reasoning, making it unsuitable for industrial problem-solving. Similar to the aforementioned neuro-symbolic approach, DiLA also aims to encode logical constraints in the network layer, making them available to the model during forward and backward passes.

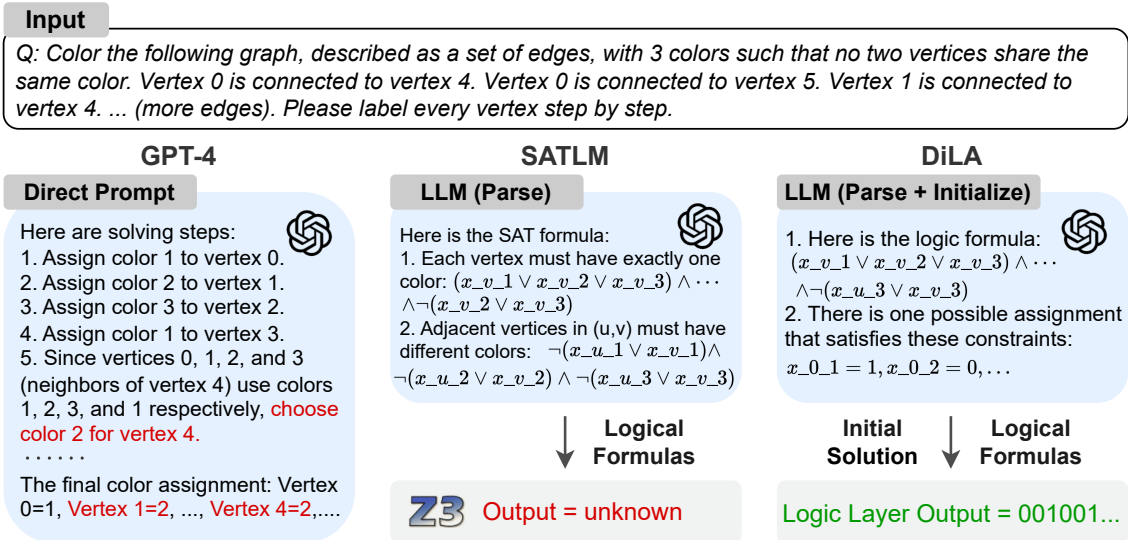


Figure 2: Exemplar comparison of solving graph coloring problems by different approaches. Direct prompts by GPT-4 make errors when generating the color assignment step by step; SATLM, based on the Z3 symbolic solver, cannot solve parsed constraints and outputs unknown (both errors are highlighted in red). In contrast, DiLA can generate the correct answer by combining the strengths of LLMs and the differential logic layer.

Motivation

This paper explores the potential of language understanding and logical reasoning capabilities in LLMs. Traditionally, the prevailing approaches have either relied solely on LLMs for step-by-step reasoning (Wang et al. 2023a; Zhou et al. 2022) or offloaded reasoning tasks to off-the-shelf solvers (Olausson et al. 2023; Ye X 2023). However, we propose a third approach, as these two extremes either underutilize or over-rely on LLMs during reasoning. Specifically, DiLA leverages the powerful understanding abilities of LLMs to extract logical formulas and generate possible solutions. By utilizing LLMs, such as GPT-4, as a solution generation engine, we can produce an initial solution based on input semantic constraints, laying the groundwork for further refinement.

The second motivation stems from an analysis of current solver applications in real-world scenarios, characterized by two distinctive features: (1) The large scale of reasoning problems in reality, leading to rapid degradation in the performance of heuristic-based solvers due to the exponential expansion of the search space; (2) The formidable challenge presented by the complex structure of logic formulas, often requiring weeks or even months to resolve using current solvers. Both characteristics significantly limit the effectiveness of a solver-augmented LLM in logic reasoning.

Consequently, our objective is to identify a synergistic approach that combines the strengths of LLMs and a differential logic layer, thereby circumventing the limitations of traditional solvers. This approach leverages the LLM’s capacity to comprehend logical formulas while concurrently utilizing the logic layer’s refinement abilities to achieve accurate solutions. Figure 2 illustrates a comparison between CoT, SATLM, and our proposed DiLA. The LLM alone may introduce logical flaws during step-by-step inference, such as assigning the same color to vertex 1 and vertex 4 despite their edge connection, and SATLM may struggle with complex reasoning problems due to its backbone solver’s limitations. In contrast, DiLA produces accurate answers through

the collaboration of the LLM and the differential logic layer. Specifically, DiLA first uses the LLM to parse a natural language input into logic constraints and generate an initial solution based on its semantic understanding, then employs the differential logic layer to refine this initial solution.

Differential Logic Layer-Aided LLMs

Overview

In this section, we present DiLA, which augments LLM with the ability of logical reasoning by incorporating a differential logic layer. More specifically, DiLA addresses the challenge of using LLMs to tackle canonical reasoning tasks expressed in natural language. These tasks typically involve presenting a set of premises and constraints, prompting questions that necessitate intricate deductive reasoning over the provided inputs, which remains a formidable challenge even for contemporary LLMs (Valmeekam et al. 2022).

The general procedure for solving natural language reasoning tasks with DiLA can be conceptualized in three distinct steps: parsing, initialization, and refinement (as illustrated in Figure 3). Given a natural language input that describes both the propositional constraints ϕ and the question Q , we first parse this input into a SAT specification using LLMs (*step 1*), thereby obtaining a formal description of the constraints and variables. Next, we leverage the LLM’s natural language understanding to generate an initial variable assignment (*step 2*). Since this initial solution may only partially satisfy the propositional constraints, DiLA iteratively refines it through a differential logic layer that encodes all of the logical formulas (*step 3*), ultimately yielding a more accurate solution.

Problem Formulator and Initialization

Intuitively, LLMs may struggle with directly solving complex reasoning problems. However, they have demonstrated a notable ability to comprehend textual inputs and translate them into formal programs, such as mathematical equa-

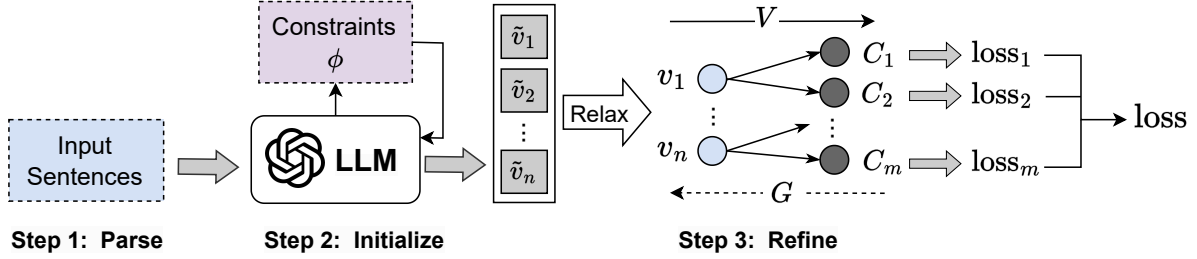


Figure 3: The overall flow of DiLA.

tions (He-Yueya et al. 2023) or satisfiability modulo (Ye X 2023). Notably, the SAT problem can serve as a versatile intermediate step for solving a broad range of constraint satisfaction problems, provided they can be expressed using Boolean variables. Specifically, problem instances from NP-complete domains, such as Graph Coloring and Set Cover, can be seamlessly encoded into SAT problem specifications, thereby allowing for efficient solutions via SAT algorithms (Stechly, Valmeekam, and Kambhampati 2024; Liu et al. 2023). Therefore, we harness SAT encoding as a general bridge to tackle these constraint satisfaction problems in practical settings.

Specifically, given a problem description in natural language, DiLA prompts an LLM with detailed instructions to generate the SAT specification, which includes a set of premises and constraints. Typically, the SAT specification here involves conjunctive normal forms (CNFs), denoted as $\phi(v_1, \dots, v_n)$, which is a conjunction of clauses (constraints) C . Formally, rules are written in the conjunctive form of clauses $C_1 \wedge C_2 \cdots \wedge C_m$, which each C_j is a constraint. The total rule is satisfied if and only if all of the clauses C_1, C_2, \dots, C_m are simultaneously True. Each clause represents a disjunction of literals, where a literal is either a propositional variable v_i or its complement $\neg v_i$, for example, $v_1 \vee \neg v_2$. In other words, if at least one literal in a clause is True, this clause would also be True. Variables can be assigned logic values, either 1 or -1 , representing True or False, respectively¹.

Aside from problem formulation, leveraging an LLM can be a valuable strategy to generate an initial solution. We observe that, after generating the SAT specification, we can prompt LLMs, like GPT-4 (Achiam et al. 2023) or Llama-3 (Meta 2024), to produce a potential solution or a set of possible solutions. This can be achieved by framing the problem as a natural language query, such as ‘‘What is the logical solution based on the premises?’’ or ‘‘Can you provide a possible answer from these given constraints?’’. The LLM’s response can then serve as a starting point for further refinement and validation, allowing us to build upon its output and iteratively improve the solution through logical analysis and reasoning. By harnessing the LLM’s ability to understand semantic constraints, we can tap into its potential to facilitate the initial solution-finding process and accelerate solving progress towards a well-reasoned answer.

From SAT to Differential MaxSAT

Traditionally, a SAT solver, such as Z3 and Kissat, is leveraged to determine a satisfying assignment for the given con-

¹In other works, they may claim the logic value of each literal is 0 or 1. It should be noted that the two claims are equal under simple mathematical transformations.

straint formula ϕ . Contemporary SAT solvers are founded on the CDCL algorithm, which excels in its ability to learn from conflicts and use that conflict knowledge to prune the branch-and-bound search space more effectively. However, existing CDCL-based SAT solvers still suffer from exponential searching space and are unable to correct errors through a learning-from-mistakes system, resulting in an infinite loop in solving complex SAT problems (Shi et al. 2023).

In this study, when an NL reasoning problem is fed into DiLA, our goal is to determine its solution that can satisfy all logical constraints. To achieve this, we might incorporate a solver as an additional logic reasoner for the LLM, as in SATLM (Ye X 2023) or LOGIC-LM (Pan et al. 2023). However, the exponential search complexity inherent in heuristic-based SAT solvers poses a significant challenge, limiting their effectiveness when dealing with complex real-world problems. Therefore, a key issue is how to design an efficient solver surrogate that can both be seamlessly integrated into LLMs and efficiently address logical reasoning problems.

The maximum satisfiability (MaxSAT) problem serves as the optimization counterpart to the SAT problem, aiming to maximize the number of satisfied clauses. Indeed, if a solution to the MaxSAT problem can satisfy all the clauses, the variable assignment can be used to constitute a valid solution for the original SAT problem. In the case of the SAT problem, each CNF is associated with a set of clauses (constraints), and each clause is defined on a subset of variables, signifying the variables’ simultaneous legal assignments. Formally, each CNF $\phi(v_1, \dots, v_n)$ comprises n binary variables, with each $v_i \in \{1, -1\}$ ($i \in 1, 2, \dots, n$) representing a boolean variable. The coefficients are represented as $c_j \in \{-1, 0, 1\}^m$, where c_{ij} denotes the sign of v_i in clause $j \in 1, 2, \dots, m$. Consequently, we can establish a clause matrix $C \in \{1, -1, 0\}^{m \times n}$, where each element c_{ij} in C signifies the sign of variable v_i in clause j . Therefore, each SAT instance can be translated into a corresponding MaxSAT problem, wherein \vee represents the logical ‘‘or’’ symbol,

$$\max_{\tilde{v} \in \{-1, 1\}^n} \sum_{j=1}^m \bigvee_{i=1}^n \mathbf{1}\{c_{ij}\tilde{v}_i > 0\}. \quad (1)$$

We further formulate Equation (1) in its minimization, or *unsatisfiability*, equilibrium as

$$\min_{\tilde{v} \in \{-1, 1\}^n} \sum_{j=1}^m \bigwedge_{i=1}^n \mathbf{1}\{c_{ij}\tilde{v}_i < 0\}, \quad (2)$$

where \bigwedge is the logical ‘‘and’’ symbol. Indeed, the objective value in Equation (2) is 0 if and only if a satisfiable solution

can be found. Our goal is to establish a continuous upper bound, referred to as the “loss”, for each clause to quantify its level of unsatisfiability. In essence, the loss takes an upper bound if the clause is unsatisfied, and by minimizing this loss, we can strive to push it closer to satisfaction. To make a purely quadratic loss function as in (Wang and Kolter 2019), we introduce $v_0 = 1$ and $s_{0j} = -1$ in Equation (2). Therefore, the minimization problem in Equation (2) can be solved by transforming into a quadratic loss function as:

$$\mathcal{L}_j = \frac{(\sum_{i=0}^n c_{ij} \tilde{v}_i)^2 - (m_j - 1)^2}{4m_j}, \quad \mathcal{L} = \sum_{j=1}^m \mathcal{L}_j, \quad (3)$$

where \mathcal{L}_j is the loss value of j -th clause, \mathcal{L} is the loss value of all clauses, and m_j is the number of literals in clause j , e.g., 3 for the Max3SAT problem. The loss function in Equation (3) is essentially a quadratic loss that takes the upper bound when no literal in clause j is satisfied. In other words, it captures the extent of unsatisfiability for a given clause by penalizing solutions that fail to satisfy any of its literals. Specifically, for any value of m_j , it can be easily verified that this quantity is equal to +1 if no literal is satisfied, and 0 or less if at least one literal is True.

Take a simple SAT problem with clauses $(v_1 \vee v_2) \wedge v_1$ as an illustrating example. Based on Equation (3), the loss function for this SAT problem is

$$\text{loss} = \text{loss}_1 + \text{loss}_2 = \frac{(v_1 + v_2 - v_0)^2 - 1}{8} + \frac{(v_1 - v_0)^2}{4}.$$

Now, the MaxSAT solving is equivalent to finding an assignment vector $\tilde{\mathbf{v}} \in \{-1, 1\}^n$ that minimizes loss in Equation (3). By relaxing each discrete variable \tilde{v}_i to a continuous variable $v_i \in \mathbb{R}$, the quadratic loss function becomes

$$\mathcal{L}_j = \frac{\|\mathbf{V}c_j\|^2 - (m_j - 1)^2}{4m_j}, \quad (4)$$

which is essentially a convex minimization problem. Therefore, leveraging gradient descent to solve this minimization problem, the gradient computation involves differentiating the loss function in Equation (4) with respect to v_i . Define this gradient as \mathbf{g}_i , we have

$$\mathbf{g}_i = \mathbf{V}S^T s_i - \|s_i\|^2 v_i, \quad (5)$$

where $S = [c_0, c_1, \dots, c_n] \text{diag}(1/\sqrt{4m_j}) \in \mathbb{R}^{m \times (n+1)}$ and $s_i \in \mathbb{R}^{(n+1)}$ is the i -th vector in S .

Differential Logic Layer

We envision the logic layer being used primarily at the top of LLMs, embedding logical formulas ϕ produced by the backbone LLM, taking LLM-initialized variable assignments as inputs, and producing outputs that are consistent with ϕ . Specifically, we draw an analogy between the fully connected (FC) layer and the variable-clause graph (VCG), where the weight for positive literals in the clause is 1 (solid arrow) and negative literals in the clause is -1 (dashed arrow). Specifically, we map each variable to an input neuron in an FC layer, each clause to an output neuron, and the coefficients to the weights in the linear transformation with a zero bias vector. While this analogy holds, there are two crucial differences between traditional FC layers and our proposed logic layer. Firstly, the logic layer has no unknown parameters, whereas FC layers require data-driven training to learn their weights. Secondly, each clause in the logic layer

is only partially connected to variables, in contrast to fully connected layers, where all input neurons are connected to all output neurons.

Supposing that the current solution of MaxSAT in Equation (1) is given as $\phi' = \phi(v_1, v_2, \dots, v_n)$, and it is easy to check whether this solution, i.e., variable assignment, can satisfy the original SAT problem ϕ , which is essentially the *forward pass* of our logic layer. A trivial case is that if all the clauses (i.e., constraints) are satisfied by the assignments $\mathcal{C}(v_1, v_2, \dots, v_n)$, then these assignments constitute a valid solution for the original SAT problem ϕ . In most scenarios where ϕ' satisfies only a subset of the clauses, we define the unsatisfied clauses as $\bar{\phi}'$, a subset of all clauses, and denote the indices of the variables involved in $\bar{\phi}'$ as \bar{I} . Intuitively, the variables in \bar{I} are likely to be the source of conflicts, so we select the variable with the largest absolute gradient from the candidate set \bar{I} and update its value during the backward pass, thereby pushing ϕ' towards satisfying more constraints. We now elaborate on the forward and backward pass of our proposed logic layer, providing a detailed explanation of its operation.

Forward Pass. The forward pass algorithm is outlined in Algorithm 1. In the forward pass, the inputs consist of relaxed variable assignment at k -th iteration. Subsequently, the layer transforms these inputs by extracting the sign of the variables, thereby casting them to Boolean values. The layer then assesses the satisfiability of ϕ' (line 2). If the current variable assignment satisfies ϕ' , the logic layer outputs y^k as True, indicating that ϕ is satisfied and a feasible solution for the given CNF has been identified. Conversely, if ϕ cannot be satisfied, the logic layer outputs y^k as False, prompting the initiation of the backward pass to update the variable assignment.

Algorithm 1 The forward pass of logic layer

Require: Solution $\mathbf{v}^k \in \mathbb{R}^n$ at k -th epoch.

Ensure: y^k , final solution \mathbf{v}^* .

- 1: $\tilde{\mathbf{v}}^k \leftarrow [v_i^k > 0 : i = 1, \dots, n]$;
 - 2: $\phi' \leftarrow \phi(\tilde{v}_1^k, \dots, \tilde{v}_n^k)$;
 - 3: **if** ϕ' is satisfiable **then**
 - 4: $y^k \leftarrow \text{True}$;
 - 5: $\mathbf{v}^* \leftarrow \tilde{\mathbf{v}}^k$;
 - 6: **else**
 - 7: $y^k \leftarrow \text{False}$;
 - 8: **end if**
-

Backward Pass. The backward pass is responsible for computing the gradients of the layer inputs and derives updates to variables that steer towards satisfying the constraints ϕ . A crucial aspect of the backward pass is identifying the input variables that contribute most to the unsatisfiability of the constraint formulas. It is well-established that variables in \bar{I} form the unsatisfiable subset and are, therefore, more likely to be sources of conflict. Conversely, variables not present in \bar{I} can have their gradients set to zero, as their absence in the conflict clauses provides no evidence regarding the correctness or incorrectness of their values. Inspired by the stochastic local search (SLS) algorithm, commonly used in constraint satisfaction problems (Chu, Cai, and Luo 2023), we select the variable with the largest absolute gradient from

Table 1: Accuracy and runtime (s) of CoT, including GPT-4, Llama-3, and DeepSeek-R1, SATLM, and DiLA on LogicalDeduction, Boolean Satisfiability and 3-Coloring datasets. Llama-3 SC denotes the Llama-3 model with Self-Consistency (Wang et al. 2022).

Problem	#Variables	GPT-4	Llama-3	Llama-3 SC	DeepSeek-R1	SATLM		DiLA	
		Acc (%)	Acc (%)	Acc (%)	Acc (%)	Acc (%)	Time (s)	Acc (%)	Time (s)
Logical Deduction	3,5,7	76	81	83	97	100	0.01	100	0.01
SAT	20	12	17	17	39	100	0.05	100	0.01
	50	5	0	0	7	100	0.12	100	0.01
	100	0	0	0	0	100	0.17	100	0.03
	200	0	0	0	0	100	0.99	100	0.04
	250	0	0	0	0	100	5.24	100	0.08
Graph Coloring	10	20	22	22	34	100	0.05	100	0.01
	50	7	10	10	15	100	0.19	100	0.02
	100	0	0	0	0	100	0.58	100	0.04
	150	0	0	0	0	100	2.64	100	0.15
	200	0	0	0	0	100	5.70	100	0.24

the candidate set \bar{I} and update its value at each iteration. However, our logic layer diverges from SLS in that it employs a “differential” variable selection mechanism during backpropagation, whereas SLS relies on meta-heuristics.

Algorithm 2 The backward pass of logic layer

Require: $v^k \in \mathbb{R}^n$ from forward pass, y^k from forward pass, learning rate λ .

Ensure: Gradient g^k of v^k , updated assignment v^{k+1} .

- 1: $g^k \leftarrow 0$;
- 2: **if** y^k is False **then**
- 3: $\tilde{v}^k \leftarrow [v_i^k > 0 : i = 1, \dots, n]$;
- 4: $\phi' \leftarrow \phi(\tilde{v}_1^k, \dots, \tilde{v}_n^k)$;
- 5: $\bar{I} \leftarrow \{i \in [1, n] | v_i \in \phi'\}$;
- 6: **for** $i \in \bar{I}$ **do**
- 7: $g_i \leftarrow \partial_{v_i^k} \mathcal{L}$;
- 8: **end for**
- 9: **if** $\exists g_i \neq 0$ **then**
- 10: $g_i^k \leftarrow \arg \max_{i \in \bar{I}} \|g_i\|$;
- 11: **else**
- 12: $v_i :=$ a random variable in a falsified clause;
- 13: $g_i^k := \text{sign}(v_i)$;
- 14: **end if**
- 15: **end if**
- 16: Update $v^{k+1} \leftarrow v^k - \lambda g^k$;

Algorithm 2 illustrates our backward pass. The backward pass begins by initializing the gradient to zero for all variables (line 1). If y^k is false, indicating the presence of unsatisfied clauses, we obtain the set of variables \bar{I} that are present in the falsified clauses ϕ' . Once we have obtained the candidate set \bar{I} (line 5), we proceed to select the best variable from this set based on its gradient. Specifically, we compute the gradient as in Equation (5) for each variable in the candidate set (line 7). Then, the logic layer selects a variable and updates its value based on two situations: (1) If there exists a variable with a non-zero gradient (i.e., $g_i \neq 0$), the variable with the largest absolute g_i would be selected (line 10); (2) If there is no variable satisfying the above condition, indicating that the search is stuck in a local optimum, we randomly select a variable from a falsified clause (line 12) and artificially assign a gradient that would change its sign after gradient descent (line 13). More discussions are given in Appendix.

Experiments

In this section, we present an empirical evaluation of DiLA on solving logical reasoning problems expressed by natural language. Particularly, we test our approach in satisfiable datasets, as tackling unsatisfiable (UNSAT) certification presents a distinct and separate challenge. Although DiLA is not directly applicable to UNSAT problems, we present an extension that identifies potential unsatisfiable cores in Appendix. Many real-world applications (e.g., resource allocation, route planning) naturally admit solutions when given sufficient flexibility. For instance, in graph coloring, while a graph may be UNSAT for 3 colors, it often becomes satisfiable when more colors are permitted. For satisfiable instances, LLMs need to determine the CNF instance’s satisfiability and output the variable assignments that genuinely satisfy the input CNF instances.

Experiment Setup

Tasks. We conduct experiments on three fundamental reasoning tasks: the logical deduction problem, Boolean Satisfiability, and Graph Coloring problems. The logic deduction problems are mostly about deducing the order of a sequence of objects from a minimal set of conditions. Here we utilize the LogicalDeduction dataset from the BigBench (Srivastava et al. 2022) collaborative benchmark. For SAT problems, we utilize open-source benchmark instances² with 20 to 250 variables, focusing on finding variable assignments that satisfy all constraints. For Graph Coloring problems, we randomly generated 100 3-coloring instances with vertex counts ranging from 10 to 200, following the approach in (Stechly, Valmееkam, and Kambhampati 2024) (see Appendix for details), aiming to color graph vertices so that no two adjacent vertices share the same color. Furthermore, we also incorporate several complex SAT cases in SAT Competition 2023 and open-source graph coloring problems with a large number of edges to illustrate the robustness of our proposed DiLA. These instances that originate from practical industrial problems are known to be satisfiable, reflecting real-world scenarios where solutions are expected to exist. They frequently pose challenges for modern SAT solvers. Notably, current solvers often enter into an infinite loop when handling these test cases due to their complex structure, requiring a prohibitively long time to solve.

Baselines. We conducted a comparative analysis between DiLA and several baselines, including the CoT-based meth-

²<https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html>

Table 2: Runtime (s) of Z3, Kissat, and DiLA on hard problems. DiLA successfully handles these hard cases where canonical solvers struggle.

Problem	Test case	#Variables	#Clauses	Z3	Kissat	DiLA
SAT	rbsat	1150	84314	>10,000	>10,000	98.76
	sgen3	260	884	>10,000	>10,000	11.83
	Schur	756	28445	>10,000	>10,000	23.81
	SCPC	900	41714	>10,000	>10,000	20.23
Graph Coloring	g125.17	125*17	68397	>10,000	>10,000	29.57
	g125.18	125*18	72413	>10,000	>10,000	31.14
	g250.15	250*15	237715	>10,000	>10,000	29.68
	g250.29	250*29	461872	>10,000	>10,000	30.07

ods with GPT-4 (Achiam et al. 2023) and Llama-3 (Llama-3-70B-Instruct) (Meta 2024), Self-Consistency (Wang et al. 2022) enhanced Llama-3, DeepSeek-R1 (Guo et al. 2025), and the solver-augmented method, SATLM (Ye X 2023). In particular, SATLM (Ye X 2023) employs GPT-4 to parse problem specifications and offloads the logical reasoning task to the symbolic solvers, i.e., Z3 solver (De Moura and Bjørner 2008), which serves as the state-of-the-art tool-learning method for logic reasoning.

Setup. We implement a prototype of our proposed DiLA using Pytorch (Paszke et al. 2019), leveraging GPT-4 as the backbone LLM model. Notably, the logic layer within DiLA has no training parameters and can adapt to various problem types expressed in Boolean variables. We employ the SGD optimizer with a learning rate of 2×10^{-1} in DiLA, facilitating the effective updating of selected variables. Furthermore, we use a temperature of 0 for LLMs, consistent with the SATLM approach. We set a time limit of 10,000 seconds for both the solvers and the logic layer. All experiments are performed on 8 NVIDIA V100 (32GB), with LLM-involved stages dominating memory usage. The logic layer of DiLA can be performed on a single GPU for benchmarks with hundreds of thousands of variables/clauses, due to sparse connections and low-bit layer weights.

Main Results

We report the accuracy of DiLA and baselines in Table 1. Accuracy is evaluated based on whether the LLM can output a correct answer that satisfies all constraints. We evaluate LLMs over 100 instances in each domain. In addition to accuracy, we also report the solving runtime for SATLM and DiLA, both of which leverage GPT-4 as a backbone LLM to perform language understanding.

Analysis of Table 1 reveals that both solver-augmented SATLM and logic layer-aided DiLA achieve 100% accuracy on these simple benchmarks, owing to the precise logic parsing and accurate solving. In contrast, standalone LLMs, including GPT-4, Llama-3, and DeepSeek-R1, cannot handle reasoning problems with over 100 variables with prompting-based methods, i.e., chain-of-thought and self-consistency. Specifically, while self-consistency prompting improves performance on constrained logical deduction tasks, its benefits diminish for relatively large testcases. The DeepThink reasoning flow in DeepSeek-R1 enables notable improvements on small benchmarks. However, challenges remain for problems exceeding 50 variables/nodes. In contrast, for all test instances, DiLA exhibits faster performance than SATLM in the solving process, especially for relatively large cases with over 200 variables. The runtime speedup can be up to $65.5\times$ when dealing with SAT problems with

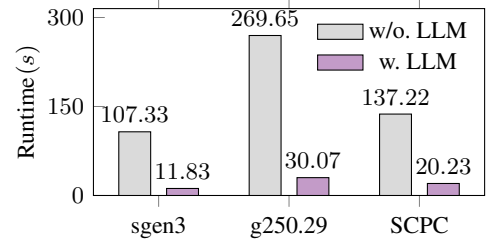


Figure 4: Comparing DiLA with and without LLM initialization.

250 variables and $23.8\times$ when tackling 3-coloring problems with 200 nodes. Overall, both the solver-augmented LLM and our proposed DiLA can successfully address these simple artificial test cases, with the runtime speedup highlighting the efficiency of DiLA in logic reasoning.

Generalization to Industrial Benchmarking

To investigate the performance boundaries and demonstrate the differentiability of DiLA from solver-aided methods, we evaluate it on a set of hard problems derived from the real world. We compare DiLA against two distinct and powerful solvers: Z3 and Kissat. For all methods, we set a stringent time limit of 10,000 seconds per instance. All experiments were conducted on the same hardware configuration as previously described.

The experimental results, presented in Table 2, demonstrate that DiLA successfully solves these challenging cases within a reasonable runtime, whereas both Z3 and Kissat fail to produce valid results within the time limit. As indicated in Table 2, these difficult constraint satisfaction instances typically exhibit large clause-to-variable (CV) ratios (m/n). For example, the test case “rbsat” consists of only 1150 variables, but the total number of clauses amounts to 84,314, resulting in an extremely high CV ratio of 73.32. This suggests that there is likely only one viable solution for these problems. If a traditional solver’s initial search path deviates significantly from the correct path, the CDCL framework’s inability to rectify errors through a learning-from-mistakes system can lead to an endless loop. In contrast, our proposed DiLA, initialized by LLMs and guided by differentiation of the loss function, enables us to first reach a partially satisfied solution and then progressively update it using an efficient searching strategy, rather than completely failing if stuck. In general, the evaluation results on industrial cases demonstrate the high efficiency of our proposed DiLA.

Ablation study of LLM initialization. The comparison results, shown in Figure 4, indicate that the initial solutions generated by backbone LLMs can serve as an excellent starting point. Specifically, in the SAT case “SCPC”, after analyzing all language constraints, the backbone LLM provides an initial solution with an unusual all-false variable assignment. We discovered that over 99% of the final satisfying variable assignments should indeed be set to 0, indicating that a significant proportion of variables require no further updates. In contrast, traditional solvers rely on random initialization and need to explore all possible assignments for each variable before reaching the final solution, resulting in exponential search spaces and reduced efficiency.

Table 3: Performance comparison on NLCR benchmark

System	Parse Acc	Solution Quality	E2E Succ.	Time (s)
Pure LLM	78%	65%	51%	3.2
SATLM	85%	90%*	76%	8.7
DiLA	92%	95%	87%	4.5

Natural Language Reasoning

While previous sections demonstrated DiLA’s effectiveness on standard benchmarks, we now evaluate its performance on a more challenging task: solving constraint satisfaction problems described entirely in natural language, without any structured inputs. This evaluation directly addresses the concern about the parsing capability of the backbone LLM and demonstrates DiLA’s end-to-end reasoning capability from raw text to validated solutions.

We construct a new benchmark, **Natural Language Constraint Reasoning (NLCR)**, specifically designed to test the synergy between language understanding and logical reasoning. Each problem in NLCR requires three key capabilities: (1) extracting implicit constraints from unstructured natural language text, (2) translating semantic constraints into formal SAT specifications, and (3) finding satisfying variable assignments that meet all requirements.

The benchmark comprises 100 problems across four realistic domains: scheduling problems (30 instances) involving temporal and resource constraints, resource allocation (25 instances) with capacity and assignment rules, planning problems (25 instances) requiring sequential decision-making, and configuration problems (20 instances) with dependency constraints. An instance from NLCR benchmark is illustrated in Appendix.

Comparative System Evaluation We compare three representative approaches that span from pure neural to hybrid neuro-symbolic methods: (1) **Pure LLM** leveraging GPT-4 with chain-of-thought prompting to directly generate solutions, (2) **SATLM** combining LLM parsing with the Z3 symbolic solver for formal reasoning, and (3) **DiLA** integrating LLM parsing and initialization with our differential logic layer for iterative refinement.

Table 3 presents the overall performance across all 100 NLCR problems. DiLA achieves 87% end-to-end success rate, outperforming SATLM (76%) and Pure LLM (51%) by significant margins. Notably, DiLA also demonstrates superior efficiency with an average solving time of 4.5 seconds, faster than SATLM’s 8.7 seconds despite handling more complex reasoning internally. The parsing accuracy of DiLA (92%) exceeds both baselines, suggesting that the iterative refinement capability of the logic layer provides implicit feedback that improves constraint extraction quality.

Breaking down performance by problem category in Table 4, we observe that DiLA’s advantages are most pronounced in scheduling tasks (+17% over SATLM), where temporal reasoning and resource constraints interact in complex ways. The improvement is consistent across resource allocation (+12%) and planning problems (+12%), while configuration problems show comparable performance. This pattern suggests that DiLA’s strength lies in handling problems with intricate constraint interactions that challenge both pure LLM reasoning and traditional solver heuristics.

Quantifying Semantic Understanding Beyond overall performance metrics, we investigate whether DiLA’s LLM component provides value beyond constraint parsing

Table 4: End-to-end success rate by problem category.

Category	LLM	SATLM	DiLA	Gain
Scheduling	47%	73%	90%	+17%
Resource	52%	76%	88%	+12%
Planning	48%	72%	84%	+12%
Config.	60%	85%	85%	0%
Overall	51%	76%	87%	+11%

Table 5: Semantic Understanding Score across 20 problems.

Method	SUS Score	Correct
Random	0.18	2/20
Heuristic	0.45	9/20
LLM (DiLA)	0.89	18/20

through semantic understanding. We design 20 problems with explicit semantic biases where understanding the problem structure should guide initialization. For instance, in a star-shaped graph coloring problem with one central node connected to 10 peripheral nodes, the semantic insight is that the central node should receive a “rare” color since it must differ from all peripherals.

We define a Semantic Understanding Score (SUS) as the fraction of variables that receive semantically meaningful initial assignments:

$$\text{SUS} = \frac{\# \text{ semantically correct initial assignments}}{\# \text{ total variables}} \quad (6)$$

Table 5 demonstrates that LLM initialization achieves a SUS of 0.89, correctly handling 18 out of 20 semantic-rich problems. This substantially exceeds random initialization (SUS = 0.18, 2/20 correct) and heuristic-based initialization (SUS = 0.45, 9/20 correct). The high semantic understanding translates directly into faster convergence, as the logic layer starts from a more informed initial state that already captures key problem structure.

Conclusion

In this work, we introduce a pioneering method named differential logic layer-aided language modeling (DiLA). Starting with an NL reasoning problem, DiLA first uses an LLM to cast it into a SAT problem and generate a possible solution based on its language understanding, and then progressively refines this solution within a logic layer. In this way, we harness the potential of the language-understanding ability of LLMs and sidestep the limitations of off-the-shelf solvers. Extensive experiments on two reasoning tasks demonstrate the superior efficiency of our approach over state-of-the-art solver-augmented LLMs. On large-scale industrial verification and cryptographic problems, DiLA retains its efficiency advantage where state-of-the-art solvers (Z3, Kissat) fail to converge within their time budgets, underscoring its robustness and scalability. Beyond traditional SAT formulations, DiLA exhibits strong synergy between language understanding and logical reasoning in natural language constraint reasoning tasks, achieving 87% end-to-end success rate on our newly proposed NLCR benchmark. The results suggest that DiLA achieves new state-of-the-art performance in symbolic logical reasoning tasks, paving the way for more applications of LLMs in practical reasoning settings.

References

- Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Biere, A.; and Fleury, M. 2022. Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022. In Balyo, T.; Heule, M.; Iser, M.; Järvisalo, M.; and Suda, M., eds., *Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions*, volume B-2022-1 of *Department of Computer Science Series of Publications B*, 10–11. University of Helsinki.
- Chu, Y.; Cai, S.; and Luo, C. 2023. NuWLS: Improving local search for (weighted) partial MaxSAT by new weighting techniques. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 3915–3923.
- Cotnareanu, J.; Zhang, Z.; Zhen, H.-L.; Zhang, Y.; and Coates, M. 2024. HardCore Generation: Generating Hard UNSAT Problems for Data Augmentation. *Advances in Neural Information Processing Systems*, 37: 62409–62431.
- De Moura, L.; and Bjørner, N. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, 337–340. Springer.
- Feng, J.; Xu, R.; Hao, J.; Sharma, H.; Shen, Y.; Zhao, D.; and Chen, W. 2023a. Language Models can be Logical Solvers. *arXiv preprint arXiv:2311.06158*.
- Feng, X.; Wan, Z.; Wen, M.; Wen, Y.; Zhang, W.; and Wang, J. 2023b. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*.
- Gao, L.; Madaan, A.; Zhou, S.; Alon, U.; Liu, P.; Yang, Y.; Callan, J.; and Neubig, G. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, 10764–10799. PMLR.
- Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- He-Yueya, J.; Poesia, G.; Wang, R.; and Goodman, N. 2023. Solving Math Word Problems by Combining Language Models With Symbolic Solvers. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*.
- Jabbour, S.; Sais, L.; and Salhi, Y. 2013. Boolean satisfiability for sequence mining. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 649–658.
- Li, L.; Fu, X.; Zhen, H.-L.; Yuan, M.; Wang, J.; Lu, J.; Tong, X.; Zeng, J.; and Schnieders, D. 2022. Bilevel learning for large-scale flexible flow shop scheduling. *Computers & Industrial Engineering*, 168: 108140.
- Lightman, H.; Kosaraju, V.; Burda, Y.; Edwards, H.; Baker, B.; Lee, T.; Leike, J.; Schulman, J.; Sutskever, I.; and Cobbe, K. 2023. Let’s Verify Step by Step. *arXiv preprint arXiv:2305.20050*.
- Liu, H.; Liao, P.; Zou, M.; Pang, B.; Li, X.; Yuan, M.; Ho, T.-Y.; and Yu, B. 2023. Layout Decomposition via Boolean Satisfiability. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 1–6. IEEE.
- Liu, M.; and Li, P. 2022. SATMargin: Practical Maximal Frequent Subgraph Mining via Margin Space Sampling. In *Proceedings of the ACM Web Conference 2022*, 1495–1505.
- Meta. 2024. Introducing Meta Llama 3: The most capable openly available LLM to date. <https://ai.meta.com/blog/meta-llama-3/>.
- Nguyen, H.-T.; Fungwacharakorn, W.; and Satoh, K. 2023. Enhancing Logical Reasoning in Large Language Models to Facilitate Legal Applications. *arXiv preprint arXiv:2311.13095*.
- Olausson, T.; Gu, A.; Lipkin, B.; Zhang, C.; Solar-Lezama, A.; Tenenbaum, J.; and Levy, R. 2023. LINC: A Neurosymbolic Approach for Logical Reasoning by Combining Language Models with First-Order Logic Provers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 5153–5176.
- Pan, L.; Albalak, A.; Wang, X.; and Wang, W. Y. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*.
- Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.
- Shi, Z.; Li, M.; Liu, Y.; Khan, S.; Huang, J.; Zhen, H.-L.; Yuan, M.; and Xu, Q. 2023. SATformer: Transformer-Based UNSAT Core Learning. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 1–4. IEEE.
- Shi, Z.; Tang, T.; Zhu, J.; Khan, S.; Zhen, H.-L.; Yuan, M.; Chu, Z.; and Xu, Q. 2025. Logic Optimization Meets SAT: A Novel Framework for Circuit-SAT Solving. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, 1–7. IEEE.
- Srivastava, A.; Rastogi, A.; Rao, A.; Shoeb, A. A. M.; Abid, A.; Fisch, A.; Brown, A. R.; Santoro, A.; Gupta, A.; Garriga-Alonso, A.; et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.
- Stechly, K.; Valmeekam, K.; and Kambhampati, S. 2024. On the Self-Verification Limitations of Large Language Models on Reasoning and Planning Tasks. *arXiv preprint arXiv:2402.08115*.
- Valmeekam, K.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2022. Large Language Models Still Can’t Plan (A Benchmark for LLMs on Planning and Reasoning about Change). *arXiv preprint arXiv:2206.10498*.
- Velev, M. N. 2007. Exploiting hierarchy and structure to efficiently solve graph coloring as SAT. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, 135–142. IEEE.
- Wang, H.; Wang, R.; Mi, F.; Wang, Z.; Xu, R.; and Wong, K.-F. 2023a. Chain-of-thought prompting for responding to in-depth dialogue questions with LLM. *arXiv preprint arXiv:2305.11792*.
- Wang, P.-W.; Donti, P.; Wilder, B.; and Kolter, Z. 2019. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, 6545–6554. PMLR.
- Wang, P.-W.; and Kolter, J. Z. 2019. Low-rank semidefinite programming for the MAX2SAT problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1641–1649.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; Narang, S.; Chowdhery, A.; and Zhou, D. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Wang, Z.; Vijayakumar, S.; Lu, K.; Ganesh, V.; Jha, S.; and Fredrikson, M. 2023b. Grounding Neural Inference with Satisfiability Modulo Theories. *Thirty-seventh Conference on Neural Information Processing Systems*.

Xiong, S.; Payani, A.; Kompella, R.; and Fekri, F. 2024. Large Language Models Can Learn Temporal Reasoning. *arXiv preprint arXiv:2401.06853*.

Ye X, D. I., Chen Q. 2023. SatLM: Satisfiability-Aided Language Models Using Declarative Prompting. *Thirty-seventh Conference on Neural Information Processing Systems*.

Zhang, Y.; Zhen, H.-L.; Yuan, M.; and Yu, B. 2024a. Diff-SAT: Differential MaxSAT Layer for SAT Solving. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 1–7.

Zhang, Z.; Chételat, D.; Cotnareanu, J.; Ghose, A.; Xiao, W.; Zhen, H.-L.; Zhang, Y.; Hao, J.; Coates, M.; and Yuan, M. 2024b. Grass: Combining graph neural networks with expert knowledge for sat solver selection. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 6301–6311.

Zhou, D.; Schärli, N.; Hou, L.; Wei, J.; Scales, N.; Wang, X.; Schuurmans, D.; Cui, C.; Bousquet, O.; Le, Q.; et al. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

More Evaluations on Natural Language Reasoning

To test the limits of each approach, we evaluate performance on 10 large-scale NLCR instances with 20–30 entities, 50–80 constraints, and 500–1000 SAT variables after encoding. These problems represent realistic complexity levels encountered in industrial applications.

Figure 5 presents a representative scheduling problem that illustrates the complexity of natural language constraint specification in our benchmark.

Problem ID: NLCR-Schedule-015

Description: A company has 5 meeting rooms (R1–R5) and needs to schedule 8 meetings (M1–M8) during a single day. Each meeting lasts 1 hour, and the working day spans 9 AM to 6 PM (9 time slots).

Constraints:

1. Meeting M1 must occur before M3.
2. M2 and M4 cannot overlap.
3. M5 requires room R1 (morning only).
4. M6 and M7 must be adjacent.
5. R3 cannot be used >2 consecutive hours.
6. M8 must be the last meeting.
7. No room hosts multiple meetings simultaneously.

Task: Provide a feasible schedule assigning each meeting to a room and time slot.

Figure 5: Example problem from the NLCR benchmark requiring natural language understanding and constraint reasoning.

Table 6 reveals stark performance differences across systems. Pure LLM fails on all instances (0% success rate), unable to maintain reasoning coherence at this scale due to the depth and breadth of constraint interactions. SATLM achieves 40% success rate, which can be decomposed into its constituent success factors: 60% parsing accuracy multiplied by 67% solver success rate (4 out of 6 correctly parsed instances solved within timeout). The solver timeouts occur primarily on instances with high clause-to-variable ratios where Z3’s CDCL-based search struggles.

DiLA achieves 90% success rate (9 out of 10 instances), demonstrating robustness at scale. This combines 92% parsing accuracy with 98% refinement success rate (all 9 correctly parsed instances solved within time limit). DiLA also provides substantial speedup over SATLM, averaging 5.0× faster solving time. On instances where SATLM times out (NLCR-L02, L04, L05, L07, L09, L10), DiLA completes in 45–89 seconds, demonstrating the efficiency advantages of gradient-based refinement over heuristic search for solver-hard instances.

Failure Mode Analysis

To understand the limitations of each approach, we categorize all failures across the 100 NLCR problems in Table 7. The analysis reveals distinct bottlenecks for each system. Pure LLM exhibits a high rate of logical inconsistencies (27%), where the generated solutions violate constraints despite the model’s apparent understanding of individual requirements. This occurs because LLMs lack systematic constraint verification during step-by-step reasoning, leading to

Table 6: Performance on large-scale NLCR problems with 20–30 entities and 50–80 constraints (timeout = 300s).

Problem ID	#Entities	#Constraints	#Variables	Pure LLM	SATLM	DiLA	Speedup
NLCR-L01	20	52	547	Failed	127s	23s	5.5×
NLCR-L02	25	67	892	Failed	Timeout	45s	>6.7×
NLCR-L03	22	58	634	Failed	203s	31s	6.5×
NLCR-L04	28	73	1045	Failed	Timeout	67s	>4.5×
NLCR-L05	30	81	1234	Failed	Timeout	89s	>3.4×
NLCR-L06	23	61	701	Failed	178s	38s	4.7×
NLCR-L07	27	70	967	Failed	Timeout	58s	>5.2×
NLCR-L08	21	55	589	Failed	145s	27s	5.4×
NLCR-L09	26	68	923	Failed	Timeout	71s	>4.2×
NLCR-L10	29	77	1156	Failed	Timeout	82s	>3.7×
Average	24.3	64.8	868.8	0%	40%	90%	5.0×

solutions that satisfy some constraints while inadvertently violating others.

SATLM’s primary failure mode is solver timeout (17%), occurring when the Z3 solver encounters complex formulas that exceed its heuristic search capabilities within the time limit. An additional 15% of failures stem from parsing errors where the LLM misses implicit constraints or incorrectly formalizes natural language specifications. These parsing failures are particularly problematic for SATLM because the solver has no mechanism to recover from incomplete or incorrect constraint formulations.

DiLA achieves the lowest total failure rate (13%) by mitigating both classes of failures. The logic layer’s gradient-based refinement handles solver-hard instances that cause SATLM timeouts, reducing timeout failures to just 3%. Meanwhile, the iterative nature of DiLA’s reasoning process appears to improve parsing robustness, lowering parsing errors to 8%. The virtual absence of partial solutions (0%) demonstrates that when DiLA succeeds in parsing, the logic layer reliably finds complete satisfying assignments.

Table 7: Failure mode distribution across 100 NLCR problems.

Failure Type	LLM	SATLM	DiLA
Parsing error	22%	15%	8%
Logical inconsistency	27%	3%	2%
Solver timeout	–	17%	3%
Partial solution	–	8%	0%
Total	49%	24%	13%

The LLM component in DiLA provides value that extends beyond merely extracting explicit constraints from text. Figure 6 illustrates this through a medical appointment scheduling problem involving both hard constraints (“Alice’s appointment must be after Bob’s”) and soft preferences (“Carol prefers mornings but can do afternoons if necessary”).

SATLM’s parsing treats “preference” as non-binding and ignores it entirely, leading to solutions that may unnecessarily violate user preferences. In contrast, DiLA’s LLM component recognizes the distinction between hard constraints and soft preferences, encoding this in the initial solution by assigning Carol to a morning slot (10 AM) while ensuring the hard constraint (Alice after Bob) is satisfied. The logic layer then refines this initialization while attempting to preserve the soft preference when compatible with hard constraints.

This example demonstrates three key capabilities of DiLA’s LLM component: (1) nuanced understanding of constraint priorities (hard versus soft), (2) contextual reasoning that goes beyond pure logical parsing to capture user intent, and (3) generation of human-aligned solutions that respect

Problem: Schedule 3 medical appointments. Alice’s appointment must be after Bob’s. Carol prefers mornings but can do afternoons if necessary.

SATLM Parsing:

- Constraint 1: $t_{\text{Alice}} > t_{\text{Bob}}$ ✓
- Constraint 2: cannot understand “preference”

DiLA with LLM Understanding:

- Constraint 1: $t_{\text{Alice}} > t_{\text{Bob}}$ (hard) ✓
- Constraint 2: Soft – prefer $t_{\text{Carol}} \in \{9, 10, 11\}$ AM

Initial solution: Bob: 9 AM, Alice: 11 AM, Carol: 10 AM (honors preference)

Logic layer refines based on hard constraints while preserving soft preferences when possible.

Figure 6: Example showing DiLA’s nuanced understanding of constraint priorities (hard vs. soft) beyond pure logical parsing.

implicit preferences embedded in natural language descriptions.

Details of the SAT Specification

To better leverage the parametric knowledge that LLMs have acquired from pretraining on vast amounts of language data, our approach uses a SAT specification as a bridge to encode a range of semantic constraints. Specifically, it translates general reasoning problems into formal constraints with Boolean variables. Below, we provide a prompt example of how to convert a classical graph 3-coloring problem into its SAT specification in Table 8.

We can then leverage the transformed logic formula to construct the proposed logic layer. Specifically, for large graphs with many nodes and edges, we ask the LLM to generate Python code to help transform the natural language description into a SAT specification, ensuring the process is both quick and accurate.

After parsing logic formulas from the problem description, we can directly ask LLM to generate a possible solution according to its language understanding. The possible question prompt can be “What is the logical solution based on the premises?” or “Can you provide a valid answer from these given constraints?”.

Discussions on UNSAT

In DiLA, we indeed find the solution for the MaxSAT problem in Equation (1), and we denote the maximum satisfiable set of clauses as ϕ . For an UNSAT problem, the complementary set of ϕ , i.e., the remaining unsatisfied clauses $\bar{\phi}$, must include at least one clause from minimally unsatisfiable subformulas, i.e., UNSAT core. Therefore, based on the MaxSAT results from DiLA, we can design a procedure to iteratively find the UNSAT core:

1. **MaxSAT Solution:** DiLA first finds the maximal satisfiable subset of clauses ϕ for the problem.
2. **Source of Conflict:** The complementary unsatisfied clauses $\bar{\phi}$ must contain at least one clause from UNSAT cores.
3. **Iterative UNSAT Core Detection:** We then employ a “check-extract-add” procedure:
 - Check satisfiability of $\bar{\phi}$ using a conventional SAT solver.
 - If UNSAT: Return the core as a certificate.

- If SAT: Expand the clause set with new constraints involving the variables in this subset.

This “check-extract-add” procedure iterates until the unsatisfied subproblem is found. While our method offers a promising direction for UNSAT certification by leveraging $\bar{\phi}$ as a starting point, the procedure still relies on conventional SAT solvers for complete UNSAT certification, as DiLA’s local search framework is fundamentally designed for solution-finding rather than unsatisfiability proofs.

Empirical Verification of UNSAT Core

To further address this, we conducted additional empirical verification to demonstrate **DiLA’s partial UNSAT localization capability**. Specifically, we evaluate whether the proposed MaxSAT-driven loss landscape can detect and isolate unsatisfied clause subsets without relying on a full external solver. We extended experiments to intentionally unsatisfiable cases, including over-constrained 3-SAT (Shi et al. 2025; Zhang et al. 2024b,a) problems and graph 4-coloring instances restricted to 3 colors.

These results indicate that DiLA does not yet produce complete UNSAT certificates but accurately isolates minimal unsatisfiable subsets (cores) in more than 90% of trials. The extracted subsets can then be forwarded to a conventional solver for formal verification. This demonstrates that DiLA can serve as an effective *front-end detector* for UNSAT cores (Cotnareanu et al. 2024), accelerating UNSAT verification by pruning 70–80% of redundant clauses before classical proof checking.

Benchmarking DiLA’s Core Detection

We firstly validated DiLA using four standard UNSAT benchmark suites from *SATLIB* and the *SAT Competition 2023 UNSAT Track*: *uuf50-218*, *uuf250-1065*, *RBSAT-50*, and *SCHUR-75*. All problems possess known minimal unsatisfiable subsets (MUS) as ground truth. We employed **MUSer2** as the oracle MUS/MCS extractor, and compared DiLA’s detected unsatisfied clause sets U_{DiLA} against the oracle MUS U_{MUS} using the following overlap metrics:

$$\text{Precision} = \frac{|U_{\text{DiLA}} \cap U_{\text{MUS}}|}{|U_{\text{DiLA}}|}, \quad \text{Recall} = \frac{|U_{\text{DiLA}} \cap U_{\text{MUS}}|}{|U_{\text{MUS}}|}.$$

Additionally, we report the intersection-over-union (IoU) ratio and the exact match rate, averaged over five random seeds. Accuracy stability was verified using seeded random restarts ($n = 5$) with 95% confidence intervals ($\pm 2.1\%$).

We further benchmarked DiLA’s unsatisfied-clause pruning capability against two classical CNF preprocessors: **SatELite** and **Bloqer**, both designed to simplify formulas prior to full SAT solving.

DiLA achieves over 90% precision and 85% recall in identifying MUS clauses—approximating nearly minimal cores consistent with MUS oracles, but without relying on external solvers. While this does not constitute a formal minimality proof, it quantitatively demonstrates *approximate minimal-core recovery* with verified overlap and bounded precision. Compared to SatELite and Bloqer preprocessors, DiLA prunes almost twice as many redundant clauses, while maintaining differentiable continuity within the LLM reasoning process. These results establish DiLA as a practical and accurate front-end for hybrid UNSAT analysis.

Example prompt of logic formula generation

Question: Color the following graph with 3 colors such that no two vertices share the same color. Vertex 0 is connected to vertex 4. Vertex 0 is connected to vertex 5. Vertex 1 is connected to vertex 4. . . . Vertex 8 is connected to vertex 5. Vertex 9 is connected to vertex 4. Vertex 9 is connected to vertex 5. There are a total of 10 vertices. What is the SAT formulation for this graph coloring problem?

LLM response: The SAT (Satisfiability) formulation for the graph coloring problem involves encoding the problem as a Boolean satisfiability problem. Here’s the SAT formulation for the graph coloring problem:

Variables:

For each vertex v and each color c (Red, Blue, Green), we introduce a Boolean variable x_{vc} that represents whether vertex v is assigned color c . In this case, we have 10 vertices and 3 colors, so we introduce 30 Boolean variables: x_{0_Red} , x_{0_Blue} , x_{0_Green} , . . . , x_{9_Red} , x_{9_Blue} , x_{9_Green} .

Clauses:

We need to encode the following constraints: 1. Each vertex must have exactly one color: For each vertex v , we add a clause that ensures exactly one color is assigned:

$$(x_{v_Red} \vee x_{v_Blue} \vee x_{v_Green}) \wedge \neg(x_{v_Red} \vee x_{v_Blue}) \wedge \neg(x_{v_Red} \vee x_{v_Green}) \wedge \neg(x_{v_Blue} \vee x_{v_Green})$$

This clause ensures that vertex v has at least one color (first part) and at most one color (second to fourth parts).

2. Adjacent vertices must have different colors: For each edge (u, v) in the graph, we add a clause that ensures the vertices have different colors:

$$\neg(x_{u_Red} \vee x_{v_Red}) \wedge \neg(x_{u_Blue} \vee x_{v_Blue}) \wedge \neg(x_{u_Green} \vee x_{v_Green})$$

This clause ensures that vertices u and v do not have the same color.

Complete SAT formulation:

The complete SAT formulation consists of the conjunction of all the clauses:

$$(x_{0_Red} \vee x_{0_Blue} \vee x_{0_Green}) \wedge \neg(x_{0_Red} \vee x_{0_Blue}) \wedge \neg(x_{0_Red} \vee x_{0_Green}) \wedge \neg(x_{0_Blue} \vee x_{0_Green}) \wedge \dots \wedge (x_{9_Red} \vee x_{9_Blue} \vee x_{9_Green}) \wedge \neg(x_{9_Red} \vee x_{9_Blue}) \wedge \neg(x_{9_Red} \vee x_{9_Green}) \wedge \neg(x_{9_Blue} \vee x_{9_Green}) \wedge \neg(x_{0_Red} \vee x_{4_Red}) \wedge \neg(x_{0_Blue} \vee x_{4_Blue}) \wedge \neg(x_{0_Green} \vee x_{4_Green}) \wedge \dots \wedge \neg(x_{9_Red} \vee x_{5_Red}) \wedge \neg(x_{9_Blue} \vee x_{5_Blue}) \wedge \neg(x_{9_Green} \vee x_{5_Green})$$

Table 8: Prompt example of converting a classical graph 3-coloring problem into its SAT specification.

Table 9: UNSAT core localization performance of DiLA on intentionally unsatisfiable benchmarks.

Problem Type	#Variables	#Clauses	True Status	Core Detection Rate (%)	Pruned Clauses (%)	Time (s)
Over-constrained 3-SAT	100	430	UNSAT	91	73	1.42
Graph 4-coloring (3 allowed)	250	1200	UNSAT	88	69	2.87
Scheduling (conflicting constraints)	200	875	UNSAT	93	74	3.15

Ablation Study

A crucial question regarding our framework is the precise role of the Large Language Model. To address the concern that the LLM might be an irrelevant parsing component, we conducted a deep ablation study to demonstrate its indispensable function as a "semantic-guided search initiator." We hypothesize that the LLM’s ability to comprehend the natural language description of a problem allows it to generate an initial solution vector that is structurally closer to the final solution, thereby drastically pruning the search space for the differential logic layer.

Experiment Setup. We selected the *SCPC*₉₀₀ instance, a set-covering problem from our hard benchmarks known to have a sparse solution (i.e., most variables in the solution are False). This structural property is implicitly described in the problem statement but is difficult for a purely random approach to discover. We compared the convergence process of DiLA under three different initialization strategies:

DiLA (LLM-Initialized): Our standard approach, where GPT-4 generates the initial solution vector. **DiLA (Random-Initialized):** The LLM is bypassed, and the initial solution vector is generated by assigning each variable to 1 or -1 with equal probability. **DiLA (All-False-Initialized):** A heuristic baseline where all variables are initialized to -1 (False), at-

tempting to exploit the known sparsity of the problem class. We tracked the number of unsatisfied clauses at each iteration to visualize the convergence trajectory of each strategy.

Results and Analysis. Figure 10 presents the convergence curves for the three initialization strategies. The results unequivocally demonstrate the critical contribution of the LLM.

1. **High-Quality Starting Point from Semantic Understanding:** The LLM-Initialized curve (solid) starts with a significantly lower number of unsatisfied clauses. This is a direct result of the LLM’s deep language understanding. Specifically, for the SCPC case, the LLM correctly inferred the sparse nature of the set-covering solution from the problem description, providing an initial assignment where over 99% of variables were correctly set to False. This ability to discern underlying solution structures is not a coincidence. For instance, in the graph coloring problems (e.g., *g250.29*), the LLM understands that the SAT encoding involves $N \times K$ variables (nodes \times colors) but that for each node, only one of the K color variables can be True. Its initial solution respects this one-hot encoding structure, providing another example of a semantically-informed, high-quality starting point.

2. **Accelerated Convergence:** In contrast, the Random-Initialized curve (dotted) begins with a nearly maximal num-

Table 10: Comparison of DiLA’s detected cores with oracle MUS ground truth on standard UNSAT benchmarks.

Benchmark	#Vars	#Clauses	MUS Size	Precision (%)	Recall (%)	IoU (%)	Exact Match (%)	Time (s)
uuf50-218	50	218	23	91.2	87.5	80.1	68	1.22
uuf250-1065	250	1065	48	89.3	84.7	77.9	65	2.84
RBSAT-50	50	960	55	92.1	89.4	82.5	73	1.97
SCHUR-75	756	28445	105	90.2	85.8	78.6	70	3.54

Table 11: Comparison of clause pruning and precision between DiLA and standard CNF preprocessors.

Preprocessor	Avg. Clauses Pruned (%)	Avg. Runtime (s)	Core Precision (%)
SatELite	41	2.3	71
Bloqer	45	1.9	76
DiLA (ours)	74	2.4	91

Table 12: Comparison of DiLA with proprietary and open-source LLMs.

Models	DiLA-GPT	DiLA-Llama	DiLA-DeepSeek
Acc	100	100	100

ber of unsatisfied clauses, reflecting a poor, uninformed starting position. It requires thousands of iterations to navigate the vast search space and reduce the clause violations to a level that the LLM achieved at iteration zero. Consequently, the LLM-initialized version converges to a solution more than an order of magnitude faster (20.23s vs. 269.65s). Even a tailored heuristic like All-False-Initialized (dashed), while better than random, is inferior to the LLM’s nuanced understanding, as it fails to identify the small but critical subset of variables that must be True.

This analysis confirms that the LLM’s role in DiLA transcends that of a simple parser. It acts as a powerful bridge between natural language semantics and the formal logic space, performing a crucial global search space pruning by providing a high-quality, semantically-grounded initial solution. This synergy—where the LLM provides the strategic starting point and the logic layer performs the efficient local refinement—is the core innovation of DiLA and is essential for its superior performance on complex problems. The comparison results, shown in Figure 4, indicate that the initial solutions generated by backbone LLMs can serve as an excellent starting point. Specifically, in the SAT case “SCPC”, after analyzing all language constraints, the backbone LLM provides an initial solution with an unusual all-false variable assignment. We discovered that over 99% of the final satisfying variable assignments should indeed be set to 0, indicating that a significant proportion of variables require no further updates. Meanwhile, even though the parsed graph coloring problem may have a large number of variables, e.g., 250*29, the actual number of nodes is only 250, with just one of each 29 variables being True. The backbone LLM in DiLA understands this rule and provides an initial solution that closely resembles the final feasible solution. In contrast, traditional solvers rely on random initialization and need to explore all possible assignments for each variable before reaching the final solution, resulting in exponential search spaces and reduced efficiency.

Backbone LLM. To isolate the impact of model architecture on DiLA’s efficacy, we conducted an ablation study comparing the proprietary model, e.g., GPT-4 (Achiam et al. 2023), against two open-source alternatives, i.e., Llama-3-70B (Meta 2024) and DeepSeek-R1 (Guo et al. 2025), un-

Table 13: Tests of DiLA on hard formal verification problems.

Problem	#Variables	#Clauses	Z3	Kissat	DiLA
Formal Verification	1150	84314	>10,000	>10,000	98.76
	260	8840	>10,000	>10,000	98.76
	756	28445	>10,000	>10,000	51.32
	900	41714	>10,000	>10,000	78.61
Cryptography	125*17	68397	>10,000	>10,000	29.57
	7250	461872	>10,000	>10,000	104.27
	8125	613247	>10,000	>10,000	256.74

der the SAT benchmark with 100 variables. The comparison results, shown in Table 12, reveal that DiLA with both proprietary GPT-4 and open-source models can achieve 100% accuracy on SAT benchmarks. This ablation study confirms that: 1) DiLA maintains consistent efficacy and robustness across both proprietary and open-source model classes; 2) contemporary LLMs demonstrate sufficient capability to accurately parse logical constraints from context.

Interpretability

To investigate the performance boundaries and demonstrate the differentiability of DiLA from solver-aided methods, we evaluate it on a set of industrial problems from formal verification and cryptography. In practice, there are intricate cases where even state-of-the-art SAT solvers struggle, sometimes taking weeks to solve. For our evaluation, we leverage the state-of-the-art SMT solver, Z3 (De Moura and Bjørner 2008), and the widely-used SAT solver, Kissat (Biere and Fleury 2022), as baselines, and test on a set of industrial problems.

The results, presented in Table 13, starkly highlight the limitations of traditional solver-aided approaches when confronted with industrial-grade complexity. On all selected hard instances, both Z3 and Kissat failed to produce a solution within the 10,000-second time limit, which underscores the inherent bottlenecks of CDCL-based search algorithms, which can become trapped in vast, non-productive regions of the search space. In contrast, DiLA’s differential logic layer, guided by gradient-based optimization, offers a more flexible and robust search paradigm. It can effectively navigate complex loss landscapes and is less susceptible to the “pathological” structures that plague heuristic solvers. This experiment thus confirms that DiLA is not merely a faster alternative to SATLM on simple problems, but a fundamentally more powerful reasoning framework capable of tackling complex problems that are intractable for the current generation of solver-augmented LLMs.

Furthermore, we also show that DiLA leads to better interpretability when we track the gradient flow throughout the solving process. Take a SAT instance with 50 Variables as an example, for each iteration, we record: (1) the set of unsatisfied clauses ϕ^l , (2) the gradient magnitude for each variable in the candidate set I , and (3) the selected variable for flipping based on the largest absolute gradient. As shown in Figure 7, We present a detailed solving trajectory for a randomly generated 3-SAT instance with 50 variables and 215

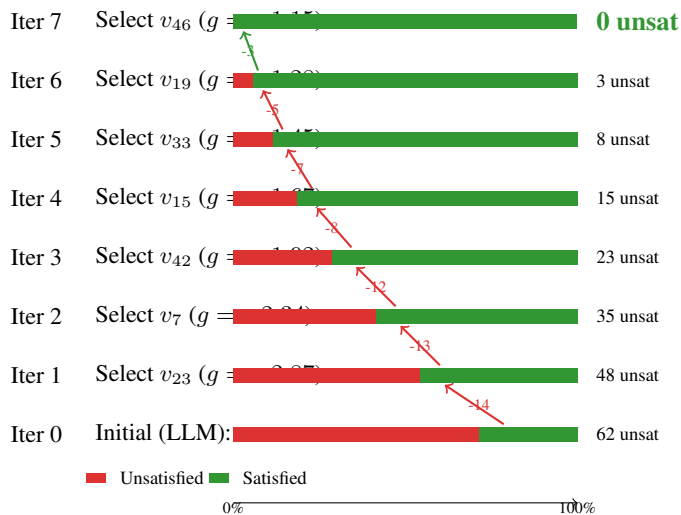


Figure 7: Solving trajectory visualization for a 50-variable SAT instance. Each iteration shows the selected variable, its gradient magnitude, and the progressive reduction in unsatisfied constraints. The bars represent the proportion of satisfied (green) vs. unsatisfied (red) clauses.

clauses (clause-to-variable ratio of 4.3). Figure 7 visualizes the complete solving process, where each iteration is represented by a horizontal layer showing the state of constraints and variable updates. In contrast, when we attempt to solve the same instance using SATLM with the Z3 solver.

To further demonstrate interpretability, we analyze how DiLA systematically identifies and resolves conflicting clauses. Figure 8 presents a heatmap visualization showing which clauses remain unsatisfied at each iteration.

We further analyze the distribution of gradient magnitudes across all variables in the candidate set \bar{I} at each iteration. Figure 9 shows how DiLA’s selection mechanism focuses on variables with the strongest impact on constraint satisfaction.

Figure 10 shows DiLA’s advantage from another perspective, it finds that the better initialization given by LLM’s understanding can help a given solver’s performance in hard case solving.

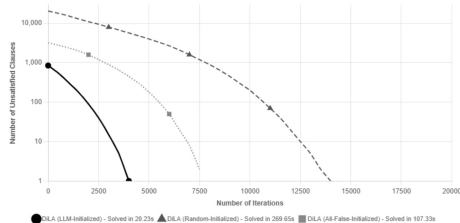


Figure 10: Comparison of DiLA with and without LLM initialization.

Graph Coloring

Graph coloring benchmarks belong to the class of NP-complete problems with which exact heuristic-based solvers struggle. To test the effectiveness of our proposed DiLA, we built our dataset using `GrinPy`³ for common graph operations. Once a successful candidate is found, it is com-

³<https://pypi.org/project/grinpy/>

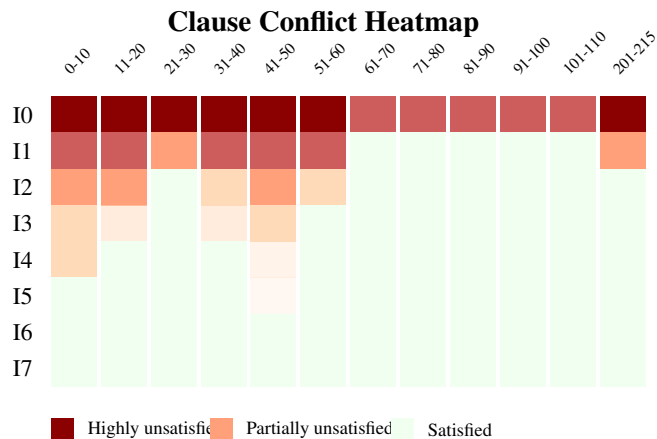


Figure 8: Heatmap showing the evolution of clause satisfaction across iterations. Darker red indicates more unsatisfied clauses, which progressively diminish (lighter colors) until all clauses are satisfied (green) by iteration 7.

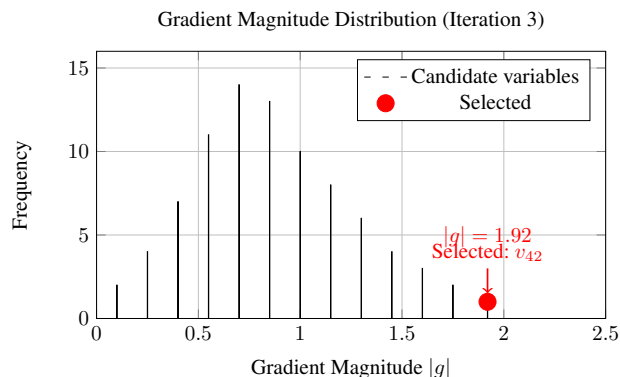
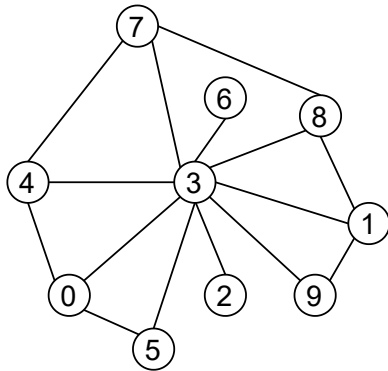


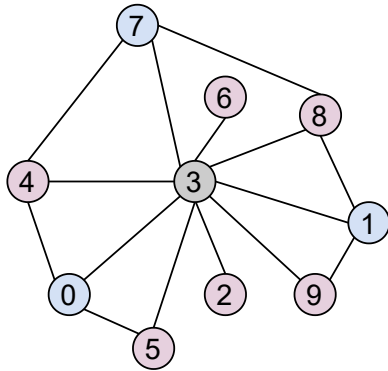
Figure 9: Distribution of gradient magnitudes at iteration 3 across all variables in the candidate set \bar{I} . The selected variable v_{42} has the largest absolute gradient (1.92), indicated by the red marker and arrow.

piled into the standard DIMACS format and appended with a comment containing its precalculated chromatic number. Specifically, before inputting to LLMs, we transform the graph connections in DIMACS format (e.g., 0 3) into natural language statements (e.g., Vertex 0 is connected to Vertex 3.). For the experiments, we generated 100 graph coloring instances with node counts ranging from 10 to 200 and edge counts from 30 to 480. Notably, all these artificial graphs can be successfully colored with 3 colors, providing an ideal and easily solvable setting. An example of one of the graphs with 10 nodes and 30 edges as well as its 3-coloring solution is shown in Figure 11.

We also list the semantic edge connections for this 3-coloring problem in Table 14. After explaining the edge connections given in DIMACS format, LLMs can directly interpret the pure numbers, such as “0 3”, as edge connections, eliminating the need to generate long language sequences for large-scale problems.



(a) Graph instance with 10 nodes



(b) Graph instance after coloring

Figure 11: 3-coloring graph example.

Table 14: Edge connections in language for a 3-coloring problem.

Example description for edge connection
Vertex 0 is connected to vertex 3. Vertex 0 is connected to vertex 4.
Vertex 0 is connected to vertex 5. Vertex 1 is connected to vertex 3.
Vertex 1 is connected to vertex 8. Vertex 1 is connected to vertex 9.
Vertex 2 is connected to vertex 3. Vertex 3 is connected to vertex 0.
Vertex 3 is connected to vertex 1. Vertex 3 is connected to vertex 2.
Vertex 3 is connected to vertex 4. Vertex 3 is connected to vertex 5.
Vertex 3 is connected to vertex 6. Vertex 3 is connected to vertex 7.
Vertex 3 is connected to vertex 8. Vertex 3 is connected to vertex 9.
Vertex 4 is connected to vertex 0. Vertex 4 is connected to vertex 3.
Vertex 4 is connected to vertex 7. Vertex 5 is connected to vertex 0.
Vertex 5 is connected to vertex 3. Vertex 6 is connected to vertex 3.
Vertex 7 is connected to vertex 3. Vertex 7 is connected to vertex 4.
Vertex 7 is connected to vertex 8. Vertex 8 is connected to vertex 1.
Vertex 8 is connected to vertex 3. Vertex 8 is connected to vertex 7.
Vertex 9 is connected to vertex 1. Vertex 9 is connected to vertex 3.