

LLM-Guided Quantified SMT Solving over Uninterpreted Functions

Kunhang Lv^{1,3}, Yuhang Dong^{2,3}, Rui Han^{1,3}, Fuqi Jia^{1,3}, Feifei Ma^{2,3*}, Jian Zhang^{1,3*}

¹SKLCS and Key Laboratory of System Software, ISCAS, Beijing, China

²Laboratory of Parallel Software and Computational Science, ISCAS, Beijing, China

³University of Chinese Academy of Sciences, Beijing, China

{lvkh, maff, zj}@ios.ac.cn

Abstract

Quantified formulas with Uninterpreted Functions (UFs) over non-linear real arithmetic pose fundamental challenges for Satisfiability Modulo Theories (SMT) solving. Traditional quantifier instantiation methods struggle because they lack semantic understanding of UF constraints, forcing them to search through unbounded solution spaces with limited guidance. We present AquaForte, a framework that leverages Large Language Models to provide semantic guidance for UF instantiation by generating instantiated candidates for function definitions that satisfy the constraints, thereby significantly reducing the search space and complexity for solvers. Our approach preprocesses formulas through constraint separation, uses structured prompts to extract mathematical reasoning from LLMs, and integrates the results with traditional SMT algorithms through adaptive instantiation. AquaForte maintains soundness through systematic validation: LLM-guided instantiations yielding SAT solve the original problem, while UNSAT results generate exclusion clauses for iterative refinement. Completeness is preserved by fallback to traditional solvers augmented with learned constraints. Experimental evaluation on SMT-COMP benchmarks demonstrates that AquaForte solves numerous instances where state-of-the-art solvers like Z3 and CVC5 timeout, with particular effectiveness on satisfiable formulas. Our work shows that LLMs can provide valuable mathematical intuition for symbolic reasoning, establishing a new paradigm for SMT constraint solving.

Code — <https://github.com/kylelv2000/Aqua-Forte>

Introduction

Satisfiability Modulo Theories (SMT) solving (Kroening and Strichman 2016; Barrett et al. 2021) has become an indispensable computational foundation for numerous applications including software verification (Beyer, Dangl, and Wendler 2018), program analysis (Gavrilenko et al. 2019; Zhang and Wang 2001), model checking (Cordeiro, Fischer, and Marques-Silva 2009), and automated test generation (Peleska, Vorobev, and Lapschies 2011; Zhang 2000). Among the various SMT theories, Quantified Uninterpreted Functions with Non-linear Integer and Real Arithmetic

(QUFNIRA) poses particularly formidable computational challenges. This theory naturally emerges when modeling hybrid systems that combine discrete control logic with continuous dynamics (Cimatti, Mover, and Tonetta 2013; Cimatti et al. 2015), requiring reasoning about both integer and real-valued variables alongside abstract computational components represented as uninterpreted functions.

The undecidability of QUFNIRA stems from the interaction of quantifiers over infinite mixed integer-real domains, uninterpreted functions, and non-linear arithmetic constraints (Becker, Müller, and Summers 2019; Bjørner and Nachmanson 2020). This forces practical solvers to rely on incomplete instantiation techniques, where success critically depends on generating effective quantifier instantiations that can bridge discrete and continuous reasoning.

Current state-of-the-art approaches for QUFNIRA predominantly rely on syntactic instantiation techniques, most notably E-matching (De Moura and Bjørner 2007; Reynolds, Barbosa, and Fontaine 2018) and model-based quantifier instantiation (MBQI) (Ge and De Moura 2009). E-matching identifies instantiation opportunities through syntactic pattern matching within the solver’s congruence closure, while MBQI iteratively constructs candidate models and verifies their consistency against quantified constraints. Leading solvers such as Z3 (de Moura and Bjørner 2008) and CVC5 (Barbosa et al. 2022) employ increasingly sophisticated heuristics to guide these processes, yet they remain fundamentally limited by their purely syntactic treatment of uninterpreted functions.

This syntactic approach creates a profound disconnect between the mathematical richness of real-world applications and the solver’s reasoning capabilities. In practice, uninterpreted functions often represent well-understood mathematical concepts—distance metrics exhibiting triangle inequality properties, cost functions with monotonicity constraints, or physical transformations preserving certain invariants. However, current solvers treat these functions as completely opaque symbols, constrained only by functional congruence. This semantic blindness forces solvers to explore enormous instantiation spaces without leveraging mathematical intuition that could dramatically reduce search complexity.

The limitations of syntactic approaches manifest through two critical challenges that significantly impact solving performance. **Semantic opacity** prevents solvers from exploit-

*Corresponding authors.

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

ing inherent mathematical properties of uninterpreted functions, causing them to miss natural simplifications and structural relationships that could lead to immediate proof discovery or efficient refutation. **Instantiation inefficiency** results from the inability to generate semantically meaningful ground terms, leading to extensive exploration of irrelevant instantiation spaces that provide little useful information for the underlying decision procedures.

To address these fundamental limitations, we propose AquaForte, a novel framework that integrates Large Language Models (LLMs) with traditional SMT solving to provide semantic guidance for QUFNIRA. Our approach leverages LLMs’ extensive training on mathematical literature to analyze the contextual usage patterns of uninterpreted functions within constraint systems and generate semantically plausible concrete instantiations. Rather than treating uninterpreted functions as purely syntactic objects, our method employs LLMs to infer concrete mathematical definitions that preserve the essential behavioral properties while providing the algebraic structure necessary for efficient reasoning over mixed integer-real arithmetic constraints.

The key insight is that LLMs can identify semantic patterns in uninterpreted function usage that escape syntactic analysis, then generate concrete function instantiations preserving essential mathematical properties (Frieder et al. 2023). This semantic-guided instantiation enables solvers to work with structured mathematical expressions rather than opaque symbols, dramatically improving reasoning efficiency over mixed arithmetic domains.

Our contributions include: (1) the first systematic methodology for leveraging LLM semantic understanding to guide quantifier instantiation in QUFNIRA, transcending the limitations of purely syntactic pattern matching; (2) a soundness-preserving integration framework that provides flexible compatibility with diverse solvers, maintaining formal verification guarantees while maximizing the benefits of LLM mathematical intuition; and (3) comprehensive experimental evaluation across 1,481 benchmark instances demonstrating substantial performance improvements: 80.0% increase in solved instances for Z3 and 183.6% improvement for CVC5, with particularly significant gains on satisfiable cases where semantic guidance proves most impactful.

Preliminaries

We briefly review the mathematical foundations for SMT solving over uninterpreted functions with quantification, focusing on the challenging combination that motivates our LLM-guided approach.

SMT Foundations

Satisfiability Modulo Theories (SMT) extends propositional satisfiability by incorporating first-order theories. An SMT instance consists of a quantifier-free first-order formula φ over a background theory \mathcal{T} .

Definition 1 (SMT Problem). *Given a theory \mathcal{T} and a formula φ , the SMT problem asks whether there exists a \mathcal{T} -model \mathcal{M} such that $\mathcal{M} \models \varphi$. If such a model exists, φ is \mathcal{T} -satisfiable; otherwise, it is \mathcal{T} -unsatisfiable.*

Modern SMT solvers such as Z3 and CVC5 employ the DPLL(T) framework (Nieuwenhuis, Oliveras, and Tinelli 2006). Our work focuses on the challenging combination of real arithmetic $\mathcal{T}_{\mathbb{R}}$ and uninterpreted functions \mathcal{T}_{UF} :

Real Arithmetic Theory $\mathcal{T}_{\mathbb{R}}$ enables reasoning about polynomial constraints over real numbers:

$$\text{Terms: } t ::= x \mid c \mid t_1 + t_2 \mid t_1 \cdot t_2 \mid -t \quad (1)$$

$$\text{Formulas: } \varphi ::= t_1 = t_2 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \quad (2)$$

where x ranges over real variables and $c \in \mathbb{Q}$.

Uninterpreted Functions Theory \mathcal{T}_{UF} introduces function symbols constrained solely by the congruence axiom:

$$\forall f \in \mathcal{F}, \forall \mathbf{x}, \mathbf{y} : \bigwedge_{i=1}^n (x_i = y_i) \rightarrow f(\mathbf{x}) = f(\mathbf{y}) \quad (3)$$

Definition 2 (NIRA). *The theory of Non-linear Integer and Real Arithmetic (NIRA), denoted as \mathcal{T}_{NIRA} , combines non-linear arithmetic constraints over integer and real domains.*

This theory represents one of the most challenging fragments in SMT solving.

Definition 3 (Quantified NIRA). *When extended with quantification, NIRA formulas take the form:*

$$\varphi ::= QF\text{-formula} \mid (\forall \vec{x} : \vec{S}. \varphi) \mid (\exists \vec{x} : \vec{S}. \varphi) \quad (4)$$

where $\vec{x} = (x_1, \dots, x_n)$ are distinct variables with sorts $\vec{S} = (S_1, \dots, S_n)$, and each $S_i \in \{\mathbb{Z}, \mathbb{R}\}$, and QF-formula denotes the quantifier-free formulas defined previously.

The computational complexity critically depends on *quantifier alternation depth*. For instance, in Presburger arithmetic, formulas with bounded quantifier alternation have lower complexity than the general 2EXPTIME-complete case (Fischer and Rabin 1998).

Solving Techniques for Quantified NIRA

Given the high computational complexity of quantified NIRA, practical SMT solvers employ *quantifier instantiation* techniques to reduce quantified formulas to ground instances (Reynolds et al. 2013). For universally quantified formula $\forall x. \varphi(x)$ where x ranges over mixed integer-real domains, instantiation produces:

$$\forall x. \varphi(x) \rightsquigarrow \bigwedge_{t \in T_{\mathbb{Z}} \cup T_{\mathbb{R}}} \varphi(t) \quad (5)$$

where $T = \{t_1, \dots, t_n\}$ is a strategically chosen set of ground terms from both integer and real domains.

E-matching with Trigger Patterns. The predominant instantiation approach in modern SMT solvers (de Moura and Bjørner 2008) uses trigger-based pattern matching. For quantified subformula $\forall \vec{x}. \psi(\vec{x})$, a *trigger pattern* is a set of terms $\{t_1(\vec{x}), \dots, t_k(\vec{x})\}$ that appear in $\psi(\vec{x})$ and collectively mention all quantified variables. The solver instantiates the quantified formula only when ground terms matching these patterns appear during search.

Example 1 (E-matching Process). *Given $\forall x. P(f(x)) \wedge f(x) > 0$ with trigger pattern $f(x)$, and ground terms $\{f(3), f(a), 5\}$ in the current context, E-matching instantiates the formula at $x = 3$ and $x = a$, producing $P(f(3)) \wedge f(3) > 0 \wedge P(f(a)) \wedge f(a) > 0$.*

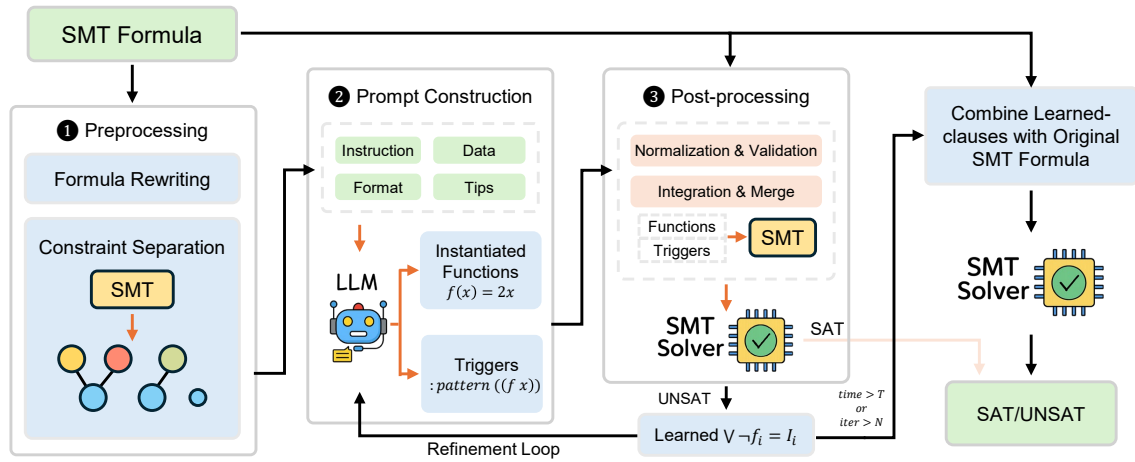


Figure 1: Overview of AquaForte.

Model-Based Quantifier Instantiation (MBQI). MBQI (Ge and De Moura 2009) constructs candidate models for quantifier-free portions, then validates quantified constraints. When a quantified formula is violated in the candidate model, the violating assignment provides witness terms for subsequent instantiation. This approach is more semantic than pure pattern matching but remains limited by the black-box treatment of uninterpreted functions.

Large Language Models and Prompt

Large Language Models (LLMs) are neural networks based on the Transformer architecture (Vaswani et al. 2017), trained on vast text corpora to predict the next token in a sequence. Modern LLMs like GPT-4 (Achiam et al. 2023) demonstrate emergent mathematical reasoning capabilities (Lewkowycz et al. 2022; Wei et al. 2022a) through scale and self-attention mechanisms that capture long-range dependencies, developing internal representations that encode semantic relationships and domain-specific knowledge.

Prompt Engineering. LLM performance critically depends on *prompt design*—the systematic construction of input prompts to elicit desired behaviors (Liu et al. 2023; Sahoo et al. 2024). Effective prompts include: (1) *Task specification* with clear instructions; (2) *Context provision* with relevant background; (3) *Output formatting* specifying desired structure; (4) *Few-shot examples* demonstrating expected patterns (Brown et al. 2020). Advanced techniques include chain-of-thought prompting (Wei et al. 2022b) for step-by-step reasoning and role-based prompting to invoke domain expertise.

Example 2 (Structured Prompt Design). *A mathematical prompt might follow: “You are a mathematics expert. Given constraints: [problem], analyze relationships and provide: [format]. Example: [demonstration]. Requirements: [constraints].”*

Integration with Formal Systems. The complementary nature of LLMs and formal systems enables hybrid architectures where LLMs provide semantic insights and hypothesis

generation, while symbolic systems ensure logical correctness and verification. This synergy combines intuitive reasoning with formal precision.

Methodology

Figure 1 presents **AquaForte**, a novel framework that leverages Large Language Models to provide semantic guidance for uninterpreted functions in SMT solving. This section describes our approach through four main components: an overview with illustrative examples, uninterpreted function instantiation techniques, auxiliary strategies for trigger generation, and integration with traditional SMT algorithms.

Overview

To illustrate our approach, consider the following SMT formula containing constraints on uninterpreted functions:

$$\varphi = \forall x \in \mathbb{R}. f(2x) = 2x \wedge g(2x) = 2x \quad (6)$$

This formula defines both functions f and g to have identical behavior at scaled inputs: $f(2x) = g(2x) = 2x$ for all x . From a mathematical perspective, it is intuitive that $f(x) = g(x)$ should hold universally. However, traditional SMT solvers struggle to derive this equivalence relationship and often timeout when queried about $\forall x. f(x) = g(x)$.

Traditional SMT solvers rely on axiomatization-based derivation methods, constructing proof processes through predefined inference rules and quantifier instantiation patterns. However, since the constraint only provides information about f and g at points of the form $2x$, the solver cannot directly establish equivalence at arbitrary points x . The instantiation process becomes an infinite search that fails to find contradictions, leading to timeout.

Our core insight is to leverage LLM’s mathematical intuition to recognize semantic relationships between functions. Based on this semantic understanding, the LLM suggests candidate function instantiations such as $f(x) = x$ and $g(x) = x$, which satisfy the original constraints and make the equivalence $f(x) = g(x)$ immediately verifiable. We instantiate uninterpreted functions with concrete mathematical

Algorithm 1: Formula Rewriting and Purification

```
1: Input: Formula  $\varphi$ , uninterpreted functions  $\mathcal{F} = \{f_1, \dots, f_k\}$ 
2: Output: Formula components  $\mathcal{C} = \{C_1, \dots, C_m\}$ 
3:  $\varphi' \leftarrow \text{REWRITEFORMULA}(\varphi)$  {Apply simplifications and expansions}
4: Initialize union-find structure  $UF$  with functions  $\mathcal{F}$ 
5:  $constraints \leftarrow \text{EXTRACTCONSTRAINTS}(\varphi')$ 
6: for each constraint  $c \in constraints$  do
7:    $funcs \leftarrow \text{GETUNINTERPRETEDFUNCTIONS}(c)$ 
8:   for each pair  $(f_i, f_j) \in funcs \times funcs$  do
9:      $\text{UNION}(UF, f_i, f_j)$ 
10:  end for
11: end for
12: Group constraints by their representatives in  $UF$ 
13: return Connected components  $\mathcal{C}$ 
```

expressions via semantic analysis, thereby eliminating quantified reasoning bottlenecks and accelerating SAT solving.

Uninterpreted Function Instantiation

This process comprises three stages: Preprocessing, LLM Instantiation, and Post-processing, transforming raw SMT formulas into semantically meaningful instantiations.

Preprocessing Algorithm 1 details the preprocessing stage that applies two techniques to reduce cognitive complexity while maintaining mathematical structure.

Formula Rewriting. We apply semantic-preserving transformations: (1) formula simplification through expression normalization, redundant operation elimination, and tautology / contradiction reduction to Boolean constants; (2) interpreted function expansion to expose mathematical semantics.

Constraint Separation. Using union-find structures, we identify connected components of constraints sharing uninterpreted functions. Functions co-occurring in any constraint merge into the same component, decomposing the formula into independent subproblems. This reduces individual query complexity from $O(|\mathcal{F}|^2)$ to $O(|C_i|^2)$ where C_i denotes the i -th component, potentially achieving $|C_i| \ll |\mathcal{F}|$ when constraints are well-separated.

LLM Instantiation For each connected component C_i , we use a Large Language Model to generate concrete instantiations $\mathcal{I} = \{I_1, I_2, \dots, I_k\}$ that materialize each uninterpreted function $f_i \in \mathcal{F}$ into explicit mathematical definitions such as polynomial functions, piecewise functions, or other computable constructs.

We construct a structured prompt \mathcal{P}_i as:

$$\mathcal{P}_i = \langle \mathcal{S}_{instruction}, \mathcal{D}_{data}, \mathcal{O}_{format}, \mathcal{T}_{tips} \rangle$$

where $\mathcal{S}_{instruction}$ establishes the mathematical interpretation task, \mathcal{D}_{data} provides the constraints from component C_i and uninterpreted function signatures, \mathcal{O}_{format} defines a structured JSON output format requiring reasoning chains and confidence assessments, and \mathcal{T}_{tips} specifies implementation constraints for valid SMT-LIB syntax (Barrett, Fontaine, and Tinelli 2025).

The prompt instructs the LLM to “understand the mathematical meaning of the SMT formula and reason from a mathematical perspective to derive the concrete form of the uninterpreted function.” This hierarchical approach ensures mathematically grounded instantiations while constraining the output space to valid SMT-LIB (Barrett et al. 2010) constructs. The structured output format promotes transparency by requiring explicit justification for each proposed instantiation, enabling quality assessment and seamless integration with existing SMT solver infrastructure.

Post-processing The LLM output undergoes a systematic validation pipeline to ensure correctness and compatibility with SMT solvers. First, we normalize the raw response by extracting the structured JSON definitions while discarding any conversational text. Each proposed instantiation is then parsed and type-checked against SMT-LIB syntax requirements; invalid expressions trigger a re-query to the LLM with corrective feedback.

Finally, instantiations from all components are merged. During this crucial step, we maintain a global symbol table to detect and resolve potential issues such as *variable name conflicts* while verifying type consistency across function signatures. This systematic pipeline transforms raw LLM output into validated, syntactically correct instantiations ready for integration into the original SMT formula.

Quantifier Elimination Strategies

Our trigger generation approach leverages large language models to automatically synthesize instantiation patterns from quantified formula constraints. Given a quantified formula φ , we directly present the logical constraints to the LLM and task it with generating effective trigger patterns.

The LLM processes these quantified constraints and generates candidate triggers $T = \{t_1, t_2, \dots, t_k\}$ that capture essential instantiation patterns. Each generated trigger t_i represents a heuristic rule that identifies when specific quantifier instantiations are likely to be productive for proof search. The key insight is that modern LLMs possess sufficient logical reasoning capabilities to recognize structural patterns within quantified formulas and translate them into actionable instantiation strategies.

The generated triggers undergo systematic validation to ensure syntactic correctness and semantic consistency. We employ filtering mechanisms to remove malformed patterns and consolidate semantically equivalent triggers to prevent redundancy. The validated triggers are then directly integrated into the quantifier instantiation module, where they guide the selection of instantiation terms during automated proof search.

Integration with Traditional Algorithms

We present a hybrid framework that systematically integrates LLM guidance with traditional SMT solving through iterative refinement cycles, where failed instantiation attempts provide structured feedback for subsequent queries.

Algorithm 2 queries the LLM with augmented information $\mathcal{Q} = \langle \varphi, history, \mathcal{F} \rangle$ (line 4), where φ is the original formula, *history* contains previous failures with contexts,

Algorithm 2: Adaptive LLM-Guided SMT Solving

```
1: Input: Formula  $\varphi$ , max iterations  $N$ , time budget  $T$ 
2: Output: SAT, UNSAT, or UNKNOWN
3: Initialize:  $history \leftarrow \emptyset$ ,  $learned \leftarrow \emptyset$ ,  $iter \leftarrow 0$ 
4:  $start \leftarrow \text{TIME}()$ 
5: while  $iter < N$  and  $\text{TIME}() - start < T$  do
6:    $(\mathcal{I}, \mathcal{T}) \leftarrow \text{QUERYLLM}(\varphi, history)$  {Get instantiations}
7:    $\varphi_{inst} \leftarrow \text{APPLYINSTANTIATIONS}(\varphi, \mathcal{I})$ 
8:    $\varphi_{inst} \leftarrow \text{ADDTRIGGERS}(\varphi_{inst}, \mathcal{T})$ 
9:    $result \leftarrow \text{SMTSOLVER}(\varphi_{inst})$  {Bounded solving}
10:  if  $result = \text{SAT}$  then
11:    return SAT
12:  else if  $result = \text{UNSAT}$  then
13:     $\psi \leftarrow \bigvee_{i=1}^k \neg(f_i = I_i)$  {Exclusion clause}
14:     $history \leftarrow history \cup \{(\mathcal{I}, \text{refuted})\}$ 
15:     $learned \leftarrow learned \cup \{\psi\}$ 
16:  else
17:     $history \leftarrow history \cup \{(\mathcal{I}, \text{timeout})\}$ 
18:  end if
19:   $iter \leftarrow iter + 1$ 
20: end while
21: Fallback:
22: return  $\text{SMTSOLVER}(\varphi \wedge \bigwedge learned)$ 
```

and \mathcal{F} specifies uninterpreted functions requiring instantiation. This enriched context enables the LLM to avoid unsuccessful interpretations and explore alternative mathematical relationships. The LLM response $(\mathcal{I}, \mathcal{T})$ provides instantiations and trigger patterns, systematically applied to construct the modified formula φ' (lines 5-6).

The validation phase (lines 7-9) applies bounded SMT solving with time limit τ_1 . Upon satisfiability, the algorithm terminates with SAT. Otherwise, failure analysis records the failed instantiation \mathcal{I} in $history$ (line 12) and generates exclusion clause $\psi = \bigvee_{i=1}^k \neg(f_i = I_i)$ to constrain future search (line 13).

Upon reaching iteration threshold N or time budget T (line 20), systematic fallback consolidates all exclusion clauses into $\varphi_{augmented} = \varphi \wedge \bigwedge learned_clauses$. The complete SMT solver operates on this constraint-enriched formula, where learned clauses eliminate previously explored interpretation regions, often yielding improved performance over naive application.

This fallback mechanism preserves the correctness and does not weaken the base solver: when LLM-guided instantiations expire within resource limits, the procedure reduces to the traditional SMT solving over the accumulated constraints, matching the decision capability of the base solver in the worst case.

Experimental Evaluation

We conduct a comprehensive empirical evaluation to assess the effectiveness of our proposed **AquaForte (AF)** framework for LLM-guided SMT solving. To facilitate reproducibility, our code and benchmarks are publicly available

on GitHub. Our evaluation is designed to answer the following research questions:

- **RQ1:** How effective is LLM-guided instantiation compared to state-of-the-art SMT solvers?
- **RQ2:** Does increased computational time benefit traditional solvers versus LLM-guided approaches?
- **RQ3:** What is the effect of increasing LLM iteration counts on the number of problem instances solved?

Benchmark Suite

SMT-COMP 2025 Benchmarks We evaluated our approach in the entire UFNIRA and UFLRA benchmark sets from SMT-LIB, comprising 281 instances in total. These categories combine uninterpreted functions with arithmetic constraints—UFNIRA includes non-linear integer and real arithmetic, while UFLRA focuses on linear real arithmetic.

Custom Benchmark Construction To assess specific algorithmic capabilities beyond standard benchmarks, we developed two specialized datasets that will be released alongside our source code.

Sum-of-Squares (SOS) Verification Dataset: Inspired by Hilbert’s 17th problem, we created 600 instances for sum-of-squares decomposition. Each instance determines whether a polynomial F can be expressed as exactly three squares:

$$F(\mathbf{x}) = g_1^2(\mathbf{x}) + g_2^2(\mathbf{x}) + g_3^2(\mathbf{x}) \quad (7)$$

where $\mathbf{x} = (x_1, \dots, x_n)$.

We employ constructive generation by creating m source polynomials and computing $F = \sum_{i=1}^m p_i^2$. Problem complexity is controlled via variable dimension $n \in \{1, 2, 3\}$ and source count $m \in \{1, 2, 3, 4\}$. The SMT encoding seeks uninterpreted functions f_a, f_b, f_c satisfying:

$$\forall \mathbf{x} : f_a^2(\mathbf{x}) + f_b^2(\mathbf{x}) + f_c^2(\mathbf{x}) = F(\mathbf{x}) \quad (8)$$

Instances with $m \leq 3$ are satisfiable, while $m = 4$ cases require “compressing” four squares into three, creating challenging algebraic reasoning problems. The dataset provides 12 parameter configurations with 50 instances each.

Mathematical Functions Dataset (MFD): We constructed 600 randomly generated UFNIRA instances across four categories (150 each): **Rational function inequalities** combine existential quantifiers with comparison operators over function evaluations at randomly sampled domain points ($\exists x : f(x) \neq f(0)$). **Piecewise function inequalities** partition the real domain into 2-4 intervals with randomly assigned linear expressions and inequality constraints at boundary points. **Recursive function problems** employ template instantiation of recurrence patterns with varying recursion depths ($g(x) = f(x) + g(x-1)$), combined with randomly selected initial conditions and positivity constraints. **Function limit problems** encode linear algebraic properties (additivity: $f(x + y) = f(x) + f(y)$, homogeneity: $f(kx) = k \cdot f(x)$) with automatically generated existential boundary conditions that test contradictory inequalities.

Our complete benchmark suite totals 1481 instances, spanning standard verification benchmarks and specialized mathematical reasoning problems.

Method	SMT-COMP (281 instances)				MFD (600 instances)				SOS (600 instances)			
	SAT	Time(s)	UNSAT	Time(s)	SAT	Time(s)	UNSAT	Time(s)	SAT	Time(s)	UNSAT	Time(s)
Z3 Solver	16	0.15	104	0.02	275	0.08	29	0.02	12	0.02	0	/
AF+Z3 (GPT-4.1)	58	0.09*	107	0.10*	429	0.07*	29	0.08*	162	0.05*	0	/
CVC5 Solver	0	/	91	0.70	108	0.01	27	0.01	0	/	0	/
AF+CVC5 (GPT-4.1)	50	0.03*	95	0.94*	311	0.04*	26	0.06*	159	0.03*	0	/

Table 1: Detailed performance breakdown across benchmark suites with 24-second timeout. Execution times shown exclude LLM inference latency (mean: 7.60s).

Experimental Setup

Hardware and Environment. All experiments were conducted on a server with AMD EPYC 7763 64-core processor (2.45 GHz) and 512 GB RAM, ensuring consistent experimental conditions.

SMT Solvers. We evaluate our approach using the two most prominent state-of-the-art SMT solvers: Z3 (v4.15.0) and CVC5 (v1.2.1), which represent the current mainstream solutions for quantified formula solving.

LLMs. We employ three widely used LLMs: GPT-4.1, DeepSeek-V3 (DeepSeek et al. 2024), and Claude-4-Sonnet, each configured with temperature 0.01 for deterministic responses.

Timeout Configuration. We primarily use a 24-second timeout, following SMT-COMP’s standard for rapid evaluation. We also conduct experiments with 1200-second timeout to assess long-term scalability.

Iteration Strategy. Given the computational cost of LLM inference, we default to single-iteration calls (N=1) unless specified otherwise, making our approach practical for real-world deployment.

RQ1: Overall Performance Comparison

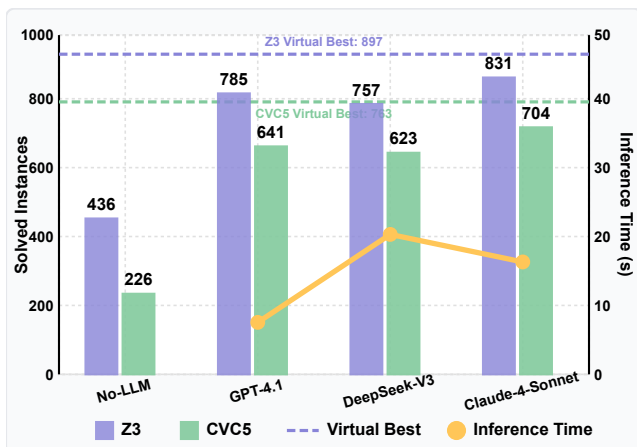


Figure 2: Total solved instances across different LLM-solver combinations with 24s timeout. Virtual Best represents the union of all LLM-enhanced configurations.

Figure 2 presents the overall performance comparison across different LLM-solver combinations. The results

demonstrate substantial improvements over baseline solvers: Z3 benefits from an 80.0% improvement with GPT-4.1 (785 vs. 436 instances), while CVC5 shows an even more dramatic 183.6% improvement (641 vs. 226 instances). Claude-4-Sonnet achieves the best single-model performance with 831 solved instances for Z3, representing a 90.6% improvement over the baseline. The LLM inference times vary across models, with GPT-4.1 requiring 7.6 seconds, Claude-4-Sonnet 16.56 seconds, and DeepSeek-V3 20.66 seconds on average.

The Virtual Best configuration solves 897 instances with Z3 and 763 with CVC5, demonstrating complementary strengths among different LLMs. The combined approach surpasses the baselines by 105.7% and 237.6%, respectively.

Detailed Benchmark Analysis

Table 1 presents a detailed breakdown of performance across individual benchmark suites. Our approach achieves particularly notable improvements on satisfiable (SAT) instances. This superior performance indicates that LLMs are highly effective at proposing appropriate instantiations that lead to satisfying assignments, leveraging their semantic understanding of underlying mathematical relationships. In addition, the approach can effectively accelerate the solver’s performance.

Comparison of Different Synthesis Strategies

Method	SAT	UNSAT	Total solved	Improvement
Base	108	118	226	0.0%
MBQI	256	108	364	+61.1%
CEGQI	337	52	389	+72.1%
ENUM	73	52	125	-44.7%
GPT-4.1(Ours)	520	121	641	+183.6%

Table 2: Comparison of cvc5 strategies with 24s timeout.

Recent work in function synthesis also offers several methods that can address our task. Mainstream approaches include MBQI, counterexample-guided quantifier instantiation (CEGQI) (Reynolds et al. 2015), and enumeration (Reynolds et al. 2019) over SyGuS grammars. These methods are available in the state-of-the-art solver cvc5, so we compare our solver against the corresponding cvc5 strategies under a uniform 24s timeout per instance. As shown in Table 2, these recent methods yield solid gains,

but still fall short of our approach by a clear margin. Just as important, these techniques do not conflict with our LLM-guided design. They are complementary: LLM proposals can seed or steer CEGQI and help prune or rank enumerations, suggesting a potential path to further improve solve rates.

RQ2: Efficiency Analysis: Rapid Solution Discovery

Method (N=1)	24s	1200s	Improvement
Z3 Solver	436	439	+0.7%
AF+Z3 (GPT-4.1)	785	788	+0.4%
CVC5 Solver	226	226	+0.0%
AF+CVC5 (GPT-4.1)	641	641	+0.0%

Table 3: Performance under different timeout constraints

Table 3 shows that both traditional SMT solvers and LLM-guided approaches gain little from longer timeout (24s to 1200s), indicating more time alone helps little. For traditional solvers, this reflects fundamental search strategy limitations; for LLM-guided methods, this confirms that single LLM calls either succeed rapidly or require additional iterations rather than extended runtime.

RQ3: Multi-Iteration Analysis

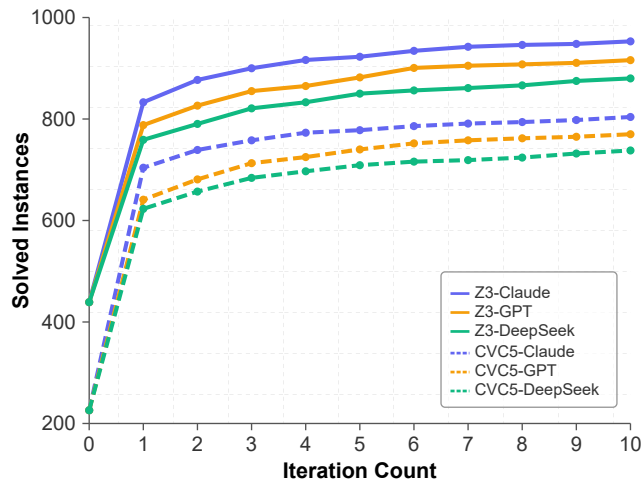


Figure 3: Performance across multiple iterations with 1200s timeout.

Figure 3 evaluates the impact of iterative LLM guidance on solver performance. Starting from identical baselines (Z3: 439, CVC5: 226), all combinations demonstrate consistent improvements with increased iterations. Comparing single iteration (N=1) to ten iterations (N=10), Claude achieves improvements of 14.4% for Z3 (833→953) and 14.2% for CVC5 (704→804). On average across all LLMs, Z3 improves by 15.5% and CVC5 by 17.6% from iteration

1 to 10. This demonstrates that iterative refinement significantly enhances performance, with diminishing returns observed after 5-7 iterations.

Discussion and Insights

Our evaluation reveals where LLM-guided SMT solving excels. The approach shows striking asymmetric performance: 3.6× improvement on SAT instances versus minimal gains on UNSAT instances, indicating LLMs excel at proposing satisfying instantiations through semantic pattern recognition but struggle with exhaustive unsatisfiability proofs. This makes the approach particularly valuable for model finding tasks. While individual LLMs exhibit varying reasoning capabilities in solving different problem instances, they demonstrate certain complementarity. The Virtual Best configuration achieves +66 instances over the best single model, indicating that ensemble approaches can effectively combine different LLM reasoning patterns. When resources permit, multiple iterations consistently improve solving rates, though diminishing returns emerge after several iterations. Unlike traditional solvers, our approach provides substantial gains with increased computational budget.

Related Work

Recent work has explored employing LLMs in constraint solving and logical reasoning. Logic LM (Pan et al. 2023) combines LLMs with logical forms for enhanced logical reasoning, while others focus on problem formulation: Logic.py (Kesseli, O’Hearn, and Cabral 2025) for search problems, multi-agent systems for logic puzzles (Berman, McKeown, and Ray 2024), and LLM-Sym for symbolic execution (Wang et al. 2024). However, these approaches target high-level formulation rather than instantiation mechanisms.

Traditional instantiation methods rely on syntactic heuristics, but theory-specific approaches have emerged leveraging domain properties. Notable examples include counterexample-guided instantiation for linear arithmetic (Reynolds, King, and Kunčak 2017), induction-based techniques for algebraic datatypes (Reynolds and Kunčak 2015), and QSMA’s model interpolation with look-ahead strategies (Bonacina, Graham-Lengrand, and Vauthier 2023). These methods still provide limited guidance for uninterpreted functions, where semantic understanding could significantly improve instantiation effectiveness.

Conclusion

We presented AquaForte, a novel framework that uses LLMs to guide quantified SMT solving over uninterpreted functions. By converting abstract function symbols into concrete mathematical expressions, our approach addresses semantic opacity and instantiation inefficiency in traditional solvers. Results show significant improvements on satisfiable instances but limited gains on unsatisfiable ones. Different LLMs exhibit complementary strengths suggesting potential ensemble methods. Our work demonstrates how LLMs can provide valuable semantic insights for symbolic reasoning while maintaining formal guarantees.

Acknowledgments

We are grateful to the anonymous reviewers for their comments and suggestions. This work has been supported by the National Natural Science Foundation of China (NSFC) under grant No.62132020 and grant No.62572461.

References

- Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Barbosa, H.; Barrett, C. W.; Brain, M.; Kremer, G.; Lachnitt, H.; Mann, M.; Mohamed, A.; Mohamed, M.; Niemetz, A.; Nötzli, A.; Ozdemir, A.; Preiner, M.; Reynolds, A.; Sheng, Y.; Tinelli, C.; and Zohar, Y. 2022. cvc5: A Versatile and Industrial-Strength SMT Solver. In Fisman, D.; and Rosu, G., eds., *TACAS 2022*, volume 13243, 415–442.
- Barrett, C.; Fontaine, P.; and Tinelli, C. 2025. The SMT-LIB Standard: Version 2.7. Technical report, Department of Computer Science, The University of Iowa. Available at www.SMT-LIB.org.
- Barrett, C.; Stump, A.; Tinelli, C.; et al. 2010. The smt-lib standard: Version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK)*, volume 13, 14.
- Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2021. Satisfiability Modulo Theories. In *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 1267–1329. IOS Press.
- Becker, N.; Müller, P.; and Summers, A. J. 2019. The axiom profiler: Understanding and debugging smt quantifier instantiations. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 99–116. Springer.
- Berman, S.; McKeown, K.; and Ray, B. 2024. Solving Zebra Puzzles Using Constraint-Guided Multi-Agent Systems. *arXiv preprint arXiv:2407.03956*.
- Beyer, D.; Dangl, M.; and Wendler, P. 2018. A Unifying View on SMT-Based Software Verification. *J. Autom. Reason.*, 60(3): 299–335.
- Bjørner, N.; and Nachmanson, L. 2020. Navigating the universe of Z3 theory solvers. In *Brazilian Symposium on Formal Methods*, 8–24. Springer.
- Bonacina, M. P.; Graham-Lengrand, S.; and Vauthier, C. 2023. QSMA: a new algorithm for quantified satisfiability modulo theory and assignment. In *International Conference on Automated Deduction*, 78–95. Springer.
- Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.
- Cimatti, A.; Griggio, A.; Mover, S.; and Tonetta, S. 2015. HyComp: An SMT-based model checker for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 52–67. Springer.
- Cimatti, A.; Mover, S.; and Tonetta, S. 2013. SMT-based scenario verification for hybrid systems. *Formal Methods in System Design*, 42(1): 46–66.
- Cordeiro, L. C.; Fischer, B.; and Marques-Silva, J. 2009. SMT-Based Bounded Model Checking for Embedded ANSI-C Software. In *ASE 2009, 24th IEEE/ACM International Conference on Automated Software Engineering, Auckland, New Zealand, November 16-20, 2009*, 137–148. IEEE Computer Society.
- De Moura, L.; and Bjørner, N. 2007. Efficient E-matching for SMT solvers. In *International Conference on Automated Deduction*, 183–198. Springer.
- de Moura, L. M.; and Bjørner, N. 2008. Z3: An Efficient SMT Solver. In Ramakrishnan, C. R.; and Rehof, J., eds., *TACAS 2008*, volume 4963, 337–340.
- DeepSeek, A.; Liu, A.; Feng, B.; Xue, B.; Wang, B.; Wu, B.; Lu, C.; Zhao, C.; Deng, C.; Zhang, C.; Ruan, C.; et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.
- Fischer, M. J.; and Rabin, M. O. 1998. Super-exponential complexity of Presburger arithmetic. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, 122–135. Springer.
- Frieder, S.; Pinchetti, L.; Griffiths, R.-R.; Salvatori, T.; Lukasiewicz, T.; Petersen, P.; and Berner, J. 2023. Mathematical capabilities of chatgpt. *Advances in neural information processing systems*, 36: 27699–27744.
- Gavrilenco, N.; de León, H. P.; Furbach, F.; Heljanko, K.; and Meyer, R. 2019. BMC for Weak Memory Models: Relation Analysis for Compact SMT Encodings. In Dillig, I.; and Tasiran, S., eds., *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, 355–365. Springer.
- Ge, C. 2025. SharpSMT: a scalable toolkit for measuring solution spaces of SMT (LA) formulas. *Frontiers of Computer Science*, 19(8): 198336.
- Ge, Y.; and De Moura, L. 2009. Complete instantiation for quantified formulas in satisfiability modulo theories. In *International Conference on Computer Aided Verification*, 306–320. Springer.
- Jia, F.; Dong, Y.; Han, R.; Huang, P.; Liu, M.; Ma, F.; and Zhang, J. 2025. A Complete Algorithm for Optimization Modulo Nonlinear Real Arithmetic. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 11255–11263.
- Kesseli, P.; O’Hearn, P.; and Cabral, R. S. 2025. Logic.py: Bridging the Gap between LLMs and Constraint Solvers. *arXiv preprint arXiv:2502.15776*.
- Kroening, D.; and Strichman, O. 2016. *Decision Procedures - An Algorithmic Point of View, Second Edition*. Texts in Theoretical Computer Science. An EATCS Series. Springer. ISBN 978-3-662-50496-3.
- Lewkowycz, A.; Andreassen, A.; Dohan, D.; Dyer, E.; Michalewski, H.; Ramasesh, V.; Slone, A.; Anil, C.; Schlag,

- I.; Gutman-Solo, T.; et al. 2022. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35: 3843–3857.
- Liu, P.; Yuan, W.; Fu, J.; Jiang, Z.; Hayashi, H.; and Neubig, G. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM computing surveys*, 55(9): 1–35.
- Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2006. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL (T). *Journal of the ACM (JACM)*, 53(6): 937–977.
- Pan, L.; Albalak, A.; Wang, X.; and Wang, W. Y. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*.
- Peleska, J.; Vorobev, E.; and Lapschies, F. 2011. Automated Test Case Generation with SMT-Solving and Abstract Interpretation. In Bobaru, M. G.; Havelund, K.; Holzmann, G. J.; and Joshi, R., eds., *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*, volume 6617 of *Lecture Notes in Computer Science*, 298–312. Springer.
- Reynolds, A.; Barbosa, H.; and Fontaine, P. 2018. Revisiting enumerative instantiation. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 112–131. Springer.
- Reynolds, A.; Barbosa, H.; Nötzli, A.; Barrett, C.; and Tinelli, C. 2019. cvc4sy: Smart and fast term enumeration for syntax-guided synthesis. In *International Conference on Computer Aided Verification*, 74–83. Springer.
- Reynolds, A.; Deters, M.; Kuncak, V.; Tinelli, C.; and Barrett, C. 2015. Counterexample-guided quantifier instantiation for synthesis in SMT. In *International Conference on Computer Aided Verification*, 198–216. Springer.
- Reynolds, A.; King, T.; and Kuncak, V. 2017. Solving quantified linear arithmetic by counterexample-guided instantiation. *Formal Methods in System Design*, 51(3): 500–532.
- Reynolds, A.; and Kuncak, V. 2015. Induction for SMT solvers. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, 80–98. Springer.
- Reynolds, A.; Tinelli, C.; Goel, A.; Krstić, S.; Deters, M.; and Barrett, C. 2013. Quantifier instantiation techniques for finite model finding in SMT. In *International Conference on Automated Deduction*, 377–391. Springer.
- Sahoo, P.; Singh, A. K.; Saha, S.; Jain, V.; Mondal, S.; and Chadha, A. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*.
- Shi, W.; Liu, M.; Zhang, W.; Shi, L.; Jia, F.; Ma, F.; and Zhang, J. 2025. ConstraintLLM: A Neuro-Symbolic Framework for Industrial-Level Constraint Programming. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 16010–16030.
- Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.
- Wang, W.; Liu, K.; Chen, A. R.; Li, G.; Jin, Z.; Huang, G.; and Ma, L. 2024. Python symbolic execution with llm-powered code generation. *arXiv preprint arXiv:2409.09271*.
- Wei, J.; Tay, Y.; Bommasani, R.; Raffel, C.; Zoph, B.; Borgeaud, S.; Yogatama, D.; Bosma, M.; Zhou, D.; Metzler, D.; et al. 2022a. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.
- Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022b. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.
- Zhang, J. 2000. Specification analysis and test data generation by solving Boolean combinations of numeric constraints. In *Proceedings First Asia-Pacific Conference on Quality Software*, 267–274. IEEE.
- Zhang, J. 2008. Constraint solving and symbolic execution. In *Verified Software: Theories, Tools, and Experiments (VSTTE 2005)*, volume 4171 of *Lecture Notes in Computer Science*, 539–544. Springer.
- Zhang, J.; and Wang, X. 2001. A constraint solver and its application to path feasibility analysis. *International Journal of Software Engineering and Knowledge Engineering*, 11(02): 139–156.