ACCELERATING SAT SOLVING VIA DIFFERENTIABLE DECISION HEURISTICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Boolean satisfiability (SAT) is a fundamental computational primitive underlying verification, planning, synthesis, and automated reasoning. Modern CDCL SAT solvers are highly optimized, but their initial branching and phase heuristics are still largely solver-internal defaults that do not explicitly adapt to the structure of each input formula. We introduce optimization-guided initialization framework that uses a GPU-accelerated model to extract formula-level guidance and warm-start key CDCL heuristics, including phase selection and variable activity. The resulting solver preserves the original CDCL search procedure and remains complete, making the method a drop-in enhancement rather than a replacement for state-of-the-art SAT solving. We evaluate on SAT Competition benchmarks and compare against strong competition solvers in both single-threaded and parallel settings. In the single-threaded setting, our best individual configurations achieve PAR-2 speedups of $1.18\times$ on SAT and $1.07\times$ on UNSAT instances over Kissat. In the parallel setting, our 32-thread phase portfolio achieves a $1.24\times$ SAT PAR-2 speedup over MallobSat (32T) and solves 15 of the 18 SAT instances on which Kissat times out. These results show that optimization-guided initialization can provide practical, structure-aware diversification for both single-threaded CDCL search and parallel SAT portfolios.

1 INTRODUCTION

Boolean Satisfiability (SAT) problem is a fundamental challenge in computer science, serving as a core primitive in hardware and software verification and combinatorial optimization, including automated theorem proving, planning, circuit synthesis, and equivalence checking Biere et al. (2021). Modern SAT solvers build on the Davis–Putnam and DPLL line of complete backtracking procedures Davis & Putnam (1960); Davis et al. (1962) and are now largely driven by the Conflict-Driven Clause Learning (CDCL) paradigm. The practical success of CDCL has been shaped by conflict analysis and clause learning with non-chronological backtracking Marques-Silva & Sakallah (1999), watched literals and VSIDS-style branching Moskewicz et al. (2001), restart strategies Gomes et al. (2000), phase saving Pipatsrisawat & Darwiche (2007), and compact modern solver engineering Eén & Sörensson (2004); Biere et al. (2024). These techniques have made CDCL solvers highly effective on large industrial instances, yet runtime scalability remains a central challenge, as small changes in branching, phase selection, and activity initialization can drastically alter the explored search trajectory. Improving these heuristics is therefore a direct route to improving solver efficiency. However, despite their importance during search, modern solvers typically initialize these heuristic states using static solver defaults rather than formula-specific information.

Recent neural and differentiable approaches suggest that formula structure can provide useful guidance for SAT solving. Neural SAT models and differentiable satisfiability layers can predict satisfiability, recover assignments, or learn logical structure Selsam et al. (2019); Amizadeh et al. (2019); Wang et al. (2019), while hybrid CDCL methods have used learned predictions to guide unsat-core focus, glue variables, branching policies, or phase initialization Selsam & Bjørner (2019); Han (2020); Kurin et al. (2020); Wang et al. (2024). However, standalone neural solvers are not drop-in sound-and-complete replacements for CDCL, supervised guidance depends on training distributions and labeled instance collections, and online learned policies often require invasive modifications to the solver's decision procedure. In parallel, GPU-accelerated SAT efforts exploit massive parallelism to speed up clause sharing, search support, or inprocessing Prevot (2020); Osama et al.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

(2024), but they primarily target computational acceleration rather than differentiable initialization of CDCL heuristic states. In contrast, we use GPU optimization only before search begins: for each input formula, we optimize a differentiable MaxSAT relaxation to derive heuristic priors, then pass this guidance to an otherwise standard Kissat solver.

Our key idea is to make the internal heuristic state of CDCL solvers differentiably initializable. Before search, we optimize a continuous SAT relaxation and translate its outputs into phase preferences and activity-based branching priorities used by existing CDCL engines. Propagation, conflict analysis, clause learning, restarts, and termination remain unchanged, preserving the completeness of the underlying solver. In parallel settings, perturbed versions of the optimized state provide diverse initializations for multiple CDCL workers, allowing one differentiable optimization pass to guide multiple search trajectories.

Our key contributions are summarized as follows:

- **Differentiable CDCL heuristic initialization**: We formulate CDCL initialization as a differentiable SAT optimization problem on GPU and extract two solver-compatible signals from the optimized relaxation. The converged logits provide phase preferences, while a new gradient-derived **saliency score** identifies variables with high influence on clause satisfaction and initializes activity-based branching heuristics such as VSIDS.
- **Drop-in integration with complete CDCL solvers**: Our method operates entirely before search begins and leaves Boolean propagation, conflict analysis, clause learning, restarts, and termination unchanged. It therefore preserves the **completeness and soundness** of the underlying solver and provides an orthogonal initialization interface with standard CDCL implementations.
- **Parallel portfolio extension**: We extend to parallel SAT solving by applying Gumbel perturbations to the optimized logits, turning one GPU optimization pass into diverse phase assignments for independent CDCL workers and multiple complementary search trajectories.
- **Evaluation against strong SAT Competition baselines**: We evaluated our implementation on SAT Competition 2024 benchmarks Heule et al. (2024). In the single-threaded setting, our best individual variants achieve PAR-2 speedups of $1.18\times$ on SAT and $1.07\times$ on UNSAT instances over Kissat. Our VBS further achieves speedups of $1.55\times$ on SAT and $1.13\times$ on UNSAT. In the parallel setting, our 32-thread phase portfolio achieves a $1.24\times$ SAT PAR-2 speedup over MallobSat (32T) Schreiber & Sanders (2024), the SAT Competition 2025 parallel-track winner Codel et al. (2025), and improves 108 of 167 common SAT instances.

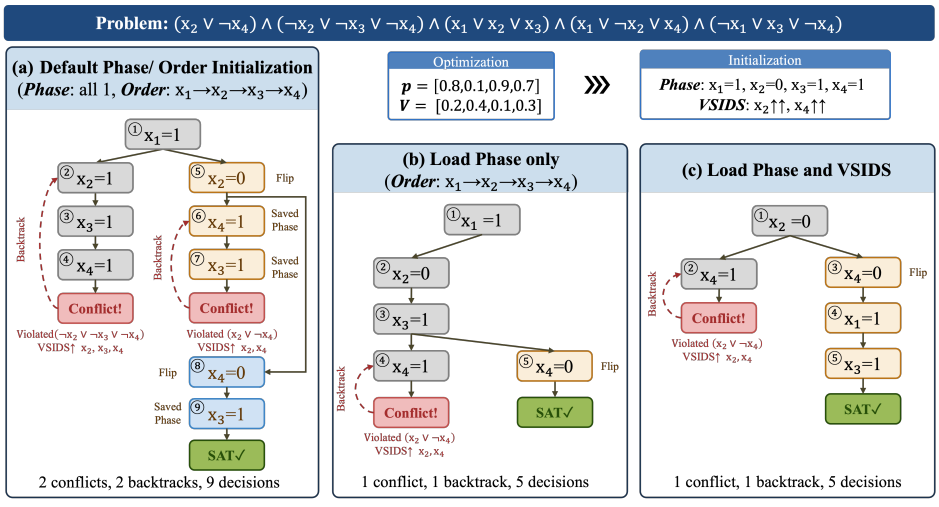

Figure 1: Illustrative CDCL search behavior under different initialization strategies on a 4-variable SAT instance. (a) **Default CDCL initialization**: the solver starts from a default heuristic initial phase and variable order in Kissat, here shown as all phases set to 1 and the illustrative branching order $x_1 \to x_2 \to x_3 \to x_4$. This trajectory encounters 2 conflicts and 2 backtracks before reaching a solution in 9 decisions. (b) **Phase initialization**: phase preferences derived from the optimized assignment logits guide the initial truth values, reducing the number of conflicts. (c) **Phase and VSIDS initialization**: gradient-derived saliency scores are additionally used to initialize variable activity, changing the branching order and further steering the early CDCL trajectory.

2 BACKGROUND

SAT Problem and CDCL Solver A Boolean Satisfiability (SAT) problem asks whether there exists an assignment of truth values to Boolean variables that satisfies a propositional formula. SAT instances are commonly represented in Conjunctive Normal Form (CNF), a conjunction of clauses, where each clause is a disjunction of literals. For example, the formula $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3)$ is satisfied by the assignment $\{x_1 = \text{True}, x_2 = \text{False}, x_3 = \text{True}\}$ because both clauses evaluate to true. A SAT solver either finds such an assignment, returning SAT, or proves that no satisfying assignment exists, returning UNSAT.

Modern complete SAT solvers are primarily based on Conflict-Driven Clause Learning (CDCL), which extends backtracking search with Boolean constraint propagation, conflict analysis, clause learning, and non-chronological backtracking Marques-Silva & Sakallah (1999); Eén & Sörensson (2004); Biere et al. (2021). CDCL repeatedly selects an unassigned variable and assigns it a truth value, then propagates all forced implications of that assignment. If propagation produces a contradiction, the solver analyzes the implication graph, learns a new clause that blocks the conflicting assignment pattern, and backtracks to a non-chronological decision level where search can continue. Modern solvers also use restarts and retain learned clauses across restarts, allowing them to preserve information from previous conflicts while exploring a new region of the search space. The effectiveness of this loop depends strongly on the heuristics that choose which variable to branch on and which truth value to try first.

Critical Heuristics in CDCL. The runtime of a CDCL solver is strongly influenced by the heuristics that control its decision sequence. We focus on two heuristic states shared by many modern solvers: variable activity for branching and phase preferences for polarity selection. Branching decides which unassigned variable to select next, often using VSIDS (Variable State Independent Decaying Sum) Moskewicz et al. (2001), which prioritizes variables appearing in recent conflicts. Although VSIDS scores are updated throughout search, their initial values are set before the solver has accumulated conflict information from the current run Liang et al. (2015), making activity initialization a natural target for gradient-derived variable saliency.

Phase selection determines the truth value assigned to a selected variable. These choices strongly affect the early search trajectory because they determine which region of the assignment space is explored first. Modern solvers often rely on phase saving after search begins Pipatsrisawat & Darwiche (2007), but the initial phase state is still chosen before meaningful conflict feedback is available. Figure 1 illustrates the two initialization targets studied in this work: optimized assignment logits initialize phase preferences, while gradient-derived saliency initializes activity scores, providing CDCL with a differentiable warm start without changing the search algorithm.

Neural, Differentiable, and GPU-Assisted SAT Solving. Recent work has explored learning and differentiable optimization for SAT solving from two main directions. Neural or differentiable SAT models learn formula embeddings, relaxed Boolean assignments, or differentiable satisfiability layers Selsam et al. (2019); Amizadeh et al. (2019); Wang et al. (2019), showing that continuous methods can capture useful logical structure but are not drop-in replacements for complete industrial CDCL solvers. Other methods inject learned guidance into CDCL by predicting unsat-core variables, glue variables, branching policies, or backtracking decisions Selsam & Bjørner (2019); Han (2020); Kurin et al. (2020); Wang et al. (2024). These approaches demonstrate the value of learned signals for CDCL heuristics, but they typically require supervised training across instance collections or modify online solver decisions during search.

A parallel line of work accelerates SAT solving through hardware or large-scale parallelism. GPUShareSAT uses GPUs to accelerate clause sharing Prevot (2020), GPU-accelerated inprocessing speeds up proof-producing CDCL preprocessing and simplification routines Osama et al. (2024), and TurboSAT explores GPU-CPU hybrid neural guidance for SAT search Dai et al. (2025). At larger scale, MallobSat provides a high-performance distributed SAT solving framework based on parallel search and clause sharing Schreiber & Sanders (2024). These systems primarily improve how search computation is parallelized, shared, or accelerated. In contrast, our work addresses a different question: how can differentiable optimization be fused with the heuristic state of a complete CDCL solver before search begins? By using GPU optimization only to initialize phase and activity heuristics, our method avoids replacing CDCL with an incomplete neural solver, avoids invasive

online policy changes, and remains complementary to both single-threaded CDCL engineering and parallel portfolio architectures.

3 METHODOLOGY

Our goal is to integrate differentiable optimization into CDCL solving without replacing the CDCL algorithm itself. Instead of using a neural or continuous solver to make decisions during search, we run a differentiable optimization procedure before CDCL begins and map its outputs onto the heuristic state already used by the solver. After initialization, Boolean propagation, conflict analysis, clause learning, restarts, and termination are handled entirely by the underlying CDCL solver.

Differentiable Initialization Interface for CDCL Figure 2 gives an overview of this interface. Given a CNF formula with n Boolean variables, we construct a learnable logit matrix $\Theta \in \mathbb{R}^{n\times 2}$, where each row $\boldsymbol{\theta}_i$ represents a relaxed assignment preference for variable x_i. We optimize Θ on GPU under a differentiable MaxSAT objective and extract two CDCL-compatible signals from the optimized relaxation. The first signal is a phase preference, derived from optimized assignment probabilities, which initializes the polarity chosen when a variable is branched on. The second signal is a variable saliency score, derived from gradients of the relaxed objective, which initializes activity-based branching heuristics such as VSIDS. These two signals correspond to orthogonal components of CDCL decision making. Phase initialization controls the truth value assigned to a selected variable, while VSIDS initialization influences which variable is selected next. Because the two injection points are independent, we evaluate three initialization variants. **Phase** uses only optimized phase preferences. **Phase+VSIDS** additionally initializes activity scores using gradient-derived saliency. **Phase+VSIDS norm** uses the same phase initialization but normalizes saliency by variable degree before activity initialization. Each variant is orthogonal to the core CDCL engine and can be implemented as a pre-search modification of solver state.

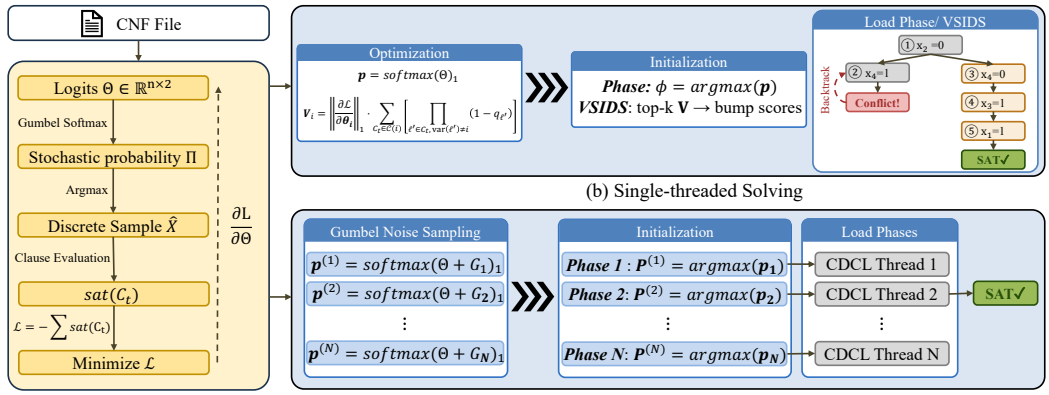

Figure 2: Overview of the differentiable CDCL initialization framework. (a) **GPU-accelerated optimization**: given a CNF formula, we optimize logits $\Theta \in \mathbb{R}^{n\times 2}$ with a differentiable MaxSAT objective and extract two CDCL-compatible signals: assignment probabilities $\mathbf{p}$ and gradient-derived saliency scores V_i. (b) **Single-threaded solving**: $\mathbf{p}$ initializes the phase heuristic, while the top-k variables by V_i receive boosted VSIDS activity scores before CDCL search begins. (c) **Parallel solving**: independent Gumbel perturbations of Θ generate diverse phase assignments for multiple CDCL workers, enabling a phase-portfolio search.

3.1 DIFFERENTIABLE MAXSAT OPTIMIZATION AND PHASE INITIALIZATION

Let the input CNF formula be $F = \bigwedge_{t=1}^{m} C_t$ over n Boolean variables. We convert Boolean clause evaluation into a differentiable arithmetic program using the standard multilinear identities $\neg x = 1 - x$, $x \wedge y = xy$, and $x \vee y = 1 - (1 - x)(1 - y)$. By De Morgan's law, an arbitrary disjunctive clause can therefore be evaluated as one minus the product of its unsatisfied literals. For a variable x_i, we define a learnable two-dimensional logit vector $\boldsymbol{\theta}_i \in \mathbb{R}^2$ and let $p_i = \mathrm{softmax}(\boldsymbol{\theta}_i)_1$ denote

the relaxed probability of assigning x_i to True. For each literal ℓ over x_i, its relaxed satisfaction probability is $q_\ell = p_i$ if $\ell = x_i$ and $q_\ell = 1 - p_i$ if $\ell = \neg x_i$. The relaxed satisfaction of a clause is then

$$\mathrm{sat}(C_t) = 1 - \prod_{\ell \in C_t} (1 - q_\ell), \qquad \mathcal{L}(\Theta) = \sum_{t=1}^{m} (1 - \mathrm{sat}(C_t)) \,. \tag{1}$$

Minimizing $\mathcal{L}$ corresponds to minimizing the relaxed number of unsatisfied clauses. In implementation, Eq. 1 is evaluated as a differentiable tensor program: literal polarities select either p_i or $1 - p_i$, sparse clause-literal incidence maps literals to clauses, and clause products/reductions are computed with vectorized GPU operations.

Continuous probabilities alone can overestimate Boolean consistency, so we use the Gumbel-Softmax straight-through estimator Gumbel (1954); Jang et al. (2017); Maddison et al. (2017) during optimization. For each variable, we perturb the logits with independent Gumbel noise and apply a temperature-controlled softmax,

$$\pi_{i,j} = \frac{\exp((\boldsymbol{\theta}_{i,j} + g_{i,j})/\tau)}{\sum_{r \in \{0,1\}} \exp((\boldsymbol{\theta}_{i,r} + g_{i,r})/\tau)}, \qquad j \in \{0, 1\}. \tag{2}$$

The forward pass uses the hard assignment $\hat{x}_i = \arg\max_j \pi_{i,j}$ for clause evaluation, while the backward pass propagates gradients through the soft relaxation. This gives a GPU-friendly optimization loop that keeps the forward computation close to discrete SAT evaluation while retaining differentiability.

After optimization, we derive the **Phase** initialization directly from the optimized logits. With B batched optimization restarts, we average the optimized True-assignment probabilities and assign each variable its most confident Boolean value:

$$\phi_i = \mathbb{1}\left[\frac{1}{B}\sum_{b=1}^{B} \mathrm{softmax}(\boldsymbol{\theta}_i^{(b)})_1 \geq 0.5\right]. \tag{3}$$

The resulting phase vector $\boldsymbol{\phi}$ is loaded into the CDCL solver before search begins, replacing only the initial polarity state. All subsequent propagation, conflict analysis, clause learning, restarts, and termination behavior remain unchanged.

3.2 Gradient-Derived Saliency for VSIDS Initialization

Phase initialization determines which polarity CDCL tries first, but it does not change the order in which variables are selected. To initialize branching order, we derive a variable **Saliency Score** from the optimized differentiable MaxSAT objective and use it to seed VSIDS activity scores before search begins. The saliency score is designed to capture two complementary signals. The first is the global gradient magnitude $\|\partial\mathcal{L}/\partial\boldsymbol{\theta}_i\|_1$, which measures how strongly the relaxed objective changes with respect to variable x_i. The second is a local clause-sensitivity term derived from the product form in Eq. 1. For a literal ℓ in clause C_t, differentiating the relaxed clause satisfaction gives $\partial\,\mathrm{sat}(C_t)/\partial q_\ell = \prod_{\ell' \in C_t, \ell' \neq \ell}(1 - q_{\ell'})$. This term is large only when the other literals in the clause are unlikely to satisfy the clause, so it acts as a gate for variables that are locally decisive.

Let $\mathcal{C}(i)$ denote the set of clauses containing either x_i or $\neg x_i$. We define the saliency score

$$V_i = \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_i} \right\|_1 \cdot \sum_{C_t \in \mathcal{C}(i)} \left[\prod_{\ell' \in C_t,\, \mathrm{var}(\ell') \neq i} (1 - q_{\ell'}) \right]. \tag{4}$$

When batched optimization is used, V_i is averaged across the B optimized restarts. A variable receives high saliency when it has both a large global effect on the relaxed MaxSAT objective and a strong local role in clauses that are difficult to satisfy.

We use V_i to initialize VSIDS-style branching activity: the top-k variables by saliency receive boosted initial activity scores, causing them to be prioritized during early CDCL decisions. This gives the **Phase+VSIDS** variant. Because the summation in Eq. 4 can favor high-degree variables, we also evaluate a degree-normalized score,

$$V_i^{\mathrm{norm}} = V_i / d_i, \qquad d_i = |\mathcal{C}(i)|, \tag{5}$$

which gives the **Phase+VSIDS norm** variant when used for activity initialization.

3.3 PARALLEL PHASE PORTFOLIO

The same initialization interface extends naturally to parallel SAT solving. In the single-threaded setting, phase initialization uses one deterministic polarity vector derived from the optimized logits. In the parallel setting, we instead use the optimized logits as a distributional center and sample multiple phase assignments around it. This gives each CDCL worker a different initial polarity configuration while preserving the same underlying solver in every thread.

Let $\Theta^\star$ be the optimized logit matrix selected for portfolio generation. When using batched optimization, we choose the batch element with the lowest final optimization loss. For each worker $r \in \{1, \ldots, N\}$, we sample an independent Gumbel noise matrix $G^{(r)} \in \mathbb{R}^{n\times 2}$ and derive the phase assignment for each variable as follows:

$$P_i^{(r)} = \arg\max_{j\in\{0,1\}} \left(\boldsymbol{\theta}^\star_{i,j} + G^{(r)}_{i,j}\right), \qquad \mathcal{P} = \{P^{(1)}, \ldots, P^{(N)}\}. \tag{6}$$

The N phase vectors are loaded into N independent CDCL workers, each of which executes the standard solver with its assigned initial phase state. For satisfiable instances, the portfolio terminates as soon as any worker finds a satisfying assignment. Thus, a single GPU optimization pass is converted into a set of concurrent and diverse CDCL search trajectories, providing parallel diversification without modifying the CDCL procedure inside each worker.

4 EXPERIMENTS

Experimental setup We evaluate our initialization framework on benchmark instances from SAT Competition 2024 Heule et al. (2024). The single-threaded evaluation contains 368 instances, consisting of 167 satisfiable and 201 unsatisfiable formulas. We use newest Kissat (2025) Biere et al. (2024) on GitHub as the sequential baseline because recent Kissat-family solvers remain among the strongest single-threaded SAT solvers. Kissat-based entries ranked first and second in the SAT Competition 2025 Codel et al. (2025). Our experiments cover both major SAT-competition execution regimes: single-threaded solving and parallel portfolio solving. Each run is capped at $5{,}000$ seconds timeout (same as SAT Competition), and all reported runtimes are wall-clock times for both our methods and baselines. We report PAR-2, the SAT Competition ranking metric, where solved instances contribute their runtime and timeouts are penalized by twice the time limit. PAR-2 is useful here because it rewards faster solving while also reflecting the high cost of failing to solve within the allotted budget.

All single-threaded variants use the same underlying Kissat implementation and differ only in how the initial solver state is populated. We evaluate three initialization variants: **Phase**, which initializes the phase heuristic from the optimized logits; **Phase+VSIDS**, which additionally initializes variable activity scores using saliency; and **Phase+VSIDS norm**, which applies normalized saliency scores. For the parallel setting, we generate 32 phase assignments from the optimized logits and compare the resulting 32-worker portfolio against MallobSat Schreiber & Sanders (2024), the winner of the SAT Competition 2025 parallel track by PAR-2 Codel et al. (2025). We report the minimum runtime across the 32 phase workers, corresponding to portfolio wall-clock time when all workers are launched concurrently.

Optimization was performed using an NVIDIA RTX 4090 GPU with the Adam optimizer, a learning rate of 0.5, and a batch size of 3,000 for 10 gradient update steps per instance. The temperature parameter in Eq. 2 was set to $\tau = 1.0$. We run all experiments on AMD EPYC 7F72 CPU nodes clocked at 3.6 GHz, with a three-level cache hierarchy of 32 KB L1, 512 KB L2, and 16 MB L3 cache per core. Each node has 64 GB of RAM. All runtimes reported in this section are wall-clock times. For **VSIDS** score initialization, we select the top-k variables by saliency score V_i with $k = 50$ and assign them an initial activity score of $10{,}000$. GPU optimization runtime is marginal, averaging 1.7 seconds per instance, and is included in all reported total runtimes. Because this is negligible relative to the 5000-second cutoff, we do not report a separate runtime breakdown.

Single-threaded evaluations Table 1 summarizes the aggregate effect of the optimized initializations in the single-threaded setting, separated by SAT and UNSAT instances. All three variants improve PAR-2 in both categories. The gains are strongest on satisfiable instances. **Phase**, **Phase+VSIDS**, and **Phase+VSIDS norm** achieve SAT PAR-2 speedups of $1.12\times$, $1.08\times$, and

Table 1: PAR-2 comparison. *Our VBS* denotes the virtual best solver over our three proposed single-threaded variants.

(a) Single-threaded

Method	Status	Solved	PAR-2 (s)
Kissat	SAT	149	1421.28
Phase	SAT	150	**1271.28**
Phase+VSIDS	SAT	149	**1317.27**
Phase+VSIDS norm	SAT	152	**1204.13**
Our VBS	SAT	152	**918.13**
Kissat	UNSAT	147	3062.24
Phase	UNSAT	152	**2865.74**
Phase+VSIDS	UNSAT	151	**2880.58**
Phase+VSIDS norm	UNSAT	151	**2870.07**
Our VBS	UNSAT	153	**2710.08**

(b) Parallel

Method	Status	Solved	PAR-2 (s)
Phase (32T)	SAT	164	**425.34**
MallobSat (32T)	SAT	161	529.51
Phase (32T)	UNSAT	155	2679.62
MallobSat (32T)	UNSAT	168	**1930.88**

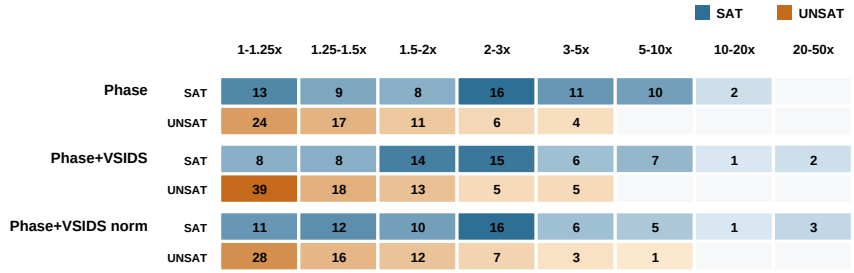

Figure 3: Distribution of improved-case speedups for single-threaded initialization variants, separated by SAT and UNSAT instances. Speedup is measured as Kissat runtime divided by variant runtime; only instances with speedup greater than $1\times$ are included. Cell values report instance counts, with blue denoting SAT and orange denoting UNSAT.

$1.18\times$ over Kissat, respectively. **Phase+VSIDS norm** is the strongest individual variant, while **Our VBS** reaches a $1.55\times$ SAT PAR-2 speedup and solves the largest number of SAT instances. On UNSAT instances, the improvements are more modest but consistent: the three variants obtain PAR-2 speedups of $1.07\times$, $1.06\times$, and $1.07\times$, with **Phase** providing the marginally best individual result. **Our VBS** further improves the UNSAT result, reaching a $1.13\times$ PAR-2 speedup and solving the largest number of UNSAT instances.

Figure 3 explains why these aggregate gains are meaningful even though no single initialization dominates all benchmarks. On SAT instances, **Phase**, **Phase+VSIDS**, and **Phase+VSIDS norm** improve over Kissat on 69, 61, and 64 cases, with median improved-case speedups of $2.20\times$, $2.04\times$, and $1.97\times$. The right tail is substantial: the 90th-percentile improved-case speedup is about $5.7\times$ for all three variants. On UNSAT instances, **Phase+VSIDS** improves the largest number of cases, accelerating 80 instances, while **Phase+VSIDS norm** shows the strongest median and tail behavior, with a median improved-case speedup of $1.36\times$ and a 90th-percentile speedup of $2.48\times$.

Overall, the single-threaded results show that optimized initialization does not make Kissat uniformly faster; rather, it changes the early CDCL trajectory in ways that are highly beneficial for a substantial subset of instances. This is reflected both in the PAR-2 improvements and in the VBS gap, which indicates that different initialization choices succeed on different instances. The speedup distributions further show a meaningful tail: some SAT instances obtain more than $10\times$ speedup, while UNSAT improvements are broader but lower in magnitude. This complementarity motivates the parallel portfolio evaluation in the next subsection. Beyond runtime and PAR-2, we also analyze timeout rescues in Section 4.1, as these capture the hardest cases where the baseline fails within the evaluation budget.

Parallel-version performance evaluations Based on the single-threaded results, we instantiate the parallel portfolio using only the **Phase** variant, which provides the most stable initialization signal across the evaluated settings. The same portfolio construction is applicable to **Phase+VSIDS** and **Phase+VSIDS norm**, but we leave their parallel evaluation to future work. As shown in Table 1, **Phase (32T)** performs strongly on SAT instances, solving more cases than MallobSat (32T) and

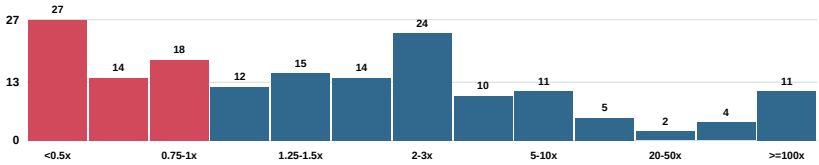

Figure 4: Direct speedup comparison between our 32-thread phase portfolio and 32-thread MallobSat on the 167 common SAT instances. The speedup ratio is Mallob wall-clock PAR-2 runtime divided by our portfolio runtime, so values above $1\times$ indicate that our method is faster. Red bars indicate bins where Mallob is faster, and blue bars indicate bins where our method is faster. Aggregate PAR-2 and solved-count results are reported in Table 1.

achieving a $1.24\times$ PAR-2 speedup. This shows that phase diversity generated from the optimized logits can translate into stronger wall-clock performance in the parallel SAT setting, rather than only improving isolated single-threaded runs. Figure 4 directly compares the two parallel solvers instance by instance on the SAT subset. The speedup ratio is Mallob wall-clock runtime divided by our 32-phase portfolio runtime, so values above $1\times$ indicate that our method is faster. Our portfolio is faster on 108 instances, MallobSat is faster on 59 instances, and one instance is tied. Thus, the SAT PAR-2 gain is not driven by only a small number of outliers; our method improves the majority of the benchmark set. The median finite speedup is $1.32\times$ in favor of our method, and the distribution also shows a nontrivial right tail where the optimized phase portfolio substantially outperforms MallobSat. Timeout rescues for parallel solving are analyzed separately in Section 4.1.

The UNSAT results highlight a limitation of the current parallel design. Our 32-thread portfolio diversifies phase assignments across independent Kissat workers, but it does not share learned clauses between workers. This is less effective for UNSAT instances, where proving unsatisfiability typically depends on accumulating and reusing conflict clauses rather than finding one satisfying trajectory. MallobSat benefits directly from clause sharing, which explains its stronger UNSAT PAR-2 performance in Table 1. In contrast, our **Phase (32T)** UNSAT result improves only modestly over the single-threaded variants, suggesting that future parallel extensions should combine differentiable phase diversity with clause-sharing mechanisms.

4.1 Timeout Rescues

Timeout rescues are a particularly stringent measure of solver improvement. They identify cases where the baseline search fails to return any answer within the evaluation budget, but an initialized variant succeeds. Under PAR-2, each timeout incurs a penalty of twice the time limit, so even a small number of rescues can have a visible effect on both solved count and aggregate score. More importantly, these instances probe the hardest part of the benchmark distribution for the baseline solver, where improvements are most consequential in practice.

Table 2 reports all instances for which Kissat times out and at least one of the evaluated methods solves the instance. In the single-threaded setting, our initialization variants rescue 3 SAT and 6 UNSAT instances, showing that the optimization-guided priors can redirect even a single CDCL run away from otherwise unsuccessful search trajectories. In the parallel setting, Kissat times out on 18 of the common SAT instances. Our 32-thread phase portfolio solves 15 of these cases, compared with 13 solved by MallobSat (32T), and the union of the two parallel methods solves 17 of the 18. This reinforces the main observation from the speedup distributions: optimized initialization is complementary to existing SAT search strategies, and its largest practical value appears on difficult instances where the baseline solver would otherwise timeout.

5 Conclusion and Limitation

We presented a differentiable initialization framework for CDCL SAT solvers. The method optimizes a GPU-based MaxSAT relaxation before search, maps the optimized logits to phase preferences, and uses gradient-derived saliency to initialize activity-based branching heuristics. The same interface extends to parallel solving by applying Gumbel perturbations to generate diverse

Table 2: Instances rescued from Kissat timeouts. The table lists instances for which Kissat times out and at least one listed method solves the instance. Bold text denotes our approach. In the parallel setting, T denotes solver threads.

Setting	Status	Kissat Timeout Instances	Rescued by	Best Runtime (s)
Single-thread	SAT	1-ET-512-K-96	**Phase+VSIDS norm**	2131.0
Single-thread	SAT	mdp-32-11-sat	**Phase**, **Phase+VSIDS norm**	3625.9
Single-thread	SAT	pcmax-scheduling-m40-26287-324155-SA	**Phase+VSIDS norm**	4958.0
Single-thread	UNSAT	tseitin_grid_n11_m20	**Phase+VSIDS norm**	1035.1
Single-thread	UNSAT	bvsub_19952.smt2	**Phase**, **Phase+VSIDS**, **Phase+VSIDS norm**	2169.8
Single-thread	UNSAT	Nb44T6	**Phase**, **Phase+VSIDS**, **Phase+VSIDS norm**	3197.4
Single-thread	UNSAT	test_v7_r17_vr5_c1_s25451.smt2-cvc4	**Phase**, **Phase+VSIDS**	3824.0
Single-thread	UNSAT	lec_mult_DvW_12x12	**Phase**, **Phase+VSIDS**, **Phase+VSIDS norm**	4410.9
Single-thread	UNSAT	pcmax-scheduling-m19-10199-62102-UNSAT	**Phase**	4719.1
Parallel	SAT	pcmax-scheduling-m40-26287-324155-SA	**Phase (32T)**, MallobSat (32T)	266.3
Parallel	SAT	1-ET-512-K-96	**Phase (32T)**, MallobSat (32T)	279.8
Parallel	SAT	Nb13T165	**Phase (32T)**, MallobSat (32T)	398.7
Parallel	SAT	sted5_0x1e3-20	**Phase (32T)**, MallobSat (32T)	409.1
Parallel	SAT	pcmax-scheduling-m12-8049-55035-SAT	**Phase (32T)**, MallobSat (32T)	719.3
Parallel	SAT	mdp-32-14-sat	**Phase (32T)**, MallobSat (32T)	786.1
Parallel	SAT	rbsat-v1150c84314gyes1	MallobSat (32T)	855.0
Parallel	SAT	pcmax-scheduling-m24-24102-255206-SA	**Phase (32T)**, MallobSat (32T)	857.6
Parallel	SAT	circuit_32in64out_with_150g	**Phase (32T)**, MallobSat (32T)	886.5
Parallel	SAT	pcmax-scheduling-m37-28831-324346-SA	**Phase (32T)**	887.4
Parallel	SAT	mdp-36-10-sat	**Phase (32T)**	1440.0
Parallel	SAT	mdp-32-11-sat	**Phase (32T)**, MallobSat (32T)	1443.0
Parallel	SAT	pcmax-scheduling-m35-32274-371389-SA	**Phase (32T)**, MallobSat (32T)	1704.2
Parallel	SAT	mdp-36-14-sat	**Phase (32T)**	2598.1
Parallel	SAT	Break_triple_20_36.xml	**Phase (32T)**, MallobSat (32T)	2911.4
Parallel	SAT	fermat-931960058139995587	MallobSat (32T)	3356.0
Parallel	SAT	apn-sbox5-cut3-symmbreak	**Phase (32T)**	4119.7

phase assignments for independent CDCL workers. On SAT Competition benchmarks, our best single-threaded variant achieves strong PAR-2 improvements over Kissat, while the VBS shows that different initialization choices expose complementary search trajectories. In the parallel setting, our 32-thread phase portfolio outperforms MallobSat(32T) on SAT instances, improving the majority of common SAT benchmarks and rescuing many Kissat timeouts.

The main limitation is on UNSAT instances. Phase diversity is effective for finding satisfying witnesses, but UNSAT solving relies on accumulating and reusing conflict clauses to build a proof. Our current parallel portfolio runs independent workers without clause sharing, while MallobSat directly benefits from shared learned clauses, explaining its stronger UNSAT PAR-2 performance. Although our method still rescues several UNSAT timeouts, future work should target UNSAT-oriented mechanisms such as learned clause prioritization, restart control, proof-guided activity initialization, or integration with clause-sharing parallel solvers.

REFERENCES

Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-SAT: An unsupervised differentiable approach. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BJxgz2R9t7.

Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh (eds.). *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2 edition, 2021.

Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. Cadical, gimsatul, isasat and kissat entering the sat competition 2024. *Proc. of SAT Competition*, pp. 8–10, 2024.

Cayden Codel, Katalin Fazekas, Marijn J. H. Heule, and Markus Iser. The results of sat competition 2025. SAT 2025 Conference, Glasgow, Scotland, 2025. URL https://satcompetition.github.io/2025/satcomp25slides.pdf.

Steve Dai, Cunxi Yu, Kalyan Krishnamani, and Brucek Khailany. Turbosat: Gradient-guided boolean satisfiability accelerated on gpu-cpu hybrid system. *arXiv preprint arXiv:2511.07737*, 2025.

Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960. doi: 10.1145/321033.321034.

Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962. doi: 10.1145/368273.368557.

Haonan Duan, Saeed Nejati, George Trimponias, Pascal Poupart, and Vijay Ganesh. Online bayesian moment matching based sat solver heuristics. In *International Conference on Machine Learning*, pp. 2710–2719. PMLR, 2020.

Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing – SAT 2003*, volume 2919 of *Lecture Notes in Computer Science*, pp. 502–518. Springer, 2004. doi: 10.1007/978-3-540-24605-3_37.

Jon William Freeman. *Improvements to propositional satisfiability search algorithms*. University of Pennsylvania, 1995.

Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1–2):67–100, 2000. doi: 10.1023/A:1006314320276.

Emil Julius Gumbel. *Statistical Theory of Extreme Values and Some Practical Applications*, volume 33. US Government Printing Office, Washington, DC, 1954.

Jesse Michael Han. Enhancing SAT solvers with glue variable predictions, 2020. URL https://arxiv.org/abs/2007.02559.

Marijn JH Heule, Markus Iser, Matti Järvisalo, and Martin Suda. Proceedings of sat competition 2024: Solver, benchmark and proof checker descriptions. 2024.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2017. URL https://arxiv.org/abs/1611.01144.

Robert G Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of mathematics and Artificial Intelligence*, 1(1):167–187, 1990.

Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Can q-learning with graph networks learn a generalizable branching heuristic for a sat solver? *Advances in Neural Information Processing Systems*, 33:9608–9621, 2020.

Jia Hui Liang, Vijay Ganesh, Ed Zulkoski, Atulan Zaman, and Krzysztof Czarnecki. Understanding vsids branching heuristics in conflict-driven clause-learning sat solvers, 2015. URL https://arxiv.org/abs/1506.08905.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables, 2017. URL https://arxiv.org/abs/1611.00712.

João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999. doi: 10.1109/12.769433.

Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pp. 530–535, 2001.

Muhammad Osama, Anton Wijs, and Armin Biere. Certified sat solving with gpu accelerated inprocessing. *Formal Methods in System Design*, 62(1):79–118, 2024.

Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Theory and Applications of Satisfiability Testing – SAT 2007*, volume 4501 of *Lecture Notes in Computer Science*, pp. 294–299. Springer, 2007. doi: 10.1007/978-3-540-72788-0_28.

Nicolas Prevot. Gpusharesat: a sat solver using the gpu for clause sharing. *arXiv preprint arXiv:2012.03119*, 2020.

Dominik Schreiber and Peter Sanders. Mallobsat: Scalable sat solving by clause sharing. *Journal of Artificial Intelligence Research*, 80:1437–1495, 2024.

Daniel Selsam and Nikolaj Bjørner. Neurocore: Guiding high-performance sat solvers with unsat-core predictions. *CoRR*, abs/1903.04671, 2019.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJMC_iA5tm.

Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6545–6554. PMLR, 2019. URL https://proceedings.mlr.press/v97/wang19e.html.

Wenxi Wang, Yang Hu, Mohit Tiwari, Sarfraz Khurshid, Kenneth McMillan, and Risto Miikkulainen. Neuroback: Improving cdcl sat solving using graph neural networks. In *The Twelfth International Conference on Learning Representations*, 2024.

A Additional Evaluation

A.1 Comparison with Non-Neural Initialization Baselines

We compare our method against several non-neural initialization baselines: Jeroslow–Wang (JW) (Jeroslow & Wang, 1990), Maximum Occurrences in Minimum-size Clauses (MOMS) (Freeman, 1995), variable degree, Kissat's `walkinitially` option, and Bayesian Moment Matching (BMM) (Duan et al., 2020). In the tables, **Ours** denotes the Phase+VSIDS-Norm configuration.

We evaluate these methods on 111 challenging instances from SAT Competition 2024, selected according to the default Kissat runtime. The subset contains 18 SAT instances on which Kissat timed out, the next 20 SAT instances with the longest runtimes, 51 UNSAT instances on which Kissat timed out, and the next 22 UNSAT instances with the longest runtimes. All reported runtimes include the time required to generate the corresponding initialization.

Table 3 reports the PAR-2 scores and timeout counts for all initialization methods. Our method achieves the lowest PAR-2 and the fewest timeouts on the full subset and on both the SAT and UNSAT splits. Degree and BMM are competitive on SAT instances, while MOMS and BMM perform relatively well on UNSAT instances, but none matches our method consistently across both categories.

The average initialization and preprocessing times are shown in Table 4. Most methods introduce little overhead relative to the solver runtime on these challenging instances. BMM is noticeably more expensive because its initialization is implemented using CPU-only computation.

Table 3: Comparison with non-neural initialization baselines on 111 challenging SAT Competition 2024 instances. PAR-2 includes initialization and preprocessing time.

Metric	Kissat	Ours	JW	MOMS	`walkinitially`	Degree	BMM
All PAR-2	6931.55	**6670.02**	7030.08	7119.43	6934.41	6880.43	6814.59
All TOs	69	**66**	72	72	69	70	68
SAT PAR-2	5563.71	**5221.08**	5617.03	6312.92	5567.84	5261.26	5421.60
SAT TOs	18	**17**	20	22	18	18	18
UNSAT PAR-2	7643.58	**7424.27**	7765.64	7539.26	7645.78	7723.28	7539.71
UNSAT TOs	51	**49**	52	50	51	52	50

Table 4: Average initialization and preprocessing time in seconds.

Split	Kissat	Ours	JW	MOMS	`walkinitially`	Degree	BMM
All	0.00	0.32	0.54	0.48	0.00	0.43	28.50
SAT	0.00	0.24	0.40	0.39	0.00	0.32	22.18
UNSAT	0.00	0.36	0.61	0.52	0.00	0.48	31.78

A.2 Hyperparameter Sensitivity

We evaluate the sensitivity of the two hyperparameters that directly control CDCL initialization: top-k, the number of variables selected for initialization, and the activity boost assigned to those variables in the VSIDS heuristic. The evaluation uses a 111-instance subset consisting of 38 SAT and 73 UNSAT instances. Each configuration is denoted by *top-k/activity boost*.

To isolate the effects of the two CDCL-facing hyperparameters, this analysis excludes initialization-generation time. All configurations use the same GPU initialization pipeline, and the purpose of this experiment is to isolate the effect of the two CDCL-facing hyperparameters.

Table 5: Sensitivity to top-k and VSIDS activity boost on 111 instances. Each column denotes *top-k/activity boost*. Bold values indicate the best result in each row. Initialization-generation time is excluded.

Metric	$50/10{,}000$	$25/10{,}000$	$100/10{,}000$	$50/5{,}000$	$50/20{,}000$
Combined PAR-2	**6669.70**	7017.74	6957.16	6845.66	6891.06
Combined TOs	**66**	71	69	68	70
SAT PAR-2	**5220.84**	5631.13	6073.29	5664.82	5969.21
SAT TOs	**17**	19	21	19	21
UNSAT PAR-2	7423.90	7739.53	7417.26	7460.34	**7370.93**
UNSAT TOs	49	52	**48**	49	49

The default configuration, $50/10{,}000$, achieves the lowest combined PAR-2 and the fewest combined timeouts. It also gives the best SAT PAR-2 and SAT timeout count. Some alternative configurations provide small improvements on individual UNSAT metrics: $100/10{,}000$ has one fewer UNSAT timeout, while $50/20{,}000$ achieves the lowest UNSAT PAR-2. However, these configurations substantially degrade SAT performance and therefore perform worse on the combined set.

Overall, the results support the default hyperparameter configuration used in the main experiments. Although performance varies with top-k and activity boost, the selected configuration provides the strongest balance across SAT and UNSAT instances rather than optimizing either subset in isolation.