Scalable Floating-Point Satisfiability via Staged Optimization

YUANZHUO ZHANG, Virginia Tech, USA
ZHOULAI FU, SUNY Korea, Republic of Korea, Virginia Tech, USA, and Stony Brook University, USA
BINOY RAVINDRAN, Virginia Tech, USA

This work introduces *StageSAT*, a new approach to solving floating-point satisfiability that bridges SMT solving with numerical optimization. StageSAT reframes a floating-point formula as a series of optimization problems in three stages, each with increasing precision. It begins with a fast, projection-aided descent objective to efficiently guide the search toward a feasible region, then proceeds to bit-level accuracy with units-in-the-last-place (ULP)^2 optimization and a final n-ULP lattice refinement to ensure correctness. By construction, the final two stages use a representing function that evaluates to zero if and only if a candidate satisfies all constraints. Thus, whenever optimization drives the bit-precise objective to zero, the resulting assignment is a valid solution, providing a built-in guarantee of soundness (no spurious SAT results). To further improve the search, StageSAT introduces a partial monotone descent property on linear constraints via an orthogonal projection technique, which prevents the optimizer from stalling on flat or misleading objective landscapes. Critically, this solver requires no heavy bit-level reasoning or specialized abstractions of floating-point arithmetic; it treats complex arithmetic as a black box, using runtime evaluations to navigate the input space.

We implement StageSAT and evaluate it on extensive benchmarks, including the SMT-COMP'25 floating-point suites and difficult cases from prior work. In our experiments, StageSAT proved both more scalable and more accurate than state-of-the-art optimization-based alternatives. It solved strictly more formulas than any competing solver under the same time budget – in fact, StageSAT found most of the satisfiable instances in our benchmarks and never produced a spurious model for an unsatisfiable formula. This amounts to 99.4% recall on satisfiable cases with 0% false SAT in our benchmarks, exceeding the reliability of prior optimization-based solvers we tested. StageSAT also delivered significant speedups (often 5–10× faster) over traditional bit-precise SMT solvers and earlier numeric solvers. These results demonstrate that our staged optimization strategy can significantly improve both the performance and correctness of floating-point satisfiability solving.

CCS Concepts: • **Mathematics of computing → Solvers**.

Additional Key Words and Phrases: Floating-Point Constraint Solving, Mathematical Optimizations

ACM Reference Format:
Yuanzhuo Zhang, Zhoulai Fu, and Binoy Ravindran. 2026. Scalable Floating-Point Satisfiability via Staged Optimization. *Proc. ACM Program. Lang.* 10, PLDI, Article 264 (June 2026), 23 pages. https://doi.org/10.1145/3808342

1 Introduction

Floating-point arithmetic is widely used in safety-critical software such as numerical libraries and industrial control software. However, its subtle semantics, such as rounding, special values, and non-associativity, can make reasoning about such programs difficult. Tools that verify floating-point programs, such as bounded model checkers and deductive verifiers, need to answer satisfiability

Authors' Contact Information: Yuanzhuo Zhang, Virginia Tech, Blacksburg, USA, yuanzhuo@vt.edu; Zhoulai Fu, SUNY Korea, Incheon, Republic of Korea and Virginia Tech, Blacksburg, USA and Stony Brook University, New York, USA, zhoulai.fu@sunykorea.ac.kr; Binoy Ravindran, Virginia Tech, Blacksburg, USA, binoy@vt.edu.

ACM 2475-1421/2026/6-ART264
https://doi.org/10.1145/3808342

queries over floating-point constraints. Automated verification of floating-point software relies on Satisfiability Modulo Theory (SMT) solvers capable of deciding formulas over IEEE 754 Floating-Point Satisfiability Modulo Theories (FP-SMT). SMT solving is therefore a key enabler for bringing the rigor of formal methods to the vast body of software that computes over the reals.

Optimization-Based Floating-Point Solving: Traditional SMT solvers for floating-point (FP) arithmetic often struggle with scalability, especially on formulas with non-linear or transcendental operations. An alternative approach reduces a floating-point satisfiability problem to a numerical optimization problem. In this paradigm, a floating-point formula (a set of constraints) is transformed into a numeric objective function R(x) that acts as a *distance to satisfaction*. This objective is non-negative and evaluates to zero if and only if the candidate assignment x is a true solution (i.e., satisfies all constraints). Deciding satisfiability then becomes an optimization task: minimize R(x) and check whether the minimum value reaches zero. Pioneering solvers like XSat [12], goSAT [3], and Grater [6] demonstrated the promise of this strategy. The key appeal of these *optimization-based* solvers is that they can treat complex arithmetic as black-box functions—they do not need to bit-blast or explicitly enumerate the intricacies of IEEE-754 semantics. Instead, the solver is guided by numeric feedback from R(x): as the assignment x moves closer to satisfying the formula, R(x) decreases, and a solution is found when R(x) hits 0. This enables reasoning about arbitrary mathematical functions (e.g., transcendentals like sin or cos) as long as those functions can be evaluated, a notable advantage over bit-blasting SMT methods.

Challenges: Early optimization-based FP solvers revealed two key challenges. First, the objective functions resulting from this reduction can be irregular and non-smooth, which hurts runtime performance and can mislead numeric search into local minima or flat plateaus. In practice, the solver might fail to find a solution even when one exists, simply because the search gets "stuck" in a region where the objective doesn't meaningfully decrease. For example, XSat has been observed to mislabel satisfiable instances as UNSAT due to inadequate minimization; Grater avoids false-UNSAT errors but often returns *timeout* results, some of which are later confirmed satisfiable.

Second, finite precision of floating-point arithmetic and coarse step adjustments can undermine solver correctness and force a trade-off with solver efficiency. Consequently, optimization-based solvers like Grater employ small tolerance thresholds to decide when a formula is satisfied [5], declaring success if $R(x) < tol$, even though mathematically only $R(x) = 0$ corresponds to a true model. However, an overly loose tolerance can yield spurious solutions, e.g. erroneously accepting $x \approx 0$ as satisfying a constraint like $x > 0 \wedge x \leq$ 1e-160. Conversely, XSat eschews tolerances entirely and only accepts a candidate when $R(x) = 0$ exactly, but it can still miss a valid solution if its step-size policy cannot nudge a candidate from $x = 0$ to a tiny non-zero value, e.g., x = 1e-200.

These challenges mean that, in practice, earlier optimization-based solutions often proved *unscalable* and *inaccurate*. They might label SAT formulas as UNSAT (or unknown) because the global minimum of a jagged objective couldn't be found, and conversely label UNSAT formulas as SAT due to floating-point error and insufficient bit precision. Notably, XSat partially addressed the precision issue by measuring constraint violations in terms of *ULP [15]*—the smallest difference between two representable FP numbers—thus bringing bit-level rigor to its objective. However, a purely ULP-based objective landscape is highly discontinuous and difficult to search, contributing to the scalability problems.

Our Approach: StageSAT. We present StageSAT, a new floating-point SMT solver that addresses these accuracy and scalability issues via *staged optimization*. The central idea is to deliberately engineer the objective function and search process in multiple phases, combining the smoothness needed for effective global search with the precision needed for correctness. StageSAT aims to be *both* robust on large benchmarks *and* precise in its results—reporting SAT only when a correct

model is found and reporting UNSAT-GUESS (i.e., a heuristic UNSAT result without proof) only after exhaustive search attempts in the final stage suggest no model is likely to exist.

StageSAT's design integrates several key techniques to achieve this balance:

- **Multi-Staged Objective Design:** Instead of a single monolithic objective, StageSAT incrementally refines its objective across three stages. It begins with smooth squared-error terms to quickly guide the search toward a promising region. It then transitions to a more precise ULP-level objective measure to target bit-level correctness. Finally, it applies an *n-ULP margin refinement [13]* in the last stage, progressively tightening the allowed error margin down to zero. This staged approach preserves searchability in early phases while achieving bit-precise satisfaction—StageSAT only reports "SAT" when the ULP objective truly reaches zero under IEEE-754 semantics.
- **Orthogonal Projection:** To mitigate misleading gradients in regions with linear equalities or flat surfaces, StageSAT employs an orthogonal projection technique [16]. Essentially, when optimizing constraints like `a·x + b = c` that form flat "valleys" in the objective landscape, StageSAT projects the search step orthogonally onto the constraint surface to better estimate the true distance to satisfaction. This improves the alignment between movements in the input space and decreases in the objective, acting as a practical safeguard against stalls and false convergence on plateaus.
- **Objective Smoothing:** StageSAT applies lightweight smoothing to the objective in each stage (for example, using squared forms of error terms or tiny perturbations) to reduce erratic behavior. By "rounding off" sharp corners and eliminating zero-gradient traps early on, this smoothing makes the objective landscape easier to navigate, leading the numerical optimizer more reliably toward a global minimum.

In theory, StageSAT is *sound* for satisfiability: it will report SAT only when a candidate solution passes a rigorous, bit-level check (R(x) = 0 exactly, meaning the assignment is a valid model). For cases where the optimizer converges to a non-zero minimum, StageSAT concludes the formula is unsatisfiable (albeit without a formal proof). While this UNSAT outcome is a heuristic "confident guess" (since StageSAT, like other numeric solvers, is incomplete), it was empirically accurate in all our evaluations. If StageSAT fails to find a solution within a given time limit, it returns *timeout*.

Results: In practice, StageSAT proves both accurate and scalable. We evaluated StageSAT on the SMT-COMP'25 [22] Quantifier-Free Floating-Point (QF_FP) benchmarks [28], encompassing the rigorous suites contributed by MathSAT developers. We classify the suites into "small/middle/large" categories based on instance size. We also include the sets of benchmarks used by Grater and JFS. StageSAT was compared against four bit-precise SMT solvers (Z3 [9], MathSAT [7], Bitwuzla [24], cvc5 [2]) and four optimization-based solvers (XSat [12], goSAT [3], Grater [6], JFS [20]). The results demonstrate significant advantages for StageSAT:

- On the "middle" benchmarks (35 formulas), StageSAT solved *all* 35 instances (SAT and UNSAT) with a median runtime of just 0.18 seconds.
- On the more challenging "large" benchmarks, StageSAT solved 24 of 26 SAT instances and returned unsat-guess on 21 UNSAT instances (matching known ground truth results). In total, StageSAT achieved the highest coverage, solving strictly more formulas than any competing solver due to its low timeout rate.
- Compared to the best SMT solvers (e.g., Bitwuzla and cvc5), StageSAT achieved up to *10× faster* median runtime on satisfiable cases and avoided many timeouts, solving several formulas that those solvers could not.
- Compared to prior optimization-based solvers, StageSAT showed significantly improved reliability and performance. It attained *99.4% SAT recall* (missing only two satisfiable instances

out of 347) and *0% false SAT* on unsatisfiable cases (never reporting a spurious model), whereas tools like goSAT and Grater often reported *unknown* or suffered misclassifications. StageSAT was also about an order of magnitude faster on average than XSat, goSAT, and Grater in our experiments.

To understand the impact of each design decision, we performed an ablation study on StageSAT's components (objective staging, orthogonal projection, and smoothing). Removing any one component significantly degraded performance or accuracy, confirming that each plays a critical role in the solver's effectiveness. Although StageSAT remains incomplete (like its predecessors, it cannot produce formal UNSAT proofs), our results illustrate that numerical optimization can be a powerful complement to traditional SMT solving—especially for difficult floating-point formulas where scale and runtime are paramount.

Contributions. In summary, this paper makes the following contributions:

- **ULP-Staged Objective Design:** A multi-stage objective strategy that moves from a smooth squared-distance metric to a squared-ULP-accurate metric and finally an n-ULP refinement. This design achieves robust global search in the early phase *and* bit-precise SAT detection in subsequent phases, addressing both scalability and floating-point precision issues in SMT solving.
- **Orthogonal Projection for Improved Descent:** A technique that applies orthogonal projection on linear constraints during optimization. This improves monotonic descent towards satisfying assignments and helps the solver avoid stagnation on misleading objective landscapes.
- **Smoothing for Stability:** The use of lightweight objective smoothing (e.g., squaring error terms) to improve convergence in regions with discontinuities or zero-gradient traps. Smoothing yields a more stable optimization process, increasing reliability and speed.
- **Extensive Evaluation:** A thorough experimental evaluation on SMT-COMP'25 QF_FP benchmarks demonstrating that StageSAT outperforms prior incomplete solvers in both the number of benchmarks solved and overall runtime. StageSAT also complements the complete solvers well by producing consistent results in much less time. We also present an ablation study confirming the necessity of each proposed component.

2 Overview and an Illustrative Example

We use the following notation throughout this paper. We consider quantifier-free floating-point (QF_FP) formulas. Terms are built from FP variables and constants using standard IEEE-754 arithmetic; atoms have the form e_1 op e_2 with op $\in \{==, \leq, <, \geq, >\}$. Mixed precision (FP32/FP64) is allowed; each atom is evaluated in its declared format.

(1) C is a CNF formula (a conjunction of clauses ϕ, each a disjunction of literals) over FP variables $\vec{x}$. We write $\vec{x} \models C$ when all clauses evaluate to true under IEEE-754 at $\vec{x}$, and define the model set $\mathcal{M} = \{\vec{x} : \vec{x} \models C\}$. We write $\sum_{\phi \in C}(\cdot)$ for summation over clauses;
(2) For $S \subseteq \mathbb{R}^d$, $d(\vec{x}, S) := \inf_{\vec{y} \in S} \|\vec{x} - \vec{y}\|_2$ is the Euclidean distance from $\vec{x}$ to a set S. When S is an affine set, the infimum is attained at the orthogonal projection of $\vec{x}$ onto S;
(3) $\mathcal{L} = \{\vec{x} \mid A\vec{x} == \vec{b}\}$ denotes the top-level linear equalities from C, and $\mathcal{N}$ denotes all remaining atoms (other linear equalities, linear inequalities, and non-linear constraints);
(4) R is a *representing function* for C if $R(\vec{x}) \geq 0$ and $R(\vec{x}) = 0 \iff \vec{x} \models C$;
(5) $A^\dagger$ is the Moore–Penrose pseudoinverse of A;
(6) $\mathrm{ULP}(a, b)$ counts the number of representable IEEE-754 values between floats a and b (zero iff $a == b$ bit-for-bit);

(7) nULP(k, z) shifts a float z by k ULPs on the FP lattice (positive: `IEEE nextUp`; negative: `IEEE nextDown`).

Scope and Goal. We target the QF_FP theory. Given a CNF formula C over IEEE-754 atoms, our aim is a practically accurate satisfiability outcome: (1) if C is satisfiable, we return **sat** with a witnessing assignment $\vec{x}^\star$ that validates under IEEE-754; (2) if optimization terminates with a strictly positive minimum, we return **unsat-guess**—not a proof, but a confident verdict supported by our evaluation; and (3) if the time budget expires without a decision, we return **timeout**.

Three-Stage Overview. The following paragraphs sketch StageSAT's design choices. The exact execution of this staged architecture is formalized in Section 3.5, which serves as the formal spine of the StageSAT procedure.

- **S1 — projection-aided squared objective (fast descent).**
  We start with a fast, albeit coarse optimization toward feasibility. S1 serves as an initialization phase that uses a smooth objective coupled with a projection on $\mathcal{L}$.
  We separate constraints C into $\mathcal{L}$ and $\mathcal{N}$, and build an objective of the form $R_{\mathcal{L}} + R_{\mathcal{N}}$. Here $R_{\mathcal{N}}$ is a sum of *squared* residuals/violations (as in traditional optimization-based encodings), while $R_{\mathcal{L}}$ uses an *orthogonal projection* onto the linear manifold to compute distance exactly. The projection fixes the non-monotone behavior illustrated in the example below, yielding a *partial* monotone-descent effect and a robust early trajectory.
- **S2 — ULP^2 objective (bit-level alignment).**
  We then switch to a bit-precise *squared ULP distance* per constraint to strictly align the objective with IEEE-754 semantics. Squaring shapes the penalty (amplifies large gaps, breaks ties), providing a stable coarse-to-fine descent and exposing bit-level issues (e.g., subnormal underflow) that magnitude-based S1 can miss.
- **S3 — n-ULP refinement over the floating-point lattice.**
  Finally, we perform a *discrete* search around the S2 incumbent on the FP lattice to *snap* near-misses to exact zeros that continuous search cannot cross.

Example. Our design principle is intuitive: as the objective decreases, the assignment should move *closer* to the model set $\mathcal{M} = \{\vec{x} : \vec{x} \models C\}$. If we could compute the Euclidean distance $d(\vec{x}, \mathcal{M})$ to the model set, then minimizing this function to zero would solve the satisfiability problem directly and would automatically satisfy our design principle. In practice, existing optimization-based solvers replace this ideal yet intractable distance with a proxy $R(\vec{x})$ that satisfies two basic properties: (i) $R(\vec{x}) \geq 0$ and (ii) $R(\vec{x}) = 0 \Rightarrow \vec{x} \models C$. Such proxies enable search but need not align with geometric distance; as our toy example shows, they can violate the intended "closer $\implies$ smaller" behavior and mislead optimization.

Our key observation is that for the linear-equality portion of a constraint, $\mathcal{L} = \{\vec{x} : A\vec{x} == \vec{b}\}$, we can compute the exact squared distance to the solution manifold without solving the equations: orthogonally project $\vec{x}$ to $\mathcal{L}$ (via the Moore–Penrose pseudoinverse [27]) and measure the squared gap to that projection. The Moore–Penrose pseudoinverse generalizes the matrix inverse: for a matrix A, the pseudoinverse $A^\dagger$ is the unique matrix that yields the minimum-norm least-squares solution to $A\vec{x} == \vec{b}$ (see Section 3.2 for the formal projection definition and closed form). This gives a closed-form "distance-to-$\mathcal{L}$" term that directly enforces the Monotone Descent principle on the linear part.

Our Stage 1 combines this geometric term with a simple representing function for the remaining atoms $\mathcal{N}$ (non-linear and/or inequalities):

$$S_1(\vec{x}) = d(\vec{x}, \mathcal{L})^2 + R_{\mathcal{N}}(\vec{x}).$$

Here $R_{\mathcal{N}} \geq 0$ and $R_{\mathcal{N}}(\vec{x}) = 0$ iff all atoms in $\mathcal{N}$ hold at $\vec{x}$. As a result, Stage 1 guarantees that any strict decrease in the objective **necessarily reduces the distance to** $\mathcal{L}$. Because both the projection and the evaluation are performed in floating point, Stage 1 is used to *guide search* rather than to certify SAT.

Here we show how a naive choice breaks the principle (monotone descent) and how projection fixes it, then explain why each stage is needed for efficient IEEE-faithful decisions.

From f_{naive} to S_1: Why Projection. Toy constraint. $x == 1 \ \wedge \ y == x$. The unique model is $(1, 1)$; the linear manifold is the single point $(1, 1)$.

Naive objective and its failure. A natural first attempt is

$$f_{\text{naive}}(x, y) \ = \ (x - 1)^2 \ + \ (y - x)^2.$$

It measures residuals of the two equations, but it does *not* align with our monotone-descent intent: moving from $(2, 2)$ to $(2, 1)$ gets *closer* to $(1, 1)$ in Euclidean distance, yet f_{naive} *increases*. Intuitively, $(y - x)^2$ penalizes motion along the manifold's tangent direction, so the objective can go up even as the point approaches the model.

Point (x, y)	**Distance to** $(1, 1)$	$\mathbf{f_{naive}(x, y)}$
(2, 2)	1.414	1.000
(2, 1)	**1.000**	**2.000**
(1, 1)	0.000	0.000
(0, 1)	**1.000**	**2.000**
(0, 0)	1.414	1.000

Projection: make the proxy be the distance itself. On $\mathcal{L}$, we measure *distance to the feasible set* rather than residuals to its defining equations. Concretely for this toy, the orthogonal projection of any (x, y) onto the linear manifold is $(1, 1)$. The Stage 1 objective becomes

$$\boxed{S_1(x, y) \ = \ (x - 1)^2 \ + \ (y - 1)^2.}$$

Because S_1 is (by construction) the squared distance to the linear solution set, any *strict decrease* in S_1 necessarily means the assignment moved *closer* to that set. This eliminates the tangential artifact in f_{naive} and restores predictable descent. While this toy example demonstrates the geometric intuition visually, Section 3.2 formally defines this projection-aided objective and proves the Partial Monotone Descent Property (Lemma 2) that guarantees this predictable trajectory.

Why S1 does not declare SAT. S1 is a *search* stage. The projection and the distance are computed in floating point, so a tiny near-zero can be a rounding artifact. Even when S_1 is numerically small, we *do not* certify SAT at S1. Instead, S1 supplies a strong initializer for the bit-precise stages:

- *S2* uses pairwise ULP^2 to decide SAT *exactly* when the objective reaches zero.
- *S3* performs a tiny n-ULP lattice refinement around S2's incumbent, snapping near-misses to exact equality (SAT), else producing *unsat-guess* or *timeout*.

Stage 2 (S2): Squared-ULP Objective — A Bit-Precise Representing Function. After S1 brings us near feasibility, Stage 2 switches to a *bit-level* view aligned with IEEE-754.

- *Equality literal* $f(\vec{x})==g(\vec{x})$: distance is ULP$(f(\vec{x}), g(\vec{x}))$.
- *Inequality literal* $f(\vec{x}) \leq g(\vec{x})$: use 0 when the inequality holds; otherwise, the minimal ULP steps needed to make it true (with the standard ±1 correction for strictness).
- *Clause* (disjunction): multiply the *squared* ULP distances of its literals so that any satisfied literal zeros the product; the full S2 objective *sums products over clauses*.

For the toy (each equality is its own clause), the *concrete* Stage 2 objective is

$$\boxed{S_2(x, y) \;=\; \mathrm{ULP}(x, 1)^2 \;+\; \mathrm{ULP}(y, x)^2.}$$

By construction, $S_2(\vec{x}) \geq 0$ and $S_2(\vec{x}) = 0$ *iff* all constraints hold *exactly* in IEEE-754; i.e., S_2 is a *representing function* for C. We formally define this property in Lemma 3 (Section 3.3). Consequently, *S2 can declare SAT* when its minimum reaches 0.

Near-miss → exact. Suppose S1 yields (x, y) with x one ULP above 1 and y two ULPs above x. Then $S_2 = 1^2 + 2^2 = 5 > 0$. S2 continues to adjust (x, y) until both gaps are zero; on this toy it typically reaches $(1, 1)$, achieving $S_2(1, 1) = 0$ and reporting *SAT*.

Why projection is not applied in S2. Projection onto the linear manifold is a real-valued geometric operation that aligns descent with the true distance to the feasible set, improving the smooth landscape of S1. However, this projection requires computing the Moore–Penrose pseudoinverse, which is not a bit-precise transformation under IEEE-754 arithmetic. S2 must operate on the original, unmodified constraints to preserve the representing-function guarantee that the objective is zero *if and only if* all constraints hold exactly under IEEE-754 semantics.

Stage 3 (S3): n-ULP Refinement — Discrete Search for the Last Bit. Even with S2 ULP-aware optimization, a continuous method can stall a *few ULPs* from equality—especially near subnormals or in tight chains of equalities—because continuous steps cannot "snap" across the final discrete gap. *Stage 3* resolves this via a tiny *discrete search* on the FP lattice around S2's incumbent (x_2^*, y_2^*).

We evaluate the same ULP penalties at stepped points and minimize over tiny integer offsets m (for x) and n (for y):

$$\boxed{S_3(m, n) \;=\; \mathrm{ULP}\big(\mathrm{nULP}(m, x_2^*),\, 1\big)^2 \;+\; \mathrm{ULP}\big(\mathrm{nULP}(n, y_2^*),\, \mathrm{nULP}(m, x_2^*)\big)^2.}$$

If the best value over a small neighborhood hits 0, S3 *declares SAT* with the corresponding stepped assignment; if the minimum remains *strictly positive*, we return *unsat-guess* (not a proof but empirically highly accurate); if the time budget elapses first, *timeout*.

Why S3 is necessary (intuition). In a 20-variable chain $x_2{=}x_1, x_3{=}x_2, \ldots, x_{20}{=}x_{19}$ anchored at a tiny $x_1{=}c$, S1 pulls toward the affine manifold and S2 aligns with bit-level equality, but one or two coordinates can remain a couple of ULPs off—continuous updates disturb neighbors or overshoot at these scales. S3 jointly tweaks last bits and snaps the whole vector to the exact lattice point, driving the objective to 0 when a model exists. While this illustrates the discrete lattice search intuitively, Section 3.4 provides the formal definition of this n-ULP refinement and proves in Lemma 4 that S_3 preserves the representing function contract at these discrete steps.

Takeaway. Replacing f_{naive} with the *projection-based distance* S_1 restores the intended "closer $\Longrightarrow$ smaller" geometry on $\mathcal{L}$ and yields a robust initializer. *S2* contributes an *IEEE-faithful* ULP^2 *representing function* that certifies SAT when it reaches zero. *S3* adds a small n-ULP *lattice refinement* to bridge the last-bit gap when smooth optimization stalls. Together, the pipeline $S_1 \to S_2 \to S_3$ provides predictable progress, exact satisfiability decisions when possible, and principled *unsat-guess* or *timeout* otherwise.

3 Technical Approach and Theory

This section makes precise how StageSAT works and why it is effective. We formalize the three stages and establish three facts: (i) **Stage 1** satisfies a **partial monotone descent** guarantee on the **linear** part; (ii) **Stage 2** and **Stage 3** are **representing functions** for the constraint set; and (iii) the overall procedure is **sound** (no false SAT), while necessarily **incomplete**. These results explain StageSAT's **accuracy** and **scalability** in practice.

3.1 Notation

All notation used in this section—including the QF_FP language, CNF structure, model set, Euclidean distance, and key primitives such as ULP and nULP—is defined in the notation block at the beginning of Section 2.

We keep formal objective definitions local to each stage to avoid redundancy and to highlight their distinct roles.

3.2 Stage 1 (S1): Projection-Aided Square Objective and Partial Monotone Descent

Recall from Section 2 that $\mathcal{L}$ denotes the top-level linear equalities and $\mathcal{N}$ the remaining atoms of C.

Square Distance for Literals and Clauses (Multiplication in $\vee$). For a literal ℓ:

- equality $h(\vec{x}) == 0$: $\mathrm{dist}_{\square}(\ell;\vec{x}) := h(\vec{x})^2$;
- inequality $f(\vec{x}) \le g(\vec{x})$: $\mathrm{dist}_{\square}(\ell;\vec{x}) := \max\{0,\, f(\vec{x}) - g(\vec{x})\}^2$.

For a *clause* ϕ (a disjunction of literals), we encode $\vee$ by *multiplication*:

$$\mathrm{Dist}_{\square}(\phi;\vec{x}) \;:=\; \prod_{\ell\in\phi} \mathrm{dist}_{\square}(\ell;\vec{x}),$$

so if *any* literal in ϕ is satisfied, one factor is 0 and the clause distance is 0 (as in XSat). Let $C_{\mathcal{N}}$ be the set of clauses formed only from atoms in $\mathcal{N}$. The *S1 objective* is

$$S_1(\vec{x}) \;=\; \underbrace{\|\vec{x} - \Pi_{\mathcal{L}}(\vec{x})\|_2^2}_{\text{exact squared distance to } \mathcal{L}} \;+\; \sum_{\phi\in C_{\mathcal{N}}} \mathrm{Dist}_{\square}(\phi;\vec{x}).$$

Projection (Definition and Closed Form).

$$\Pi_{\mathcal{L}}(\vec{x}) \;=\; \vec{x} - A^{\top}\big(AA^{\top}\big)^{\dagger}(A\vec{x} - \vec{b}), \qquad \|\vec{x} - \Pi_{\mathcal{L}}(\vec{x})\|_2^2 \;=\; \Big\|A^{\top}\big(AA^{\top}\big)^{\dagger}(A\vec{x} - \vec{b})\Big\|_2^2,$$

with $(\cdot)^{\dagger}$ the Moore–Penrose pseudoinverse [27].

Monotone Descent Property (MDP). For an objective R, we say *R satisfies monotone descent for C* if

$$\boxed{R(\vec{x}') \;<\; R(\vec{x}) \implies d(\vec{x}',\, \mathcal{M}_C) \;<\; d(\vec{x},\, \mathcal{M}_C).}$$

Lemma 1 (Projection Facts). $\Pi_{\mathcal{L}}(\vec{x})$ is the unique point in $\mathcal{L}$ closest to $\vec{x}$, and $\|\vec{x} - \Pi_{\mathcal{L}}(\vec{x})\|_2 = d(\vec{x}, \mathcal{L})$. The closed forms above hold.

Lemma 2 (Partial MDP for S1 When $\mathcal{N}$ Holds). If $\vec{x} \models \mathcal{N}$ and $\vec{x}' \models \mathcal{N}$, then

$$S_1(\vec{x}') \;<\; S_1(\vec{x}) \iff \|\vec{x}' - \Pi_{\mathcal{L}}(\vec{x}')\|_2^2 \;<\; \|\vec{x} - \Pi_{\mathcal{L}}(\vec{x})\|_2^2,$$

and therefore $d(\vec{x}', \mathcal{M}_C) \;<\; d(\vec{x}, \mathcal{M}_C)$, since under $\mathcal{N}$ the (local) model set coincides with $\mathcal{L}$. *Interpretation.* When the nonlinear/inequality part already holds, S1's decrease exactly tracks movement toward the model set. S1 provides a strong *starting point*; it is *not* used to declare SAT.

3.3 Stage 2 (S2): Squared-ULP Objective Is a Representing Function

We now align the objective with IEEE-754 semantics, again respecting CNF by multiplying inside each disjunction.

ULP Distance for Literals and Clauses (Multiplication in $\vee$). For a literal ℓ:

- equality $f(\vec{x}) == g(\vec{x})$: $d_{\text{ulp}}(\ell; \vec{x})$ is the number of adjacent FP steps (via IEEE `nextUp`/`nextDown`) between the IEEE values of $f(\vec{x})$ and $g(\vec{x})$ (0 iff bit-equal);
- inequality $f(\vec{x}) \le g(\vec{x})$: $d_{\text{ulp}}(\ell; \vec{x}) = 0$ if it holds; otherwise the *minimal* FP steps needed to make it hold (used purely as a distance).

For a *clause* ϕ, define

$$\text{Dist}_{\text{ulp}}(\phi; \vec{x}) \;:=\; \prod_{\ell \in \phi} d_{\text{ulp}}(\ell; \vec{x})^2,$$

so a satisfied literal forces the product to 0.

Representing Function and S2 Definition. A scalar R is a *representing function* for C if

$$R(\vec{x}) \;\ge\; 0 \quad \text{and} \quad R(\vec{x}) = 0 \iff \vec{x} \models C.$$

We set

$$S_2(\vec{x}) \;:=\; \sum_{\phi \in C} \text{Dist}_{\text{ulp}}(\phi; \vec{x}).$$

Lemma 3 (S2 Is Representing). S_2 is a representing function for C. *Reason.* Each clause distance is non-negative and is 0 iff the clause is satisfied; the sum is 0 iff all clauses are satisfied, i.e., $\vec{x} \models C$.

3.4 Stage 3 (S3): n-ULP Refinement Is a Representing Function (Discrete Steps)

Let $\vec{x}_2^\star$ be the *current best assignment returned by S2*. For an integer vector $\vec{n}$ (signed ULP steps applied component-wise via IEEE `nextUp`/`nextDown`), define the clause distance at the *stepped* point and sum over clauses:

$$\text{Dist}_{\text{ulp}}\big(\phi;\ \text{nULP}(\vec{n}, \vec{x}_2^\star)\big) \;:=\; \prod_{\ell \in \phi} d_{\text{ulp}}\big(\ell;\ \text{nULP}(\vec{n}, \vec{x}_2^\star)\big)^2,$$

$$S_3(\vec{n}) \;:=\; \sum_{\phi \in C} \text{Dist}_{\text{ulp}}\big(\phi;\ \text{nULP}(\vec{n}, \vec{x}_2^\star)\big).$$

Lemma 4 (S3 Is Representing at Stepped Points). For all $\vec{n}$, $S_3(\vec{n}) \ge 0$ and $S_3(\vec{n}) = 0 \iff \text{nULP}(\vec{n}, \vec{x}_2^\star) \models C$. *Interpretation.* S3 searches the FP lattice around the S2 result and can *snap* a near-miss to an exact model while preserving CNF semantics (product for $\vee$, sum for $\wedge$).

3.5 Overall Procedure and Guarantees

Inputs.

- A CNF constraint C over QF_FP atoms.
- A black-box MP_Inverse(M) that returns the Moore–Penrose pseudoinverse $M^\dagger$.
- A black-box global minimizer Global_Min$(F, \text{init}, \text{budget}) \to (\hat{\vec{x}}, F(\hat{\vec{x}}), \texttt{status})$.
- Stage iteration budgets $\text{budget}_1, \text{budget}_2, \text{budget}_3$.

Algorithm.

(1) **Build the objective functions from C.** Collect $\mathcal{L}$ into a system $A\vec{x} == \vec{b}$ and let the remaining atoms form $\mathcal{N}$.
   Define $S_1(\vec{x})$ as the sum of: (i) the squared distance from $\vec{x}$ to $\mathcal{L}$ via $\Pi_{\mathcal{L}}$ and (ii) the clause-level product of squared violation terms for $C_{\mathcal{N}}$ (cf. Section 3).
   Define $S_2(\vec{x})$ as the sum over clauses of the product of squared per-literal ULP distances (IEEE-aligned representing function; cf. Section 3).

Given an incumbent $\vec{x}_2^\star$ from S_2, define $S_3(\vec{n}; \vec{x}_2^\star)$ as the S_2 objective evaluated at the FP-lattice point $\mathrm{nULP}(\vec{n}, \vec{x}_2^\star)$ obtained by stepping each coordinate by n_j ULPs (cf. Section 3).

(2) **Minimize S1 (fast descent).** $(\vec{x}_1^\star, v_1, \_) \leftarrow \textsc{Global\_Min}\big(S_1,\ \textsc{MultiStartBox}(C),\ \mathrm{budget}_1\big)$. *Note:* we do *not* decide SAT here.

(3) **Minimize S2 (bit-precise optimization).** $(\vec{x}_2^\star, v_2, \_) \leftarrow \textsc{Global\_Min}\big(S_2,\ \vec{x}_1^\star,\ \mathrm{budget}_2\big)$. If $v_2 = 0$, return **sat** with $\vec{x}_2^\star$.

(4) **Minimize S3 (n-ULP refinement over the FP lattice).** Run a discrete FP-lattice search to minimize $S_3(\vec{n}; \vec{x}_2^\star)$. If the best value $v_3 = 0$, return **sat** with $\mathrm{nULP}(\vec{n}^\star, \vec{x}_2^\star)$.

(5) Otherwise, return **unsat-guess** with score $\min(v_2, v_3)$, or **timeout** if the time budget expires.

Lemma 3.1 (Outcome trichotomy). *For any CNF C over IEEE-754 atoms and any finite time budget, the procedure above returns exactly one of* {***sat***, ***unsat-guess***, ***timeout***}.

Proof. By case analysis on the control flow. Steps (3)-(4) return **sat** upon reaching an objective value 0 with validation; otherwise, upon termination with a strictly positive best value, step (5) returns **unsat-guess**; if the time budget elapses before either condition, step (4) returns **timeout**. These cases are mutually exclusive and exhaustive. □

Theorem 3.2 (SAT soundness). *If the procedure returns **sat** with assignment $\hat{\vec{x}}$, then $\hat{\vec{x}} \models C$.*

Proof. A **sat** result arises only in S2 or S3. In S2, $S_2(\hat{\vec{x}}) = 0$ and Lemma 3 (S2 is representing) imply $\hat{\vec{x}} \models C$. In S3, $S_3(\vec{n}^\star) = 0$ and Lemma 4 (S3 is representing at stepped points) imply $\mathrm{nULP}(\vec{n}^\star, \vec{x}_2^\star) \models C$, which is exactly the returned $\hat{\vec{x}}$. In both cases, we additionally validate under IEEE-754 before returning. Hence any **sat** output is a genuine model of C. □

Corollary 3.3 (No false SAT). *The procedure never reports **sat** on a non-model.*

Proof. Immediate from Theorem 3.2. □

Proposition 3.4 (Incompleteness). *There exist satisfiable C and finite budgets for which the procedure returns **unsat-guess** or **timeout**.*

Proof sketch. Numeric minimization in S1/S2 may converge to non-global minima; S3 searches only a finite set of integer step vectors. With finite time, neither guarantees reaching a zero even when one exists. □

4 Implementation Details

Projection construction. We assemble A and $\vec{b}$ from the top-level FP-linear equalities detected in C and compute the projection using $AA^\top$ and a rank-agnostic Moore–Penrose pseudoinverse. When $AA^\top$ is numerically singular or when no linear equalities exist, we gracefully skip the projection term and keep the squared penalties for the remaining atoms; the objective remains well-defined.

Projection computation phase and overhead. The projection is computed once during the constraint-parsing and objective-generation phase (Step (1) of the workflow in Section 3.5). We cache matrix factorizations for the projection term when A is unchanged across restarts.

Variable substitution versus orthogonal projection. Instead of projection, an alternative is to perform a change of variables, analytically eliminating linear constraints to search within a reduced space. While plausible in principle, this approach introduces numerical risks in floating-point arithmetic. Variable substitution can inadvertently amplify or reduce term magnitudes, trigger overflow or underflow in intermediate computations, and distort the objective landscape. We choose orthogonal projection because it preserves the original nonlinear constraints identically. Rather than rewriting

the nonlinear terms, projection measures the geometric distance to the linear manifold, ensuring numerically stable and reliable guidance during the initial S_1 search phase.

Clause aggregation. To evaluate CNF formulas efficiently while preserving our representing-function guarantees, we implement a "product-over-$\vee$ / sum-over-$\wedge$" scheme. By multiplying per-literal distances within a disjunction, a clause's penalty drops to exactly zero if and only if at least one of its literals is satisfied. Summing these non-negative clause distances ensures the global objective evaluates to zero if and only if all clauses hold. This specific algebraic translation is crucial for the implementation: it avoids complex branching during optimization while mathematically guaranteeing that reaching a zero objective strictly coincides with true logical satisfaction.

Global minimization. Global_Min is instantiated by a multi-start strategy with a derivative-free global minimizer; we terminate early on zero objectives and parallelize starts across cores. Stage budgets $\text{budget}_1, \text{budget}_2, \text{budget}_3$ are configurable, with S_1 typically receiving the largest share to shape a good basin, and S_3 using a small, bounded neighborhood.

ULP distances and mixed precision. We implement per-literal ULP distances in IEEE 754, handling strict inequalities with the standard ± 1 offset and dispatching to FP32/FP64 variants per variable. Mixed-precision atoms are supported by per-operand rules consistent with Section 3.

Fast descent. Squared distance facilitates faster descent compared to ULP distance. The performance gap lies primarily in *optimization landscape quality*. Squared residuals yield a continuous, real-valued objective with a smooth landscape, enabling S1 to reach the minimum with fewer objective evaluations, especially on large benchmarks. In contrast, the ULP-based objective takes only discrete integer values, forming a piecewise-constant, staircase-like surface.

S3 implementation. S3 employs a customized basin-hopping minimizer to explore a discrete FP-lattice neighborhood around each variable, bounded by a maximum of 100 ULPs in each coordinate. The neighborhood bound is configurable via input options.

Constant-time nULP stepping. The primitive $\text{nULP}(\vec{n}, \vec{x})$ is $O(1)$ per coordinate: we reinterpret each FP value as a monotone signed integer index over the FP lattice, add n_j as a *single* integer operation, and bit-cast back to FP, clamping to the nearest finite value when necessary. Subnormals and signed zeros are preserved by construction.

Numerical hygiene. We exclude NaN/$\pm\infty$ from the search space, precompute per-variable sorts (FP32/FP64) to dispatch kernels efficiently.

5 Experiments

5.1 Executive Summary

StageSAT demonstrates robust performance and reliability on widely-used floating-point satisfiability benchmarks:

- *Broad coverage*: StageSAT produced results on 45 of 49 formulas in the hardest FP benchmark suite (versus 39 by the best solver), and handled more benchmarks overall than any competing solver thanks to a much lower timeout rate.
- *Near-perfect accuracy in SAT detection*: StageSAT found solutions for all but two satisfiable formulas in our benchmark suite (99.4% recall) and never reported a satisfiable result for any unsatisfiable formula (0% false SAT). These outcomes perfectly aligned with the complete solver's UNSAT verdicts, with no incorrect models produced in any case.
- *Significant time improvement*: StageSAT delivered these results much faster than the complete solver, achieving an overall solving speedup of 5–10× faster on average. It also encountered far fewer timeouts, solving nearly all instances within the time limit in our benchmarks.

- *Essential pipeline*: Removing any one of StageSAT's three optimization stages significantly reduced the number of instances it could handle (or led to missed solutions), confirming that every component of its design is critical for effectiveness.

5.2 Experimental Setup

We evaluate *StageSAT* on five suites spanning a broad range of floating-point SMT problems: *MathSAT-Small* (130 files, ≤10 KB), *MathSAT-Middle* (35 files, 11–20 KB), *MathSAT-Large* (49 files, >20 KB), *Grater* (118 files; 5 non-RNE cases filtered), and *JFS* (111 files). The Large set serves as the primary stress test; the others provide breadth across sizes and sources. We compare two solver categories: *incomplete* solvers—*XSat*, *goSAT*, *Grater*, and *JFS* (optimization/fuzzing based; cannot prove UNSAT)—and *complete* solvers—*Z3*, *cvc5*, *MathSAT*, and *Bitwuzla* (bit-precise decision procedures). All experiments ran on an *Intel Core i9-13900HK* (14 cores), *32 GB RAM, Ubuntu 24.04.3 LTS*, with a *20-minute* wall-clock cap per instance (except for a stress test which we use 48 hours and results are reported in Table 3). Incomplete solvers (including *StageSAT*) were run 5× per benchmark.

Our evaluation uses the following metrics. *SAT coverage (a.k.a. SAT recall)* is the fraction of ground-truth SAT instances for which the solver returns `sat` with a validated model. We obtain all ground-truth SAT/UNSAT labels from complete solvers: whenever at least one complete solver finishes within the 48-hour limit, we adopt its result (and in all such cases, completed solvers agree). For one particularly hard benchmark, we reran the fastest complete solver, Bitwuzla, with a longer timeout and used its final result as ground truth.

$$\text{SAT-Recall} \;=\; \frac{\#\{\text{reported SAT} \;\wedge\; \text{ground-truth=SAT}\}}{\#\{\text{ground-truth=SAT}\}}\,.$$

Timeout rate is the share of all benchmarks with no verdict within 20 minutes:

$$\text{Timeout-rate} = \frac{\#\{\text{timeout}\}}{\#\{\text{all benchmarks}\}}.$$

Average (mean) time is the arithmetic mean per suite with timeouts counted at the cap (to reflect robustness under the budget). *Median time* is reported alongside the mean to reflect typical behavior and reduce heavy-tail effects.

We do not use **SAT precision** (the fraction of reported `sat` verdicts that are truly SAT) for comparison. In our experiments, it is effectively 100% for all solvers—rare exceptions occurred only when *Grater* accepted sub-tolerance objectives—making it non-discriminative. Moreover, precision can be inflated by reporting `sat` only on easy cases, while SAT coverage can be inflated by labeling everything `sat`. We therefore emphasize **SAT coverage, timeout rate, and time statistics**. For *StageSAT*, `unsat-guess` is a distinct, non-proof outcome that we never count as proved UNSAT or as "solved".

5.3 Quantitative Results

We organize the experimental findings by four research questions (RQ1-4).

RQ1: StageSAT vs. Incomplete Solvers (MathSAT-Large). *How does StageSAT perform relative to prior optimization-based and heuristic (incomplete) solvers on large FP benchmarks?* We compare StageSAT with XSat, goSAT, Grater, and JFS on the 49 MathSAT-Large formulas. The results show that StageSAT attains the highest coverage and accuracy among these incomplete solvers. Table 1 reports detailed outcomes per benchmark. StageSAT was able to return a result (SAT or unsat-guess) for 45 out of 49 instances, whereas the next-best incomplete solver (XSat) solved significantly fewer within the same time limit. In terms of SAT coverage on the MathSAT-Large suite, StageSAT found satisfying assignments for 24 of the 26 satisfiable benchmarks, outperforming all prior numeric competitors in the number of SAT instances solved. In fact, XSat comes closest in number of instances solved, but there is a crucial difference – soundness. XSat often misclassified difficult

Table 1. Comparison on MathSAT-Large benchmarks: StageSAT vs. incomplete solvers. Timeout as 20 mins.

Benchmark	Size(byte)	#Vars	XSat		goSAT		Grater		JFS		StageSAT	
			Verdict	Time(s)	Verdict	Time(s)	Verdict	Time(s)	Verdict	Time(s)	Verdict	Time(s)
sqrt.c.10	21701	26	unsat	7.81	unknown	0.72	timeout	>1200	timeout	>1200	sat	102.48
test_v5_r15_vr5_c1_s8246	21791	5	unsat	1.62	unknown	0.02	timeout	>1200	timeout	>1200	unsat-guess	10.76
test_v5_r15_vr1_c1_s26845	21811	5	unsat	1.24	unknown	0.02	timeout	>1200	timeout	>1200	unsat-guess	7.00
test_v5_r15_vr10_c1_s25268	21818	5	unsat	1.54	unknown	0.02	timeout	>1200	timeout	>1200	unsat-guess	8.87
test_v5_r15_vr5_c1_s26657	22070	5	unsat	1.48	unknown	0.02	timeout	>1200	timeout	>1200	unsat-guess	6.70
test_v5_r15_vr1_c1_s32559	22072	5	unsat	1.58	unknown	0.02	timeout	>1200	timeout	>1200	unsat-guess	5.78
test_v5_r15_vr1_c1_s8236	22072	5	unsat	1.22	unknown	0.02	timeout	>1200	timeout	>1200	unsat-guess	5.88
test_v5_r15_vr5_c1_s23844	22072	5	unsat	1.66	unknown	0.02	timeout	>1200	timeout	>1200	unsat-guess	9.21
test_v5_r15_vr10_c1_s14516	22252	5	unsat	1.40	unknown	0.02	timeout	>1200	timeout	>1200	unsat-guess	10.56
qurt.c.5	23169	30	unsat	9.44	unknown	0.83	sat	1177.90	timeout	>1200	unsat-guess	156.14
test_v7_r12_vr5_c1_s29826	23736	7	sat	0.18	sat	0.02	sat	0.07	sat	3.25	sat	0.06
test_v7_r12_vr10_c1_s15994	23828	7	sat	0.14	sat	0.02	sat	0.21	sat	16.94	sat	0.06
test_v7_r12_vr10_c1_s30410	24070	7	sat	1.40	sat	0.02	sat	0.89	timeout	>1200	sat	0.09
test_v7_r12_vr5_c1_s14336	24250	7	sat	0.14	sat	0.03	sat	0.07	sat	0.38	sat	0.05
test_v7_r12_vr5_c1_s8938	24251	7	sat	0.14	sat	0.02	sat	0.07	sat	0.14	sat	0.05
test_v7_r12_vr1_c1_s10576	24274	7	unsat	3.04	unknown	0.03	timeout	>1200	timeout	>1200	unsat-guess	14.56
test_v7_r12_vr1_c1_s22787	24345	7	unsat	2.24	unknown	0.04	timeout	>1200	timeout	>1200	unsat-guess	11.65
test_v7_r12_vr10_c1_s18160	24437	7	unsat	2.40	unknown	0.04	timeout	>1200	timeout	>1200	unsat-guess	21.18
test_v7_r12_vr1_c1_s703	24441	7	unsat	2.76	unknown	0.03	timeout	>1200	timeout	>1200	unsat-guess	16.54
sin2.c.15	25235	52	sat	25.81	unknown	1.78	sat	13.88	timeout	>1200	sat	3.92
gaussian.c.25	29883	79	sat	0.72	unknown	2.52	sat	9.12	timeout	>1200	sat	3.38
sqrt.c.15	32192	36	unsat	13.10	unknown	1.06	timeout	>1200	timeout	>1200	sat	101.76
test_v7_r17_vr5_c1_s2807	32711	7	unsat	2.20	unknown	0.04	timeout	>1200	timeout	>1200	unsat-guess	15.21
test_v7_r17_vr1_c1_s30331	32876	7	unsat	2.58	unknown	0.05	timeout	>1200	timeout	>1200	unsat-guess	14.96
test_v7_r17_vr5_c1_s25451	32964	7	unsat	2.30	unknown	0.04	timeout	>1200	timeout	>1200	unsat-guess	11.58
sin2.c.20	33016	67	unsat	41.81	unknown	1.74	sat	33.89	timeout	>1200	sat	169.92
test_v7_r17_vr10_c1_s8773	33151	7	sat	0.74	sat	0.03	sat	0.35	timeout	>1200	sat	0.06
test_v7_r17_vr5_c1_s4772	33222	7	unsat	2.30	unknown	0.05	timeout	>1200	timeout	>1200	unsat-guess	35.20
test_v7_r17_vr1_c1_s23882	33226	7	sat	0.14	sat	0.03	sat	0.29	timeout	>1200	sat	0.05
test_v7_r17_vr1_c1_s24331	33226	7	unsat	3.12	unknown	0.04	timeout	>1200	timeout	>1200	unsat-guess	11.63
test_v7_r17_vr10_c1_s3680	33335	7	unsat	3.06	unknown	0.05	timeout	>1200	timeout	>1200	unsat-guess	13.26
test_v7_r17_vr10_c1_s18654	33410	7	sat	0.16	sat	0.03	sat	5.73	timeout	>1200	sat	0.05
sin.c.25	40536	81	unsat	76.20	unknown	2.31	sat	98.30	timeout	>1200	sat	294.71
sin2.c.25	40747	82	unsat	61.49	unknown	2.84	sat	93.33	timeout	>1200	sat	8.62
sqrt.c.20	46804	63	unsat	45.91	unknown	1.73	timeout	>1200	timeout	>1200	sat	498.60
sqrt.c.25	46804	63	unsat	46.33	unknown	1.82	timeout	>1200	timeout	>1200	sat	581.95
qurt.c.10	47946	60	unsat	22.89	unknown	1.28	timeout	>1200	timeout	>1200	unsat-guess	393.02
qurt.c.15	73125	90	unsat	53.40	unknown	2.95	timeout	>1200	timeout	>1200	unsat-guess	728.38
gaussian.c.75	89686	229	sat	20.05	unknown	13.31	sat	542.51	timeout	>1200	sat	36.45
qurt.c.20	93126	114	unsat	68.26	unknown	3.96	timeout	>1200	timeout	>1200	timeout	>1200
qurt.c.25	93126	114	unsat	76.92	unknown	4.31	timeout	>1200	timeout	>1200	timeout	>1200
sin2.c.75	119790	231	unsat	242.70	unknown	11.70	timeout	>1200	timeout	>1200	sat	363.37
sin.c.75	119794	231	unsat	214.49	unknown	12.32	timeout	>1200	timeout	>1200	sat	660.70
gaussian.c.125	150792	379	sat	13.20	unknown	40.24	sat	89.02	timeout	>1200	sat	90.62
sin.c.125	200503	381	unsat	1000.64	unknown	34.38	error	–	timeout	>1200	sat	379.39
sin2.c.125	200503	381	unsat	984.14	unknown	33.26	error	–	timeout	>1200	sat	313.85
gaussian.c.175	210711	529	unsat	1101.04	unknown	80.37	timeout	>1200	timeout	>1200	sat	273.68
sin2.c.175	280962	531	timeout	>1200	unknown	76.20	error	–	timeout	>1200	timeout	>1200
sin.c.175	280984	531	timeout	>1200	unknown	72.63	error	–	timeout	>1200	timeout	>1200
SAT Coverage			**46.2%**		**30.8%**		**57.7%**		**15.4%**		**92.3%**	
Timeout Rate			**4.1%**		**0.0%**		**51.0%**		**91.8%**		**8.2%**	
Average Time(s)			**134.02**		**8.27**		**819.24**		**1102.46**		**208.00**	
Median Time(s)			**3.04**		**0.05**		**>1200**		**>1200**		**14.96**	

satisfiable problems as "UNSAT" when it failed to locate a solution, whereas StageSAT never falsely reported UNSAT in our tests. Specifically, XSat produced 12 incorrect UNSAT answers on benchmarks that actually have solutions (StageSAT managed to find valid models for all 12 of these). All StageSAT models were independently validated with a bit-precise checker (Z3), confirming their correctness [10]. This highlights that StageSAT's optimization strategy improves not only the quantity of problems solved but also the precision of the results on challenging formulas.

Other baselines fared less well on MathSAT-Large. Grater, the state-of-the-art optimization-based solver, timed out on most large benchmarks that StageSAT solved, yielding a lower overall solve count. Moreover, Grater's use of a fixed tolerance can lead to spurious SAT reports – in one case it reported "SAT" even though the formula is actually unsatisfiable (both Z3 and cvc5 proved it UNSAT). StageSAT avoids this pitfall by requiring a strict zero objective for SAT, and accordingly never reports SAT for an unsatisfiable instance. The heuristic fuzzer JFS and the global optimizer goSAT solved the fewest problems in this suite, struggling with the complex constraints (many of their runs resulted in timeouts or unknown results). Overall, StageSAT offers the best balance of coverage and reliability: it solves significantly more large benchmarks than JFS or goSAT, more than Grater, and matches XSat's coverage *without any* of XSat's misclassification errors [12]. In practical terms, StageSAT was able to solve the vast majority of these large-instance challenges (often on every trial run), whereas the other incomplete solvers were either inconsistent or outright failed on many instances. These results confirm that StageSAT's novel staged optimization approach yields a substantial improvement in both solver effectiveness (higher solve rate) and solver soundness (no incorrect answers) for difficult floating-point problems.

RQ2: StageSAT vs. Complete Solvers (MathSAT-Large). *How does StageSAT compare to modern complete SMT solvers on the same large benchmarks?* In this analysis we evaluate StageSAT against Bitwuzla, cvc5, Z3, and MathSAT5 on the 49 MathSAT-Large formulas, as shown in Table 2. A key question is whether StageSAT's heuristic approach can achieve similar coverage to these complete (bit-precise) solvers, which can in principle solve or refute any instance given unlimited time. We found that StageSAT's coverage on large benchmarks is competitive with the best of the complete solvers. Within the 20-minute limit, StageSAT produced results for 45 out of 49 instances, slightly more than even the top-performing complete solver (Bitwuzla solved 39 in time). cvc5 solved 36, while Z3 and MathSAT5 handled fewer cases within the timeout. In other words, StageSAT's incomplete strategy allowed it to tackle about 92% of these hard instances, essentially matching state-of-the-art coverage on this large-scale set. This is notable because complete solvers sometimes exhaust the time on very complex formulas, whereas StageSAT either finds a model or converges on an unsatisfiability guess more quickly in most cases.

In terms of performance, StageSAT also shows advantages on satisfiable instances: it typically finds solutions significantly faster than complete solvers. Many satisfiable MathSAT-Large benchmarks that Bitwuzla or cvc5 only solve near the timeout are solved by StageSAT in a fraction of the time (often 5–10× faster, and on the largest satisfiable cases over 10× faster in our tests). We also ran stress tests with StageSAT and Bitwuzla under an extended 48-hour timeout to evaluate their behavior on the largest benchmarks where Bitwuzla times out, with results summarized in Table 3. For example, on one particularly difficult satisfiable formula (sin.c.125), StageSAT succeeded quickly while Bitwuzla *failed to find any model even with 48 hours of searching*. This underscores StageSAT's strength in navigating the search space for models efficiently. On the other hand, complete solvers have an edge in proving unsatisfiability. By design, StageSAT cannot provide formal UNSAT proofs – it will output unsat-guess when its optimization process concludes that no better (lower) objective can be found, but this is a heuristic indication. We examined all cases: every benchmark that was proven UNSAT by any complete solver was classified as unsat-guess by StageSAT. Crucially, we observed no discrepancies – StageSAT did not label any instance as satisfiable that a complete solver proved unsatisfiable, and whenever StageSAT gave an unsat-guess, the instance indeed had no solution according to the complete solvers. This empirical agreement suggests that StageSAT's UNSAT guesses were reliable on MathSAT-Large: they acted as a correct heuristic proxy for true unsatisfiability in all tested cases. However, since these are not formal proofs, we do not count them as proven UNSAT; instead, they demonstrate that StageSAT can recognize unsolvable instances with reasonable confidence. On the four hardest cases where StageSAT timed out (no result), the

Table 2. Comparison on MathSAT-Large benchmarks: StageSAT vs. complete solvers. Timeout as 20 mins.

Benchmark	Size(byte)	#Vars	cvc5		Bitwuzla		Z3		MathSAT		StageSAT	
			Verdict	Time(s)	Verdict	Time(s)	Verdict	Time(s)	Verdict	Time(s)	Verdict	Time(s)
sqrt.c.10	21701	26	sat	1.01	sat	6.15	sat	492.62	sat	28.08	sat	102.48
test_v5_r15_vr5_c1_s8246	21791	5	unsat	269.56	unsat	95.73	timeout	>1200	unsat	294.65	unsat-guess	10.76
test_v5_r15_vr1_c1_s26845	21811	5	unsat	75.80	unsat	47.68	unsat	737.95	unsat	131.37	unsat-guess	7.00
test_v5_r15_vr10_c1_s25268	21818	5	unsat	216.89	unsat	156.56	timeout	>1200	timeout	>1200	unsat-guess	8.87
test_v5_r15_vr5_c1_s26657	22070	5	unsat	160.27	unsat	63.23	timeout	>1200	unsat	177.60	unsat-guess	6.70
test_v5_r15_vr1_c1_s32559	22072	5	unsat	43.48	unsat	33.09	unsat	139.16	unsat	24.20	unsat-guess	5.78
test_v5_r15_vr1_c1_s8236	22072	5	unsat	49.45	unsat	25.43	unsat	369.09	unsat	12.66	unsat-guess	5.88
test_v5_r15_vr5_c1_s23844	22072	5	unsat	266.86	unsat	95.28	timeout	>1200	unsat	231.11	unsat-guess	9.21
test_v5_r15_vr10_c1_s14516	22252	5	unsat	530.11	unsat	202.41	timeout	>1200	timeout	>1200	unsat-guess	10.56
qurt.c.5	23169	30	unsat	6.59	unsat	0.22	unsat	13.44	unsat	6.53	unsat-guess	156.14
test_v7_r12_vr5_c1_s29826	23736	7	sat	129.16	sat	163.13	sat	425.77	sat	65.19	sat	0.06
test_v7_r12_vr10_c1_s15994	23828	7	sat	176.39	sat	98.18	sat	509.78	sat	130.89	sat	0.06
test_v7_r12_vr10_c1_s30410	24070	7	timeout	>1200	timeout	>1200	timeout	>1200	timeout	>1200	sat	0.09
test_v7_r12_vr5_c1_s14336	24250	7	sat	22.82	sat	115.91	sat	284.32	sat	84.11	sat	0.05
test_v7_r12_vr5_c1_s8938	24251	7	sat	19.00	sat	37.72	sat	105.88	sat	30.66	sat	0.05
test_v7_r12_vr1_c1_s10576	24274	7	unsat	322.33	unsat	170.45	timeout	>1200	unsat	206.12	unsat-guess	14.56
test_v7_r12_vr1_c1_s22787	24345	7	unsat	597.04	unsat	217.96	timeout	>1200	unsat	581.35	unsat-guess	11.65
test_v7_r12_vr10_c1_s18160	24437	7	timeout	>1200	timeout	>1200	timeout	>1200	timeout	>1200	unsat-guess	21.18
test_v7_r12_vr1_c1_s703	24441	7	unsat	639.55	unsat	234.56	timeout	>1200	timeout	>1200	unsat-guess	16.54
sin2.c.15	25235	52	sat	29.95	sat	159.21	timeout	>1200	timeout	>1200	sat	3.92
gaussian.c.25	29883	79	sat	3.54	sat	2.47	sat	642.11	sat	11.65	sat	3.38
sqrt.c.15	32192	36	sat	8.95	sat	1.29	timeout	>1200	sat	27.84	sat	101.76
test_v7_r17_vr5_c1_s2807	32711	7	timeout	>1200	timeout	>1200	timeout	>1200	timeout	>1200	unsat-guess	15.21
test_v7_r17_vr1_c1_s30331	32876	7	unsat	205.82	unsat	227.17	timeout	>1200	unsat	1140.11	unsat-guess	14.96
test_v7_r17_vr5_c1_s25451	32964	7	timeout	>1200	unsat	888.67	timeout	>1200	timeout	>1200	unsat-guess	11.58
sin2.c.20	33016	67	sat	560.39	sat	693.96	timeout	>1200	timeout	>1200	sat	169.92
test_v7_r17_vr10_c1_s8773	33151	7	sat	794.03	sat	530.94	timeout	>1200	timeout	>1200	sat	0.06
test_v7_r17_vr5_c1_s4772	33222	7	timeout	>1200	timeout	>1200	timeout	>1200	timeout	>1200	unsat-guess	35.20
test_v7_r17_vr1_c1_s23882	33226	7	sat	273.29	sat	340.07	sat	1162.59	sat	524.15	sat	0.05
test_v7_r17_vr1_c1_s24331	33226	7	timeout	>1200	unsat	787.68	timeout	>1200	timeout	>1200	unsat-guess	11.63
test_v7_r17_vr10_c1_s3680	33335	7	timeout	>1200	unsat	1188.71	timeout	>1200	timeout	>1200	unsat-guess	13.26
test_v7_r17_vr10_c1_s18654	33410	7	sat	1126.63	sat	588.61	timeout	>1200	timeout	>1200	sat	0.05
sin.c.25	40536	81	sat	802.21	sat	874.55	timeout	>1200	timeout	>1200	sat	294.71
sin2.c.25	40747	82	sat	713.26	sat	902.61	timeout	>1200	timeout	>1200	sat	8.62
sqrt.c.20	46804	63	sat	0.96	sat	5.96	sat	721.16	sat	18.39	sat	498.60
sqrt.c.25	46804	63	sat	0.96	sat	5.95	sat	717.69	sat	18.42	sat	581.95
qurt.c.10	47946	60	unsat	6.59	unsat	0.21	unsat	23.86	unsat	6.62	unsat-guess	393.02
qurt.c.15	73125	90	unsat	0.27	unsat	0.02	unsat	1.93	unsat	6.61	unsat-guess	728.38
gaussian.c.75	89686	229	sat	13.46	sat	6.57	sat	429.71	sat	56.89	sat	36.45
qurt.c.20	93126	114	unsat	0.30	unsat	0.01	timeout	>1200	unsat	7.25	timeout	>1200
qurt.c.25	93126	114	unsat	0.32	unsat	0.01	timeout	>1200	unsat	7.21	timeout	>1200
sin2.c.75	119790	231	timeout	>1200	timeout	>1200	error	–	timeout	>1200	sat	363.37
sin.c.75	119794	231	timeout	>1200	timeout	>1200	error	–	timeout	>1200	sat	660.70
gaussian.c.125	150792	379	sat	20.54	sat	36.94	timeout	>1200	sat	338.55	sat	90.62
sin.c.125	200503	381	timeout	>1200	timeout	>1200	error	–	timeout	>1200	sat	379.39
sin2.c.125	200503	381	timeout	>1200	timeout	>1200	error	–	timeout	>1200	sat	313.85
gaussian.c.175	210711	529	sat	460.36	sat	745.12	timeout	>1200	timeout	>1200	sat	273.68
sin2.c.175	280962	531	timeout	>1200	timeout	>1200	error	–	timeout	>1200	timeout	>1200
sin.c.175	280984	531	timeout	>1200	timeout	>1200	error	–	timeout	>1200	timeout	>1200
SAT/UNSAT Coverage			**73.5%**		**79.6%**		**32.7%**		**53.1%**		**91.8%**	
Timeout Rate			**26.5%**		**20.4%**		**55.1%**		**46.9%**		**8.2%**	
Average Time(s)			**492.82**		**443.89**		**911.09**		**648.33**		**208.00**	
Median Time(s)			**269.56**		**170.45**		**>1200**		**581.35**		**14.96**	

complete solvers eventually determined two to be satisfiable and two unsatisfiable after extended runs. This highlights a limitation: for the very largest or trickiest satisfiable formulas, StageSAT may also struggle (as it did on two extremely large satisfiable benchmarks that required more than 20 minutes). Conversely, for some tough UNSAT cases, complete solvers' heavy-duty reasoning succeeded where StageSAT's heuristic approach did not finish in time.

Table 3. Comparison on MathSAT-Large benchmarks where Bitwuzla timeout: StageSAT vs. Bitwuzla. Timeout as 48 hours.

Benchmark	Size(byte)	#Vars	StageSAT		Bitwuzla	
			Verdict	Time(s)	Verdict	Time(s)
test_v7_r12_vr10_c1_s30410	24070	7	sat	0.09	sat	3042.79
test_v7_r12_vr10_c1_s18160	24437	7	unsat-guess	21.18	unsat	7347.42
test_v7_r17_vr5_c1_s2807	32711	7	unsat-guess	15.21	unsat	1466.26
test_v7_r17_vr5_c1_s4772	33222	7	unsat-guess	35.20	unsat	93950.26
sin2.c.75	119790	231	sat	363.37	sat	4180.57
sin.c.75	119794	231	sat	660.70	sat	9361.38
sin.c.125	200503	381	sat	379.39	timeout	>48 hours
sin2.c.125	200503	381	sat	313.85	sat	105514.52
sin2.c.175	280962	531	timeout	>48 hours	sat	151645.17
sin.c.175	280984	531	timeout	>48 hours	sat	68898.42

Despite these few limitations, StageSAT performs well against the complete solvers. It matched or exceeded their solve counts within the standard time limit and delivered solutions faster on the instances it could solve. This indicates StageSAT can serve as a practical complement to traditional SMT solvers. In a usage scenario, one might run StageSAT alongside a complete solver: StageSAT will quickly find a model if one exists for hard satisfiable problems (saving potentially hours of search), while for the few instances that require a proof of unsatisfiability, a complete solver can take over. StageSAT's ability to handle large benchmarks competitively demonstrates the benefit of its approach – bridging numeric optimization with SMT – in achieving both scalability and empirical correctness on floating-point constraints.

RQ3: Ablation Study of StageSAT's Design. *Are all three stages and key design components of StageSAT necessary for its performance?* StageSAT's solving process comprises three sequential stages (S1, S2, S3) with different objective formulations, plus additional techniques like the projection term in S1 and a clause-wise ULP aggregation in later stages. To understand the contribution of each part, we performed an ablation study: running modified versions of StageSAT with one component removed or altered, and measuring the impact on MathSAT-Large results. Table 4 presents the outcome counts for each variant. The full StageSAT (S1–S2–S3 as designed) solved 45 of 49 large benchmarks (24 reported SAT, 21 unsat-guess) with an average solve time of 208 s per instance. We use this as the baseline for comparison.

Table 4. Ablation summary on MathSAT-Large benchmarks. Timeout as 20 mins.

Variant	SAT (#)	UNSAT (#)	Timeout (#)	Avg. time (s)
Full S1–S2–S3	24	21	4	208.00
No S1 (start S2 directly)	13	22	14	409.07
No S3 (S1–S2 only)	22	23	4	262.61
S1 without projection term	21	24	4	291.82
S1 with absolute residuals	18	27	4	196.05
S2/S3 without clause-wise ULP product	22	21	6	264.06

Removing Stage 1 and starting directly with Stage 2 had a drastic effect on performance. This "No S1" variant solved only 35/49 instances (a drop from 45) and suffered 14 timeouts (versus 4 in full StageSAT). In particular, the number of satisfiable instances found fell sharply (from 24 down to

just 13). The average runtime on solved cases also nearly doubled (409 s vs 208 s). This shows that Stage 1 is crucial for scalability – its projection-aided, squared-residual objective quickly guides the solver toward feasible regions. Without S1's fast coarse guidance, the solver struggled and timed out much more frequently. Intuitively, S1's objective (based on magnitudes of residuals) requires fewer objective evaluations to converge compared to the more precise ULP-based objectives in S2 and S3, so it can explore the search space broadly and find a promising region quickly. Conclusion: Stage 1 is indispensable for reaching high coverage on large problems.

Conceptually, Stage 2 is critical in regimes where purely squared-magnitude objectives get stuck, such as subnormal regions and underflow. The MathSAT-Large set doesn't contain such stress cases, which is why we do not include a "No S2" configuration in this ablation study. In contrast, the JFS and Grater benchmarks include many formulas with similar structure, and on those suites Stage 2 is essential for steering StageSAT toward correct models.

Next, removing Stage 3 had a more subtle but important impact. The "No S3" variant solved the same total number of instances (45/49), *but* it found fewer SAT solutions (22 vs 24) and correspondingly classified more cases as unsat-guess (23 vs 21). In fact, two benchmarks that full StageSAT successfully solved as SAT were missed by the S1–S2-only solver, which converged to a non-zero minimum and reported them (incorrectly) as unsatisfiable. This indicates that Stage 3 is essential for correctness on edge cases – it performs a discrete, high-precision search over the FP lattice that can "close the gap" when the continuous optimization in S2 stalls. Without S3, StageSAT's SAT coverage would drop and it would misclassify some satisfiable problems as unsat-guess. Conclusion: Stage 3 is necessary to achieve StageSAT's 100% soundness for SAT results, by catching those tricky cases where numerical smoothing alone isn't enough.

We also tested variations in the *within-stage techniques*. Removing the orthogonal projection term from Stage 1 (while still doing S1–S2–S3) kept the total solved count at 45, but StageSAT's SAT find rate worsened slightly (21 SAT vs 24) and the average solve time increased to ~292 s. This suggests the projection term – which imposes a partial monotone descent property – indeed helps the solver find more models (3 extra in the full version) and do so faster. Its absence likely caused the optimizer to sometimes get bogged down on flat or misleading surfaces, turning a few would-be SAT cases into unsat-guesses and generally slowing progress. Replacing S1's squared-residual objective with a simpler absolute residual objective had a mixed effect: it slightly improved runtime (196 s average) but at a steep cost in SAT solutions (only 18 SAT found, with 27 unsat-guess). This trade-off indicates that while absolute residuals might speed up convergence in some cases, they provide a weaker guidance toward exact solutions, causing StageSAT to miss many satisfiable cases that the squared residual version would solve. Lastly, disabling the clause-wise ULP product in Stages 2–3 (an aggregation scheme that emphasizes satisfying all constraints together) reduced StageSAT's overall solves to 43/49 and led to 6 timeouts (versus 4 in full StageSAT). It also slightly diminished SAT coverage (22 SAT vs 24 in full StageSAT). This indicates that the clause-wise product, though a simple heuristic to combine per-clause errors, helps StageSAT prioritize assignments that satisfy *every* constraint, thereby avoiding some timeouts and finding a couple more models.

In summary, the ablation study confirms that each stage and each design choice contributes meaningfully to StageSAT's success. Stage 1's projection-aided coarse search is vital for scale and speed; Stage 2's ULP-based continuous optimization is critical for navigating tricky FP regions; Stage 3's discrete refinement is crucial for final correctness; and the added touches (projection term, squared metrics, ULP aggregation) further improve both performance and result quality. Removing any one of these either hurts StageSAT's ability to solve tough instances or causes it to miss solutions, validating the full S1–S2–S3 design as necessary to achieve the reported high coverage and accuracy.

Table 5. Summary on MathSAT-Large, MathSAT-Middle, MathSAT-Small, JFS, and Grater benchmarks. #Unk/UG = #Unknown/Unsat-Guess. Timeout as 20 mins.

Dataset	Solver	#SAT	#UNSAT	#Timeout	#Unk/UG	#Error	SAT Coverage	Avg. time(s)	Med. time(s)
MathSAT-Large (49 benchmarks, 26 SAT, 23 UNSAT)									
	CVC5	19	17	13	0	0	73.1%	492.82	269.56
	Bitwuzla	19	20	10	0	0	73.1%	443.89	170.45
	Z3	10	6	27	0	6	38.5%	911.09	>1200
	MathSAT	12	14	23	0	0	46.2%	648.33	581.35
	XSat	12	23	2	0	0	46.2%	134.02	3.04
	goSAT	8	0	0	41	0	30.8%	8.27	0.05
	Grater	16	0	25	0	4	57.7%	819.24	>1200
	JFS	4	0	45	0	0	15.4%	1102.46	>1200
	StageSAT	24	0	4	21	0	92.3%	208.00	14.96
MathSAT-Middle (35 benchmarks, 29 SAT, 6 UNSAT)									
	CVC5	29	6	0	0	0	100.0%	94.65	39.24
	Bitwuzla	29	6	0	0	0	100.0%	31.10	22.59
	Z3	19	5	6	0	5	65.5%	370.31	323.14
	MathSAT	28	6	1	0	0	96.6%	77.19	33.47
	XSat	29	6	0	0	0	100.0%	1.06	0.14
	goSAT	23	0	0	12	0	79.3%	0.26	0.02
	Grater	26	0	9	0	0	89.7%	337.13	0.18
	JFS	9	0	26	0	0	31.0%	892.06	>1200
	StageSAT	29	0	0	6	0	100.0%	8.59	0.18
MathSAT-Small (130 benchmarks, 63 SAT, 67 UNSAT)									
	CVC5	63	55	12	0	0	100.0%	136.44	3.34
	Bitwuzla	63	64	3	0	0	100.0%	35.64	1.93
	Z3	56	53	13	0	8	88.9%	194.18	24.79
	MathSAT	63	51	16	0	0	100.0%	168.37	7.31
	XSat	62	67	0	0	1	98.4%	0.46	0.14
	goSAT	58	0	0	72	0	92.1%	0.07	0.02
	Grater	63	0	38	0	28	100.0%	471.50	0.81
	JFS	53	0	77	0	0	84.1%	717.96	>1200
	StageSAT	63	0	0	67	0	100.0%	1.64	0.66
JFS (111 benchmarks, 111 SAT, 0 UNSAT)									
	CVC5	104	0	6	0	1	93.7%	110.87	4.74
	Bitwuzla	108	0	3	0	0	97.3%	103.65	3.24
	Z3	85	0	18	0	8	76.6%	327.55	4.27
	MathSAT	102	0	9	0	0	91.9%	149.23	11.60
	XSat	101	4	0	0	6	91.0%	13.04	0.14
	goSAT	88	0	0	19	4	79.3%	2.21	0.03
	Grater	107	0	4	0	0	96.4%	71.86	0.06
	JFS	69	0	42	0	0	62.2%	458.33	0.72
	StageSAT	111	0	0	0	0	100.0%	13.10	0.05
Grater (118 benchmarks, 118 SAT, 0 UNSAT)									
	CVC5	118	0	0	0	0	100.0%	37.62	2.13
	Bitwuzla	117	0	1	0	0	99.2%	64.30	10.61
	Z3	103	0	12	0	3	87.3%	241.12	64.73
	MathSAT	111	0	7	0	0	94.1%	131.05	25.29
	XSat	115	2	0	0	1	97.5%	1.62	0.14
	goSAT	92	0	0	26	0	78.0%	0.70	0.02
	Grater	118	0	0	0	0	100.0%	4.88	0.41
	JFS	40	0	78	0	0	33.9%	793.65	>1200
	StageSAT	118	0	0	0	0	100.0%	10.34	0.18

RQ4: Overall Performance Across All Benchmark Suites. *What is StageSAT's overall performance when considering all benchmark categories (small, medium, large, and different sources) and how does it compare to other solvers across the board?* To answer this, we aggregated results from all five benchmark sets in Table 5: MathSAT-Large, MathSAT-Middle, MathSAT-Small, JFS, and Grater. StageSAT's trends observed in the large suite extend to the others, showing a consistently strong performance. First, StageSAT was able to complete every single instance in the MathSAT-Small, MathSAT-Middle, JFS, and Grater sets within the timeout. In those four suites (totaling over 280 formulas), StageSAT had zero timeouts and solved all satisfiable instances, achieving 100% coverage. The MathSAT-Large suite remained the only source of timeouts for StageSAT (4 out of 49, as discussed). This means that across all benchmarks tested, StageSAT solved 45 (Large) + (all 35 Middle) + (all 130 Small) + (all 111 JFS) + (all 118 Grater) = 439 problems, missing only the 4 hardest large cases. By contrast, other incomplete solvers often left many problems unsolved (especially on the larger instances), and even complete solvers occasionally timed out on some medium or small benchmarks with complex formulas. StageSAT's ability to handle *every* instance in the easier suites underscores its robustness and efficiency even on simpler or moderate problems.

Looking at SAT coverage in particular – i.e. the fraction of truly satisfiable benchmarks for which a solver can find a model – StageSAT was the top performer on every suite against the incomplete solvers. It consistently identified the most solutions. For example, in the Grater benchmarks (many of which are satisfiable tricky cases), StageSAT found solutions for all satisfiable instances, whereas XSat, goSAT, or JFS missed some and/or gave up early. In the JFS suite, which was tailored to a fuzzing approach, StageSAT also managed to solve all satisfiable cases, effectively equaling or surpassing JFS's coverage but with a systematic method. Meanwhile, StageSAT's unsat-guess heuristic did not lead it astray on the smaller benchmarks either: on all problems where complete solvers reported UNSAT, StageSAT either also reported unsat-guess or, in many cases, was able to quickly decide unsat-guess before the complete solver timed out. There were no instances where StageSAT's answer disagreed with the known ground truth. This level of agreement gives confidence that StageSAT's approach scales down well in addition to scaling up.

Comparing StageSAT to complete solvers across all sets, we observe a complementary profile. Complete SMT solvers like Bitwuzla and cvc5 maintain perfect soundness (never guessing UNSAT) and can eventually solve every problem given enough time, but they often require significantly more time on satisfiable cases. StageSAT, on the other hand, excels in speed: across the board it dramatically reduced solve times on satisfiable benchmarks, with 3×–100× lower mean runtimes and one to two orders of magnitude lower median runtimes compared to the complete solvers in each category. Especially on the harder end of MathSAT-Large and on the JFS/Grater sets (which contain complex mathematical constraints), StageSAT's numeric search finds solutions much faster than exhaustive search, while the complete solvers tend to hit many timeouts or long runs. In terms of coverage, StageSAT is competitive: it solved as many or more instances as the best complete solver on Large, and it solved *every* instance on the other suites, whereas some complete solvers struggled with a handful of those (due to the exponential blow-up of bit-blasting on certain formulas). For example, on the Grater benchmarks, StageSAT had no timeouts, whereas some complete solvers timed out on a few; similarly for the JFS suite. This suggests that for these types of floating-point problems, StageSAT can serve as an effective front-line solver to quickly catch the easy satisfiable cases and many of the hard ones, thereby alleviating the load on complete solvers.

6 Related Work

Mainstream SMT Solvers for Floating-Point. Modern SMT solvers such as Z3 [9], cvc5 [2], MathSAT5 [7, 17], and Bitwuzla [24] represent the state-of-the-art in bit-precise reasoning for floating-point constraints. These solvers reduce floating-point formulas to lower-level theories (typically

bit-vectors via bit-blasting) and leverage DPLL(T)/CDCL(T) [8, 14, 21, 25] search to ensure completeness. This approach has proven highly effective on many benchmarks – indeed, Bitwuzla and cvc5 have dominated recent SMT-COMP competitions in floating-point divisions [22]. However, bit-precise methods can struggle with the enormous search space induced by low-level encodings, especially for complex numerical constraints. In contrast, StageSAT takes an alternative route: rather than exhaustive logical reasoning at the bit-level, it harnesses numerical optimization techniques [13, 23, 26] to navigate the search space. This strategy is fundamentally different and complementary to bit-blasting approaches. StageSAT avoids enumerating bit patterns explicitly, potentially scaling to constraints that are intractable for traditional solvers, while still ultimately producing bit-precise satisfying assignments. Unlike Bitwuzla's recent incorporation of bit-level local search heuristics [11], StageSAT operates at the numeric level, using a staged optimization process to guide the solver toward a solution. This novel perspective allows StageSAT to tackle floating-point constraints from a fresh angle, without directly competing with the intricate but heavyweight bit-precise procedures of Z3, cvc5, MathSAT, and Bitwuzla.

Heuristic and Fuzzing-Based Solvers. An orthogonal line of work has explored heuristic methods for SMT solving. Notably, JFS (the "Just Fuzz It" Solver) [20] applies coverage-guided fuzzing [4, 31] to floating-point satisfiability, randomly mutating inputs using feedback from solver executions. This approach can quickly find solutions for certain FP problems that confound traditional solvers. However, JFS is neither complete nor optimization-based: it cannot prove unsatisfiability and lacks a principled objective beyond code coverage heuristics. StageSAT is markedly different: we define a clear mathematical objective whose minimization corresponds to satisfying the formula. While JFS demonstrates the value of unconventional search strategies, StageSAT provides a more systematic approach grounded in optimization. Unlike JFS's random exploration, StageSAT leverages its optimization framework to navigate toward models with principled guidance rather than guessing.

Optimization-Based Floating-Point Solvers. Our work is most directly inspired by prior attempts to solve floating-point constraints via mathematical optimization. XSat [12] pioneered this direction by translating constraints into a real-valued optimization problem checked against IEEE-754 semantics [1]. This yielded a fast solver but could miss corner cases where real and floating-point solutions diverge. goSAT [3] formulated floating-point satisfiability as a global optimization problem, efficiently finding models for difficult FP problems but remaining incomplete and vulnerable to discontinuous regions. More recently, Grater [6] refined this paradigm using carefully constructed continuous objectives and gradient-based methods, achieving impressive performance matching or surpassing Bitwuzla and cvc5 on several benchmarks. parSAT [19] extends this work by parallelizing multiple stochastic optimization methods [18, 29, 30] on multi-core CPUs. Despite these advances, all prior optimization-based solvers operate in a single phase: they reduce the entire formula to one monolithic objective and apply an off-the-shelf optimizer. This one-stage approach leaves them vulnerable to local minima and irregular constraint landscapes with piecewise-defined semantics or abrupt discontinuities.

StageSAT distinguishes itself by its *staged optimization* strategy, which to our knowledge has not been explored in prior SMT solvers. Instead of a single optimization run, StageSAT breaks the solving process into multiple stages, each solving a sub-problem that incrementally moves closer to satisfiability. This enables StageSAT to surmount hurdles that tripped up earlier tools: it first solves a relaxed version of the constraints (smoothing out non-linear or discontinuous behaviors), then progressively re-introduces full floating-point semantics. This avoids local minima that one-shot methods fall into and handles discontinuities by isolating them in specific stages. None of XSat, goSAT, or Grater employs such a multi-phase scheme. This multi-stage design balances the

smoothness of numeric optimization with bit-level precision, resulting in a solver that is more robust on certain hard benchmarks than prior techniques.

7 Conclusion

We introduced StageSAT, a solver for satisfiability in the SMT logic QF_FP, which addresses the challenges of floating-point constraints with a novel three-stage architecture. By integrating numeric search with a bit-level refinement, StageSAT efficiently navigates the floating-point search space while ensuring solution correctness. This design is both accurate and practically effective, overcoming key limitations of prior numeric solvers that often suffer from imprecise search or incomplete results. In particular, StageSAT's lattice-refinement phase guarantees high precision, making it novel compared to earlier numeric SMT solvers that could not always ensure correct or complete results. Overall, StageSAT's multi-phase approach represents a principled advancement in floating-point SMT solving, combining the strengths of continuous optimization and discrete search in a way not seen in previous tools.

Our experimental results validate StageSAT's approach: it matched or exceeded the SAT recall of all other solvers in our benchmarks. It also solved more large, challenging benchmarks than any competing solver, attaining the highest coverage on the most complex problem sets. At the same time, StageSAT exhibited the lowest timeout rate among all tools evaluated, indicating its superior reliability on hard constraints. For example, on the largest benchmark category, StageSAT solved the most instances with the fewest timeouts, whereas the best alternative left several satisfiable problems unsolved. These concrete results demonstrate that StageSAT delivers robust and accurate performance in practice, substantially outperforming both prior numeric approaches and state-of-the-art bit-precise solvers on difficult floating-point problems. In conclusion, the StageSAT solver is novel, accurate, and effective – it advances the state of the art in QF_FP satisfiability by solving complex floating-point constraints with high precision and practical efficiency in our evaluation.

Data Availability

Our code is available at https://github.com/stagesat-solver/stagesat, and an artifact [32] on Zenodo (DOI: 10.5281/zenodo.19573062) contains all the benchmarks and instructions for reproducibility.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments and suggestions. This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Prime Contract No. HR001124C0492, by DARPA and the Naval Information Warfare Center Pacific (NIWC Pacific) under Contract No. N6600121-C-4028, and by the US National Science Foundation (NSF) under Grant No. CNS-2234257. Any views, opinions, findings, conclusions, or recommendations expressed in this paper are those of the author(s) and should not be interpreted as representing the official policies or positions, either expressed or implied, of these agencies. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] 2019. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), 1–84. doi:10.1109/IEEESTD.2019.8766229

[2] Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. 2022. cvc5: A Versatile and Industrial-Strength SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems: 28th International Conference, TACAS 2022, Held as Part of the European*

Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings, Part I (Munich, Germany). Springer-Verlag, Berlin, Heidelberg, 415–442. doi:10.1007/978-3-030-99524-9_24

[3] M. Ammar Ben Khadra, Dominik Stoffel, and Wolfgang Kunz. 2017. goSAT: Floating-point satisfiability as global optimization. In *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*, Daryl Stewart and Georg Weissenbacher (Eds.). IEEE, 11–14. doi:10.23919/FMCAD.2017.8102235

[4] Marcel Böhme, Van-Thuan Pham, and Abhik Roychoudhury. 2019. Coverage-Based Greybox Fuzzing as Markov Chain. *IEEE Transactions on Software Engineering* 45, 5 (2019), 489–506. doi:10.1109/TSE.2017.2785841

[5] Qian Chen, Chenqi Cui, Fengjuan Gao, Yu Wang, Ke Wang, and Linzhang Wang. 2025. Grater-experiment: artifact and code for "Solving Floating-Point Constraints with Continuous Optimization". https://github.com/grater-exp/grater-experiment. Commit `e405648`, accessed 13 Nov 2025.

[6] Qian Chen, Chenqi Cui, Fengjuan Gao, Yu Wang, Ke Wang, and Linzhang Wang. 2025. Solving Floating-Point Constraints with Continuous Optimization. *Proc. ACM Program. Lang.* 9, PLDI (2025), 725–747. doi:10.1145/3729279

[7] Alessandro Cimatti, Alberto Griggio, Bastiaan Schaafsma, and Roberto Sebastiani. 2013. The MathSAT5 SMT Solver. In *Proceedings of TACAS (LNCS, Vol. 7795)*, Nir Piterman and Scott Smolka (Eds.). Springer.

[8] Martin Davis and Hilary Putnam. 1960. A Computing Procedure for Quantification Theory. *J. ACM* 7, 3 (July 1960), 201–215. doi:10.1145/321033.321034

[9] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: an efficient SMT solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Budapest, Hungary) *(TACAS'08/ETAPS'08)*. Springer-Verlag, Berlin, Heidelberg, 337–340.

[10] Leonardo de Moura, Nikolaj Bjørner, and Eric Pony. 2012. Z3Py Tutorial. https://ericpony.github.io/z3py-tutorial/guide-examples.htm. Accessed 13 November 2025.

[11] Andreas Fröhlich, Armin Biere, Christoph M. Wintersteiger, and Youssef Hamadi. 2015. Stochastic Local Search for Satisfiability Modulo Theories. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, Blai Bonet and Sven Koenig (Eds.). AAAI Press, 1136–1143. doi:10.1609/AAAI.V29I1.9372

[12] Zhoulai Fu and Zhendong Su. 2016. XSat: A Fast Floating-Point Satisfiability Solver. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 9780)*, Swarat Chaudhuri and Azadeh Farzan (Eds.). Springer, 187–209. doi:10.1007/978-3-319-41540-6_11

[13] Zhoulai Fu and Zhendong Su. 2019. Effective floating-point analysis via weak-distance minimization. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. Association for Computing Machinery, New York, NY, USA, 439–452. doi:10.1145/3314221.3314632

[14] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. 2004. DPLL(T): Fast Decision Procedures. In *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings (Lecture Notes in Computer Science, Vol. 3114)*, Rajeev Alur and Doron A. Peled (Eds.). Springer, 175–188. doi:10.1007/978-3-540-27813-9_14

[15] David Goldberg. 1991. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.* 23, 1 (March 1991), 5–48. doi:10.1145/103162.103163

[16] Gene H. Golub and Charles F. Van Loan. 1996. *Matrix Computations, Third Edition.* Johns Hopkins University Press.

[17] Leopold Haller, Alberto Griggio, Martin Brain, and Daniel Kroening. 2012. Deciding floating-point logic with systematic abstraction. In *Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK, October 22-25, 2012*, Gianpiero Cabodi and Satnam Singh (Eds.). IEEE, 131–140. https://ieeexplore.ieee.org/document/6462565/

[18] P. Kaelo and M. M. Ali. 2006. Some Variants of the Controlled Random Search Algorithm for Global Optimization. *Journal of Optimization Theory and Applications* 130, 2 (2006), 253–264. doi:10.1007/s10957-006-9101-0

[19] Markus Krahl, Matthias Güdemann, and Stefan Wallentowitz. 2025. parSAT: Parallel Solving of Floating-Point Satisfiability. *CoRR* abs/2509.16237 (2025). arXiv:2509.16237 doi:10.48550/ARXIV.2509.16237

[20] Daniel Liew, Cristian Cadar, Alastair F. Donaldson, and J. Ryan Stinnett. 2019. Just fuzz it: solving floating-point constraints using coverage-guided fuzzing. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.). ACM, 521–532. doi:10.1145/3338906.3338921

[21] J.P. Marques-Silva and K.A. Sakallah. 1999. GRASP: a search algorithm for propositional satisfiability. *IEEE Trans. Comput.* 48, 5 (1999), 506–521. doi:10.1109/12.769433

[22] Martin Jonáš and François Bobot and David Déharbe and Dominik Winterer. 2025. SMT-COMP 2025: The 20th International Satisfiability Modulo Theories Competition. https://smt-comp.github.io/2025/. Accessed: 2025-11-12.

[23] W. Miller and D.L. Spooner. 1976. Automatic Generation of Floating-Point Test Data. *IEEE Transactions on Software Engineering* SE-2, 3 (1976), 223–226. doi:10.1109/TSE.1976.233818

[24] Aina Niemetz and Mathias Preiner. 2023. Bitwuzla. In *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 13965)*, Constantin Enea

and Akash Lal (Eds.). Springer, 3–17. doi:10.1007/978-3-031-37703-7_1
[25] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. 2006. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(*T*). *J. ACM* 53, 6 (2006), 937–977. doi:10.1145/1217856.1217859
[26] Jorge Nocedal and Stephen J. Wright. 1999. *Numerical Optimization.* Springer. doi:10.1007/B98874
[27] R. Penrose. 1955. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society* 51, 3 (1955), 406–413. doi:10.1017/S0305004100030401
[28] Mathias Preiner, Hans-Jörg Schurr, Clark Barrett, Pascal Fontaine, Aina Niemetz, and Cesare Tinelli. 2025. *SMT-LIB release 2025 (non-incremental benchmarks).* https://zenodo.org/records/16740866 SMT-LIB non-incremental benchmark library.
[29] T.P. Runarsson and Xin Yao. 2005. Search biases in constrained evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 35, 2 (2005), 233–243. doi:10.1109/TSMCC.2004.841906
[30] David J. Wales and Jonathan P. K. Doye. 1997. Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. *The Journal of Physical Chemistry A* 101, 28 (July 1997), 5111–5116. doi:10.1021/jp970984n
[31] Michał Zalewski. 2013. Technical whitepaper for afl-fuzz. http://lcamtuf.coredump.cx/afl/technical_details.txt. Accessed 13 Nov 2025.
[32] Yuanzhuo Zhang. 2026. *Artifact for [Scalable Floating-Point Satisfiability via Staged Optimization].* doi:10.5281/zenodo.19573062

Received 2025-11-13; accepted 2026-04-03