Learning Simple Interpolants for Linear Integer Arithmetic

Minchao Wu
The University of Tokyo
minchaow@g.ecc.u-tokyo.ac.jp

Naoki Kobayashi
The University of Tokyo
koba@is.s.u-tokyo.ac.jp

Abstract

Craig interpolation plays a central role in formal verification tasks such as model checking, invariant generation, and abstraction refinement. In the domain of linear integer arithmetic (LIA), interpolants are crucial for deriving inductive invariants that characterize unreachable or safe program states, enabling scalable and precise reasoning about software and hardware correctness. Despite progress in interpolation algorithms, generating concise and interpretable interpolants remains a key challenge. We propose a lightweight learning-based approach to generating simple interpolants for LIA. Our model learns to lazily sample input problems directly and is complementary to existing logical methods. We show that when Z3 is guided by our learned model, the complexity of the interpolants it produces can be reduced by up to 47.3%. For older solvers, the reduction rate can reach up to 69.1%.

1 Introduction

Craig interpolation is a foundational technique in formal verification, instrumental in applications such as abstraction refinement, invariant generation, and compositional reasoning [20, 14, 21, 22, 16, 15, 23, 33, 1]. In the realm of linear integer arithmetic (LIA), interpolants serve as crucial intermediaries, connecting preconditions and postconditions within program verification frameworks. However, generating interpolants in LIA presents significant challenges due to the discrete nature of integers and the complexity of associated arithmetic operations. Traditional methods often produce interpolants with large coefficients and intricate structures, which can hinder the efficiency and scalability of verification tools.

Recent advancements in machine learning have shown promise in complementing formal methods by providing heuristics that guide complex decision-making processes. Despite this progress, the integration of learning-based techniques into the domain of LIA interpolation remains relatively unexplored. Existing interpolation procedures primarily rely on symbolic reasoning and decision procedures, which, while sound and complete, may not always yield the most concise interpolants [1].

In this work, we introduce a novel learning-based framework aimed at simplifying interpolants in LIA. Our approach leverages a lightweight neural architecture that combines a message-passing graph neural network (GNN) [40] with a self-attention-based [34] selector mechanism. The GNN encodes the structural and semantic information of disjunction-free LIA formulas, while the selector network tries to identify minimal subsets of the input formulas that are sufficient for generating valid interpolants. This design enables a single invocation of the satisfiability modulo theories (SMT) solver during the computation, enhancing efficiency and scalability.

To facilitate this learning process, we require a dataset of challenging interpolation problems for both training and evaluation. As no such dataset exists in the literature to our knowledge, we construct one comprising approximately 66,000 interpolation problems in LIA. The dataset is generated through a

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

systematic process that ensures realistic and practical coverage of problem instances. We hope it will serve as a valuable resource for advancing learning-based interpolation methods.

Empirical evaluations demonstrate the effectiveness of our approach. When integrated with state-of-the-art interpolation solvers such as Z3 [9], our model achieves a reduction in interpolant complexity of up to 47.3%. When applied to older solvers, the reduction rate increases to 69.1%, demonstrating the potential of our method to significantly enhance existing tools.

Our contributions can be summarized as follows:

- We propose a novel learning-based approach that guides the generation of simpler interpolants in linear integer arithmetic, complementing traditional logic-based methods. Motivated by the demands of downstream verification workloads, our model architecture emphasizes both fast evaluation and high-quality interpolant generation.
- We introduce a dataset containing approximately 66,000 challenging interpolation problems in linear integer arithmetic. We detail the generation process to ensure reproducibility and coverage.
- We demonstrate that our approach effectively reduces interpolant complexity, achieving significant improvements when integrated with both modern and legacy interpolation solvers.

2 Preliminaries

Craig interpolation is a foundational concept in logic and program verification, widely used in constructing inductive invariants and enabling efficient model checking. Given two logical formulas A and B such that their conjunction $A \wedge B$ is unsatisfiable, a *Craig interpolant* is a formula I satisfying the following properties:

1. $A \Rightarrow I$
2. $I \Rightarrow \neg B$
3. $\text{Vars}(I) \subseteq \text{Vars}(A) \cap \text{Vars}(B)$, where $\text{Vars}(\cdot)$ denotes the set of variables in a formula.

The existence of an interpolant is guaranteed by Craig's interpolation theorem for any two unsatisfiable (or *disjoint*) formulas A and B in LIA. In what follows, we refer to the *complexity* of a formula as the number of logical connectives it contains. Since our domain is integer arithmetic, negation $\neg$ is unnecessary and we count only $\wedge$ and $\vee$.

As an example, consider the following formulas A and B:

$$A = (x = 0 \wedge y = 0) \vee (x = 1 \wedge y = 2) \vee (x = 2 \wedge y = 4)$$
$$B = \neg(x = 100 \Rightarrow y = 200)$$

One can verify that all of

$$I_1 = (x = 0 \wedge y = 0) \vee (x = 1 \wedge y = 2) \vee (x = 2 \wedge y = 4)$$
$$I_2 = x = 100 \Rightarrow y = 200$$
$$I = y = 2x$$

are interpolants between A and B.

2.1 Applications to Program Verification and Why Simple Interpolants Matter

In program verification, Craig interpolants are often used to find an invariant, which certifies that error states are unreachable. Let A be a formula describing an approximation of states reachable from initial states, and B be a formula describing an approximation of states reachable to error states. Then, an interpolant I between A and B serves as a candidate invariant.

The example interpolation problem above may be extracted from the following program.

```
x = 0; y = 0;
while(*) {x++; y+=2}
assert(x=100 => y=200);
```

The formula $A \equiv (x = 0 \wedge y = 0) \vee (x = 1 \wedge y = 2) \vee (x = 2 \wedge y = 4)$ describes the states reachable within two iterations of the loop, and the formula $B \equiv \neg(x = 100 \Rightarrow y = 200)$ describes an error state. To prove that an error state is unreachable, we need to find an (inductive) invariant I such that $A \Rightarrow I$ and $I \Rightarrow \neg B$, i.e., I is an interpolant between A and B (with an additional condition that I is closed under the transition relation).

In practice, the complexity of the interpolant I plays a crucial role as simplicity often leads to generality. A formula I being an interpolant between reachable states and error states is merely a necessary condition for I to be an inductive invariant — I must additionally be closed under the transition relation — and that multiple interpolants can exist between the same pair of formulas. A simpler interpolant is more likely to constitute a true invariant, as it is less prone to overfitting to specific approximations A and B of reachable and error states. In the example above, both I_1 and I_2 are valid interpolants, but it is not difficult to see that only I, the simplest of the three, is an inductive invariant of the program.

This is analogous to machine learning, where a simple model that captures the core pattern will generalize better than an overly complex one that overfits to training data. In the verification context, a complex interpolant can "overfit" particular approximations of reachable and error states, whereas a concise interpolant focuses on the essential relationship and is less prone to depend on such approximations.

The importance of simple interpolants in program verification has been widely recognized. Albarghouthi and McMillan [1] emphasized this point and empirically demonstrated that simpler interpolants can indeed improve the efficiency of verification.

2.2 Lazy Sampling Method

The *lazy sampling method* described in Albarghouthi and McMillan's compositional approach [1] aims to generate simpler interpolants through compositional reasoning. It proceeds as follows:

1. Given an interpolation problem (A, B), the goal is to construct an interpolant I satisfying the above conditions. Two initially empty sets, known as sample sets, are maintained for feasible disjuncts of A and B.
2. The algorithm lazily samples disjuncts from A and B using an SMT solver, and iteratively attempts to find a simple interpolant I separating these samples.
3. If the candidate interpolant fails, additional disjuncts contradicting this interpolant are sampled and incorporated. The method then restarts with these newly updated sample sets, iteratively refining the interpolant.

As an illustrative example, consider the following interpolation problem from [1]:

$$A = (x \leq 1 \wedge y \leq 3) \vee (1 \leq x \leq 2 \wedge y \leq 2) \vee (2 \leq x \leq 3 \wedge y \leq 1)$$
$$B = (x \geq 2 \wedge y \geq 3) \vee (x \geq 3 \wedge 2 \leq y \leq 3)$$

These formulas represent distinct convex regions in two-dimensional space, and their conjunction $A \wedge B$ is unsatisfiable, as these regions do not intersect.

Applying the lazy sampling method, one might initially sample the disjunct $(1 \leq x \leq 2 \wedge y \leq 2)$ from A and $(x \geq 2 \wedge y \geq 3)$ from B. A candidate interpolant between the sets

$$S_A = \{1 \leq x \leq 2 \wedge y \leq 2\} \quad S_B = \{x \geq 2 \wedge y \geq 3\}$$

might be $y \leq 2$. However, since $(x \leq 1 \wedge y \leq 3)$ from A is a *counterexample*[1] (because this interpolant fails the first condition $A \Rightarrow I$), this disjunct is subsequently sampled as well. Now, considering the updated

$$S_A = \{1 \leq x \leq 2 \wedge y \leq 2,\ x \leq 1 \wedge y \leq 3\} \quad S_B = \{x \geq 2 \wedge y \geq 3\}$$

a new valid interpolant $x + y \leq 4$ between them may be found. In this case, the interpolant also successfully separates the original formulas A and B.

[1] In practice, this fact is checked by an SMT solver.

Note that the counterexample-guided selection of new disjuncts does not guarantee simplicity. It is possible to end up with unnecessary disjuncts in the final sets S_A and S_B, especially when multiple disjuncts can serve as counterexamples simultaneously. In such cases, additional heuristics may be required to rank and prioritize the formulas.

3 Models

We now describe our approach to generating simple interpolants, following the lines described in Section 2.2. Since every LIA formula can be converted to its disjunctive normal form, below we always treat an input formula of an interpolation problem as a set (or sequence) of conjunctive formulas. Each formula in this set is referred to as a *disjunct*.

3.1 Motivation

A naive way to improve upon the logical methods in Section 2.2 is to replace the counterexample-guided selection of formulas with learned heuristics. There are however two major drawbacks with this approach. The first is that it would require multiple invocations of both the model and the SMT solver throughout the computation. This is because we are essentially constructing the subsets of the two input formulas from the bottom up. Each time the SMT solver provides feedback on the validity of a candidate interpolant, the model is called to make a prediction. A new candidate is then composed, and the process is repeated. This becomes increasingly expensive as the input formulas grow larger. The second drawback is the lack of a clear definition of ground truth in this setting. Given any timestep t and the selected subsets $(S_A)_t$ and $(S_B)_t$, it is possible that no disjunct from the two input formulas, if added to $(S_A)_t$ or $(S_B)_t$, will immediately yield a valid interpolant between A and B. In such cases, no definitive signal can be provided to properly train the model.

A better way to achieve a similar improvement is to lazily sample the disjuncts in a top-down manner. That is, we want the model to directly construct subsets $S_A \subset A$ and $S_B \subset B$ such that the interpolant between them is also a valid interpolant between A and B. Once these subsets are identified, we only need to run the interpolation solver *once* on them.

This is a challenging task. Mathematically, any interpolant between A and B is also a valid interpolant between any $S_A \subset A$ and $S_B \subset B$. However, it is not generally true that there exist $S_A \subset A$ and $S_B \subset B$ such that *any* interpolant between them is a valid interpolant between A and B.

In practice, lazily sampling a small number of disjuncts from A and B often suffices to find an interpolant for A and B [1]. However, this heavily depends on the underlying mechanics of the interpolation solver. Given the same $S_A \subset A$ and $S_B \subset B$, different solvers may produce different interpolants, and not all of them are necessarily valid interpolants for A and B.

This implies that an ideal model will inevitably need to internalize certain aspects of the target solver's behavior. Modern SMT solvers such as Z3 have complex codebases and decades of carefully tuned optimizations. It would be fascinating if even a portion of such a solver could be approximated by a relatively small model.

We emphasize the importance of a lightweight approach and intentionally avoid relying on large language models (LLMs). This choice is motivated by the fact that both the speed and accuracy of interpolation solving are critical for downstream verification tasks.

To achieve this, we develop a model with two components: an encoder that learns representations of disjunction-free LIA formulas using a message-passing graph neural network, followed by an attention-based selector network that selects appropriate disjuncts from the original input formulas. The high-level computation flow is illustrated in Figure 1.

3.2 Encoding LIA Formulas

Given a LIA formula, we first obtain its graph representation by converting it into the corresponding abstract syntax tree (AST). Each node in the formula graph is represented by a feature vector $\mathbf{x}_i \in \mathbb{R}^d$, where $d = 1 + d_t + 1 + 1$, comprising:

- A discrete node type indicator $t_i \in \{0, 1, 2\}$ representing operators, variables, and constants, respectively

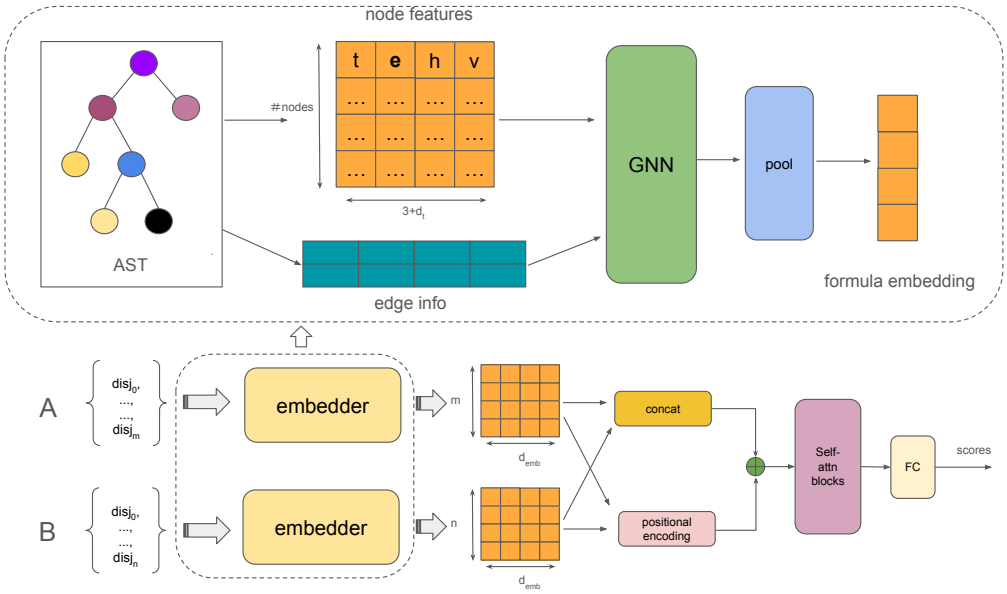

Figure 1: Computation flow. The upper part illustrates the embedding process for a single LIA formula, as described in Sections 3.2 to 3.5. The lower part shows the disjunct selection mechanism, described in Section 3.6.

- A learnable token embedding vector $\mathbf{e}_i \in \mathbb{R}^{d_t}$ capturing semantic information
- A depth value $h_i \in \mathbb{N}$ indicating the node's position in the formula tree
- A raw value $v_i \in \mathbb{Z}$ for constant nodes (zero-padded for non-constants)

A LIA formula is thus represented by a set of node features along with the corresponding edge information, which are then processed by a message-passing graph neural network to compute feature embeddings.

We use a message-passing neural network architecture specifically designed to learn representations of integer arithmetic formulas. The model employs a hierarchical feature embedding scheme followed by multiple rounds of message passing to capture both local arithmetic constraints and the global structure of the formula.

3.3 Feature Embedding

The raw node features are transformed into a latent representation through a multi-component embedding network:

$$\begin{aligned}
\mathbf{z}_i^{type} &= \text{Emb}_{type}(t_i) \in \mathbb{R}^{d_h/4} \\
\mathbf{z}_i^{token} &= \text{MLP}_{token}(\mathbf{e}_i) \in \mathbb{R}^{d_h/2} \\
\mathbf{z}_i^{depth} &= \text{MLP}_{depth}(h_i) \in \mathbb{R}^{d_h/8} \\
\mathbf{z}_i^{value} &= \text{MLP}_{value}(v_i) \in \mathbb{R}^{d_h/8}
\end{aligned} \tag{1}$$

where d_h is the hidden dimension. These embeddings are concatenated and passed through a feature combination network:

$$\mathbf{h}_i^{(0)} = \text{LayerNorm}(\text{MLP}_{comb}([\mathbf{z}_i^{type}; \mathbf{z}_i^{token}; \mathbf{z}_i^{depth}; \mathbf{z}_i^{value}])) \tag{2}$$

The resulting node embeddings are then refined through multiple rounds of message passing: in each round, a message is computed for every edge based on the concatenation of the source and target node embeddings, passed through a multi-layer perceptron (MLP), and then aggregated additively at the target node.

3.4 Message Passing

Our message passing scheme iteratively updates node representations over L layers. At each layer l, the message function $M^{(l)}$ and update function $U^{(l)}$ are defined as:

$$\begin{aligned} \mathbf{m}_{i,j}^{(l)} &= M^{(l)}(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}) = \text{MLP}_{msg}^{(l)}([\mathbf{h}_i^{(l)}; \mathbf{h}_j^{(l)}]) \\ \mathbf{a}_i^{(l)} &= \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{i,j}^{(l)} \\ \mathbf{h}_i^{(l+1)} &= U^{(l)}(\mathbf{h}_i^{(l)}, \mathbf{a}_i^{(l)}) = \text{LayerNorm}(\mathbf{h}_i^{(l)} + \text{MLP}_{upd}^{(l)}([\mathbf{h}_i^{(l)}; \mathbf{a}_i^{(l)}])) \end{aligned} \tag{3}$$

where $\mathcal{N}(i)$ denotes the neighbors of node i in the formula graph. Residual connections and layer normalization are applied at each layer to stabilize training.

3.5 Formula-Level Representation

After the message-passing phase, node embeddings are aggregated into a single graph-level vector. Specifically, we compute the final latent representation for the formula using the node embeddings in its AST. The final graph representation is obtained by combining global mean and sum pooling operations:

$$\mathbf{g} = \frac{1}{2}(\text{meanpool}(\{\mathbf{h}_i^{(L)}\}_{i=1}^{N}) + \text{sumpool}(\{\mathbf{h}_i^{(L)}\}_{i=1}^{N})) \tag{4}$$

followed by a final transformation:

$$\mathbf{y} = \text{MLP}_{out}(\mathbf{g}) \tag{5}$$

This architecture explicitly captures both local arithmetic relationships through message passing and global formula structure through hierarchical embedding and pooling mechanisms. The resulting formula-level embedding is then ready for downstream tasks, including—but not limited to—our lazy sampling method for interpolation solving. The process of encoding a (disjunction-free) LIA formula and computing its latent representation is illustrated in the upper part of Figure 1.

3.6 Selector Network

By the nature of interpolation solving, the two input formulas must be interdependent when deciding whether a particular disjunct is necessary. Each disjunct in A should be aware of every disjunct in B, and vice versa, in order to compute a valid interpolant. Moreover, since we are particularly interested in identifying redundancy within the input formulas, each formula in A (respectively, in B) should also attend to other formulas within A (or B). For this reason, we employ self-attention rather than cross-attention in the selector network.

Given the two sequences of embeddings for formulas in A and B computed by the GNN, we first concatenate them to obtain a matrix of formula embeddings $\mathbf{X} \in \mathbb{R}^{N \times d}$, where N is the total number of formulas in $A \cup B$, and d is the embedding dimension.

To preserve set membership information, each embedding in $\mathbf{X}$ is augmented with a learnable positional encoding. Specifically, a trainable vector $\mathbf{e}_A \in \mathbb{R}^d$ is added to all embeddings corresponding to formulas in set A, and a separate vector $\mathbf{e}_B \in \mathbb{R}^d$ is added to those from set B. This produces set-aware input embeddings:

$$\tilde{\mathbf{X}} = \mathbf{X} + \mathbf{E}, \quad \text{where} \quad \mathbf{E}_{i,j,:} = \begin{cases} \mathbf{e}_A & \text{if set\_ids}_{i,j} = 0 \quad \text{(indicating set } A\text{)} \\ \mathbf{e}_B & \text{if set\_ids}_{i,j} = 1 \quad \text{(indicating set } B\text{)} \end{cases}$$

This enriched input $\tilde{\mathbf{X}}$ is passed through a series of self-attention and feedforward blocks, enabling each disjunct to attend to all others in the combined input. In particular, self-attention computes

attention weights α_{ij} that quantify the influence of formula j on formula i, allowing the model to reason about interdependencies and relative importance across $A \cup B$.This is especially valuable when certain formulas are redundant or contextually less informative. The combination of set-aware encoding and self-attention allows the selector to make globally informed decisions about which disjuncts are most relevant in context.

The resulting tensor is then passed through a feedforward scoring head, which projects each formula embedding to a scalar score used to rank disjuncts from the input formulas.

4 Experiments

4.1 Dataset

To demonstrate the effectiveness of our approach, we require a dataset of challenging interpolation problems for both training and evaluation. As no such dataset exists in the literature to our knowledge, we construct one ourselves and describe the generation process in detail.

We begin by creating a bank of disjunction-free LIA formulas that serve as the "atomic" components of our dataset. This is done by running an existing software verification tool on real downstream tasks (e.g., functional program verification), where interpolation solving is used as a subroutine.[2] We then extract the execution traces of the interpolation solver and collect all disjunction-free LIA formulas and subformulas that occur in these traces.

We then scan the entire formula bank and construct a dictionary D, where each formula serves as a key, and its value is a list of formulas that are all disjoint from the key. Given a sequence of disjunction-free formulas $\varphi_1, \ldots, \varphi_n$, it is not difficult to verify that

$$A = \bigvee_{i=1}^{n} \varphi_i \quad B = \bigvee \bigcap_{i=1}^{n} D[\varphi_i]$$

forms a valid interpolation problem, provided that $\bigcap_{i=1}^{n} D[\varphi_i]$ is nonempty.

To ensure variety in the dataset, we must randomly access different combinations of keys in D. However, when D is large, enumerating all combinations and then sampling from them is computationally infeasible. Instead, we randomly sample an index c and compute the c-th combination directly. We omit the implementation details of this procedure here.

The generation of the dataset is controlled by two parameters, c_1 and c_2, denoting the maximum length of A and B, respectively. Problems where either A or B is a singleton are excluded to ensure sufficient complexity. We denote the resulting dataset as **Interp**(c_1, c_2). In the experiments that follow, we primarily use **Interp**$(4, 4)$.

Although we use relatively small values of c_1 and c_2 due to computational constraints, **Interp**$(4, 4)$ already contains challenging problems for state-of-the-art interpolation solvers. This is partly because we place no restriction on the internal complexity of the disjuncts in the input formulas—only on their count. Table 1 summarizes statistics for **Interp**$(4, 4)$. The full dataset is split into training and testing sets using an 80%–20% ratio.

Table 1: Dataset Statistics

Statistic	**Value**	**Description**
Number of problems	65,981	Total number of data points
Maximum complexity	31	Highest observed formula complexity
Average complexity	9	Average over all input formulas
Constant range	0–112	Absolute value range of constants
Vocabulary size	355	Variable names and operators only

[2]For this paper, we use a model checker called MoCHi [19] on its test benchmarks.

4.2 Ground Truths and Hyperparameters

To generate ground truths, we compute interpolants between all pairs (S_A, S_B) such that $S_A \in \mathcal{P}(A)$ and $S_B \in \mathcal{P}(B)$, using a target solver (e.g., Z3). A pair (S_A, S_B) is considered a candidate if the interpolant between S_A and S_B is also valid for the original formulas A and B. Among all valid candidates, the shortest pair (in terms of set size) is selected as the ground truth.

In practice, we observe that many problems admit multiple valid ground truths. Randomly selecting one can introduce noise and hinder model training. To provide more deterministic supervision, we apply two additional criteria to select among the candidates:

- (S_A, S_B) has the fewest total tokens among all valid candidates.
- The sum of the absolute values of constants occurring in (S_A, S_B) is minimal.

In the experiments below, we fix the token embedding dimension to 16 and the formula embedding dimension to 32. The hidden feature size of both the GNN encoder and the selector network is set to 128. We use 4 heads in the multi-head attention mechanism. The parameters of the graph encoder, selector network, and token embedding layer are jointly trained with a batch size of 256 using the Adam optimizer [18] for 300 epochs.

The learning rate is initially set to 1e-3 and is reduced by a factor of 0.5 every 50 epochs. Training is performed using PyTorch's Metal backend [27, 2] and takes approximately 1.5 days on a single MacBook Pro with the M1 MAX chip.

4.3 Improving Interpolants Found by Z3

We begin by evaluating the improvements our model brings to Z3 in terms of interpolant complexity. Although Z3 is already highly optimized and capable of producing concise interpolants for many problems in our dataset, it does not always yield the simplest solution. To better illustrate the benefit of our approach, we focus on the subset of test problems that are challenging for Z3. We define a problem as *challenging* if there exists an interpolant whose complexity is at least two units lower than that of the one produced by Z3 [3]. This results in a subset of 532 such problems from the test set. Table 2 presents a comparison between the original Z3 and Z3 guided by our model.

Table 2: Performance Gain. Numbers are averaged over valid solutions.

Methods	**Valid**	**Complexity**	**Length**	**Z3 Time (s)**
Original Z3	100%	5.38	186.7	0.012
Z3 guided by model	72.9%	**2.83**	108.6	0.009

As shown in Table 2, the complexity of the interpolants produced by Z3 is reduced by 47.3% when guided by our model, a result further supported by the corresponding reduction in plain-text length. The execution time of Z3 is also reduced by 30% under model guidance.

The 72.9% success rate may appear suboptimal, but this is not a significant limitation in practice—one can simply fall back to the original solver in cases of failure. While this fallback mechanism introduces a small overhead due to model invocation (averaging 20ms in our experiments), the overall impact on performance remains modest.

Importantly, the improvement in interpolant quality can have substantial downstream benefits: a better interpolant can often enable verification tasks to succeed that would otherwise time out completely.

4.4 Reviving an Older Solver

As discussed in Section 3.1, the effectiveness of lazy sampling is solver-dependent. Given an interpolation problem (A, B), the sampled subsets (S_A, S_B) that work for Z3 may not generalize to other solvers. In this section, we evaluate how well the trained model transfers to a different solver. Specifically, we experiment with an interpolation solver called CSIsat [4], an older tool whose interpolants on our test problems have an average complexity of 11.5. We assess the performance of

[3] This definition is independent of the model's predictions.

both Z3 and CSIsat when guided by models trained using their respective ground truths, as well as when guided by "exotic" ground truths—i.e., ground truths generated for the other solver.

Table 3: Performance of Solvers Guided by Different Models on All Test Problems

Model	**trained with Z3 ground truths**			**trained with CSIsat ground truths**		
	Valid	Complexity	Solver Time (s)	Valid	Complexity	Solver Time (s)
Z3	74.2%	2.84	0.009	52.7%	2.23	0.011
CSIsat	59.2%	5.13	0.007	73.7%	3.57	0.007

As shown in Table 3, we observe a roughly symmetric pattern in success rates. Both solvers manage to achieve around 55% validity when guided by a model trained on exotic ground truths. This suggests that the model captures and transfers some of the underlying interpolation heuristics shared between Z3 and CSIsat.

However, each solver performs significantly better when guided by the model trained specifically for it. This indicates that the dedicated models have acquired solver-specific knowledge during training. Interestingly, when Z3 is guided by the model trained on CSIsat ground truths, the resulting interpolants appear slightly simpler (in terms of complexity) than those generated using the dedicated Z3 model. We do not overinterpret this effect, as it is likely due to the complexity being averaged over a smaller subset of valid solutions.

Lastly, we note that the complexity of the interpolants produced by CSIsat is significantly reduced when guided by either model—by up to 69.1%. This demonstrates that we can elevate CSIsat's performance to a modern standard without modifying its legacy codebase.

Such a capability is particularly valuable in scenarios where alternative interpolants are sought for difficult problems in downstream verification tasks. In this role, the revived CSIsat can now serve as a practical and efficient tool.

5 Related Work

Craig Interpolation in Integer Arithmetic Craig interpolation has been extensively studied in the context of formal verification, particularly for generating intermediate assertions in safety analysis and model checking. Cimatti et al. [6] provides a comprehensive overview of efficient interpolation procedures for satisfiability modulo theories (SMT). Compositional methods [1] have proven especially effective for producing high-quality interpolants. Although learning-based methods for LIA interpolation remains relatively unexplored, traditional statistical machine learning techniques, such as Support Vector Machines (SVMs) [8] have been applied to the problem of finding half-space interpolants [31]. However, such approaches are typically incomplete when the interpolation problem is not linearly separable.

Learning-Based Methods in Theorem Proving The integration of machine learning into automated theorem proving has gained significant traction in recent years. Deep learning has been widely explored in the context of traditional automated theorem proving (ATP), including SAT solving [30, 29, 36, 5, 12]. In contrast to SAT solving, interpolation in LIA involves a richer logical language and requires more sophisticated encoding. While a SAT solution consists of either a satisfying truth assignment or a proof of unsatisfiability, an interpolant is an arbitrary formula within the underlying theory. This additional expressiveness presents unique challenges for applying learning-based techniques.

In the field of interactive theorem proving (ITP) with proof assistants such as Lean [10], Isabelle [26] and Coq [7], large language models (LLMs) have been extensively used for tasks such as autoformalization [37, 25, 39, 28], proof sketching [17, 11, 32], tactic generation [13, 17, 35] and premise selection [24, 38]. While LLMs are powerful at suggesting formulas, they tend to be slow and imprecise when it comes to symbolic reasoning, making them less suitable for our task, which requires efficient and accurate interpolation.

6 Limitations

Characterization of Complexity. In this paper, we primarily characterize the complexity of a linear arithmetic formula by the number of its logical connectives. This metric captures one of the most intuitive and relevant aspects of interpolant simplicity. However, complexity is inherently multi-dimensional. For instance, the magnitude of constants appearing in an interpolant also plays an important role, as large constants can introduce difficulties in downstream verification tasks. Similarly, the number of variables provides another useful perspective: fewer connectives suggest less redundant "regional" information, while fewer variables imply less redundant "dimensional" information. Depending on the downstream application, an interpolant with more connectives but fewer variables may in fact be preferable.

We have incorporated some of this intuition into the design of the deterministic selection criteria described in Section 4.2. Nonetheless, future work could explore dedicated learning-based heuristics to better capture and optimize for these other dimensions of interpolant complexity.

Applications and Separated Learning. Although our model-guided interpolation solvers can produce significantly simpler interpolants, the benefits do not always translate directly to downstream applications. Many verification tools invoke interpolation as a subroutine and may pre-process or decompose complex problems into simpler ones, resulting in trivial interpolation queries. In such cases, our approach offers limited advantage over standard interpolation solvers.

Moreover, interpolants of the same syntactic complexity can yield vastly different outcomes in downstream tasks, depending on the nature of the application and the behavior of the overarching ("master") solver. If the objective is to produce interpolants that are more effective for a specific downstream task, then the training process must incorporate feedback from that task. This would allow the model to internalize the task-specific heuristics and interaction patterns of the broader verification pipeline.

7 Conclusion

In this work, we introduced a lightweight learning-based framework for simplifying Craig interpolants in linear integer arithmetic (LIA). Our approach combines a message-passing graph neural network (GNN) with a self-attention-based selector network to directly identify minimal subsets of input formulas sufficient to produce valid interpolants. This design complements existing logical techniques and enables improvements over both modern and legacy interpolation solvers—without modifying their underlying implementations.

Empirical evaluations demonstrate the effectiveness of our approach. When integrated with Z3, a state-of-the-art SMT solver, our model reduces interpolant complexity by up to 47.3%. When applied to older solvers, the reduction increases to 69.1%, highlighting the method's potential to substantially enhance existing tools.

Because our method does not rely on linearity, a promising direction for future work is its extension to richer domains such as non-linear integer arithmetic or array theories. Additionally, integrating our framework into downstream tasks—such as invariant synthesis or termination analysis—may further demonstrate its generality and practical value.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number JP20H05703. We thank the anonymous reviewers for their insightful comments and suggestions. We thank Ryosuke Sato for adapting MoCHi for this project.

References

[1] Aws Albarghouthi and Kenneth L. McMillan. Beautiful interpolants. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 313–329, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[2] Apple. Accelerated pytorch training on mac. `https://developer.apple.com/metal/pytorch/`, 2025. Accessed: 2025-05-15.

[3] 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. cvc5: A versatile and industrial-strength smt solver. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 415–442, Cham, 2022. Springer International Publishing.

[4] Dirk Beyer, Damien Zufferey, and Rupak Majumdar. Csisat: Interpolation for la+euf. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification*, pages 304–308, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[5] Chris Cameron, Rex Chen, Jason Hartford, and Kevin Leyton-Brown. Predicting propositional satisfiability via end-to-end learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3324–3331, Apr. 2020.

[6] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Efficient generation of craig interpolants in satisfiability modulo theories, 2009.

[7] Coq Development Team. *The Coq proof assistant reference manual*, 2004. Version 8.0.

[8] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[9] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[10] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, pages 378–388, Cham, 2015. Springer International Publishing.

[11] Emily First, Markus N Rabe, Talia Ringer, and Yuriy Brun. Baldur: Whole-proof generation and repair with large language models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1229–1241, 2023.

[12] Mohamed Ghanem, Frederik Schmitt, Julian Siber, and Bernd Finkbeiner. Learning better representations from less data for propositional satisfiability. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[13] Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W Ayers, and Stanislas Polu. Proof artifact co-training for theorem proving with language models. *arXiv preprint arXiv:2102.06203*, 2021.

[14] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In Neil D. Jones and Xavier Leroy, editors, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, pages 232–244. ACM, 2004.

[15] Ranjit Jhala, Rupak Majumdar, and Ru-Gang Xu. State of the union: Type inference via craig interpolation. In Orna Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4424 of *Lecture Notes in Computer Science*, pages 553–567. Springer, 2007.

[16] Ranjit Jhala and Kenneth L. McMillan. Interpolant-based transition relation approximation. *Log. Methods Comput. Sci.*, 3(4), 2007.

[17] Albert Q Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *arXiv preprint arXiv:2210.12283*, 2022.

[18] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Predicate abstraction and cegar for higher-order model checking. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, page 222–233, New York, NY, USA, 2011. Association for Computing Machinery.

[20] Kenneth L. McMillan. Interpolation and sat-based model checking. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003.

[21] Kenneth L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.

[22] Kenneth L. McMillan. Lazy abstraction with interpolants. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2006.

[23] Kenneth L. McMillan. Quantified invariant generation using an interpolating saturation prover. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2008.

[24] Maciej Mikuła, Szymon Tworkowski, Szymon Antoniak, Bartosz Piotrowski, Albert Q. Jiang, Jin Peng Zhou, Christian Szegedy, Łukasz Kuciński, Piotr Miłoś, and Yuhuai Wu. Magnushammer: A transformer-based approach to premise selection. In *The Twelfth International Conference on Learning Representations*, 2024.

[25] Logan Murphy, Kaiyu Yang, Jialiang Sun, Zhaoyu Li, Anima Anandkumar, and Xujie Si. Autoformalizing euclidean geometry, 2024.

[26] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002.

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[28] Auguste Poiroux, Gail Weiss, Viktor Kunčak, and Antoine Bosselut. Improving autoformalization using type checking, 2025.

[29] Daniel Selsam and Nikolaj Bjørner. Guiding high-performance sat solvers with unsat-core predictions. In Mikoláš Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing – SAT 2019*, pages 336–353, Cham, 2019. Springer International Publishing.

[30] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

[31] Rahul Sharma, Aditya V. Nori, and Alex Aiken. Interpolants as classifiers. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification*, pages 71–87, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[32] Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

[33] Hiroshi Unno and Naoki Kobayashi. Dependent type inference with interpolants. In António Porto and Francisco Javier López-Fraguas, editors, *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, pages 277–288. ACM, 2009.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[35] Haiming Wang, Huajian Xin, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, et al. Lego-prover: Neural theorem proving with growing libraries. *arXiv preprint arXiv:2310.00656*, 2023.

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

[37] Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. *Advances in Neural Information Processing Systems*, 35:32353–32368, 2022.

[38] Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36:21573–21612, 2023.

[39] Lan Zhang, Xin Quan, and Andre Freitas. Consistent autoformalization for constructing mathematical libraries. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4020–4033, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[40] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We base our claims on the experimental results. As for the dataset, we conducted an extensive search on the internet but were unable to find any that are suitable for training and evaluating learning-based approaches to interpolation solving.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Please see Section 6.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We described the architecture of our models in Section 3. We detailed all our experimental setups in the Section 4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The code and datasets will be available at `https://github.com/hopv/scilia`. The dataset generation process is fully documented in Section 4.1. Others should also be able to generate their own dataset if interested.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We detailed all our experimental setups in the Section 4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Errors bars are not reported due to computational budget.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The resources needed are included in Section 4.2.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: No harms were caused by the research process. We do not expect harmful consequences from the integer arithmetic problems considered in this work.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: As discussed throughout the paper, interpolation solving primarily supports downstream verification tasks and does not have direct societal impact.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: For the interpolation solvers we used in the experiments, they are all credited and all the licenses are respected.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: We will make new assets publically available after the double-blind review ended.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Variables and Constants in Interpolants

In addition to the number of logical connectives in an interpolant, we also report the number of distinct variables and the largest constant appearing in each interpolant.

Table 4: Variables and constants. Numbers are averaged over all valid solutions across all test problems.

Model	**trained with Z3 GT**		**trained with CSIsat GT**		**Original**	
	Variables	Constants	Variables	Constants	Variables	Constants
Z3	2.45	5.97	2.39	6.39	2.45	4.24
CSIsat	2.50	16386579	2.52	460537	2.60	7476237725

The number of variables in a solution does not change significantly when either solver is guided by the model. However, the largest constants in CSIsat-generated solutions drop by several orders of magnitude when the solver is guided by the model. This may be attributed to CSIsat lacking effective heuristics for optimizing over integer domains.

B Example Problems from Dataset

Below we include three example problems from **Interp**(4,4). The included "Z3-solutions" are solutions generated by the original Z3.

The "simp-solutions" are the solutions found by Z3 when guided by the model trained specifically for Z3. These are not part of the original **Interp**$(4, 4)$ dataset, but we include them here for reference.

```
{
  'input1': [
    '1 + v_sep_x2 <= v_sep_x3',
    '(v_sep_x3 <= v_sep_x1 and (6 == v_sep_x0 and 1 + v_sep_x0 <=
       v_sep_x3))',
    '(95 <= v_sep_x1 and v_sep_x1 <= 100)',
    '(1 <= v_sep_x0 and 1 + v_sep_x0 <= v_sep_x1)'
  ],

  'input2': [
    '(1 == v_sep_x1 and (0 <= v_sep_x3 and (v_sep_x1 <= v_sep_x2 and 1
        + v_sep_x3 <= v_sep_x2)))',
    '(0 == v_sep_x1 and (v_sep_x2 == v_sep_x3 and 1 <= v_sep_x3))',
    '(1 + v_sep_x3 <= 0 and (1 == v_sep_x1 and (v_sep_x1 == v_sep_x2
       and 1 <= v_sep_x2)))'
  ],

  'Z3-solution': '( not (v_sep_x1  <=  1)  ||  not (v_sep_x3 + -1*
     v_sep_x2  <=  -1))  && ( not (v_sep_x1  <=  0)  ||  not (
     v_sep_x3 + -1*v_sep_x2  <=  0))',

  'simp-solution': ' not (v_sep_x1  <=  1)  ||  not (v_sep_x2 + -1*
     v_sep_x3  >=  0)',

  'time': 0.012347221374511719
}


 {
   'input1': [
     '1 + v_sep_x2 <= v_sep_x3',
     '(92 <= v_sep_x1 and v_sep_x1 <= 97)',
     '(2 + v_sep_x2 <= v_sep_var961 and (3 + v_sep_x2 <= v_sep_var961
        and (2 == v_sep_var961 and (v_sep_var961 <= v_sep_var962 and
        v_sep_var961 <= v_sep_var973))))',
```

```
    '(2 + v_sep_x2 <= v_sep_var702 and (4 + v_sep_x2 <= v_sep_var702
       and (3 == v_sep_var702 and (v_sep_var702 <= v_sep_var703 and
       v_sep_var702 <= v_sep_var711))))'
  ],

  'input2': [
    '(0 == v_sep_x3 and (0 == 1 + v_sep_var3004 and (1 ==
       v_sep_var3005 and (2 == v_sep_x1 and 2 <= v_sep_x2))))',
    '(1 == v_sep_x1 and (0 <= v_sep_x3 and (v_sep_x1 <= v_sep_x2 and 1
       + v_sep_x3 <= v_sep_x2)))',
    '(1 + v_sep_x3 <= 0 and (1 == v_sep_x1 and (v_sep_x1 == v_sep_x2
       and 1 <= v_sep_x2)))',
    '(0 == v_sep_x1 and (v_sep_x2 == v_sep_x3 and 1 <= v_sep_x3))'
  ],

  'Z3-solution': '( not (v_sep_x1 <= 1) || not (v_sep_x3 >= 0)
     || not (v_sep_x2 + -1*v_sep_x3 >= 0)) && ( not (v_sep_x1
    <= 1) || not (v_sep_x3 <= 1) || not (v_sep_x2 >= 1))
     && ( not (v_sep_x1 <= 2) || not (v_sep_x3 <= 0) ||
    not (v_sep_x2 >= 2))',

  'simp-solution': '( not (v_sep_x1 <= 2) || not (v_sep_x2 >=
    0) || not (v_sep_x3 <= 0)) && ( not (v_sep_x1 <= 1) ||
     not (v_sep_x3 >= 0) || not (v_sep_x3 + -1*v_sep_x2 <= 0))
    ',

  'time': 0.014998197555541992
 }


{
  'input1': [
    'v_sep_x3 == 1 + v_sep_x2',
    '(96 <= v_sep_x1 and v_sep_x1 <= 100)',
    '(92 <= v_sep_x1 and v_sep_var122 <= 100)',
    '(1 + v_sep_x2 <= 0 and (1 + v_sep_x3 <= 0 and v_sep_x3 <= 1))'
  ],

  'input2': [
    '(0 == v_sep_x3 and (0 == 1 + v_sep_var3004 and (1 ==
       v_sep_var3005 and (2 == v_sep_x1 and 2 <= v_sep_x2))))',
    '(1 == v_sep_x1 and (0 <= v_sep_x3 and (v_sep_x1 <= v_sep_x2 and 1
       + v_sep_x3 <= v_sep_x2)))',
    '(1 + v_sep_x3 <= 0 and (1 == v_sep_x1 and (v_sep_x1 == v_sep_x2
       and 1 <= v_sep_x2)))',
    '(0 == v_sep_x1 and (v_sep_x2 == v_sep_x3 and 1 <= v_sep_x3))'
  ],

  'Z3-solution': '( not (v_sep_x1 <= 2) || not (v_sep_x3 <= 0)
     || not (v_sep_x2 >= 2)) && ( not (v_sep_x1 <= 1) ||
    not (v_sep_x3 <= 1) || not (v_sep_x2 >= 1)) && ( not (
    v_sep_x1 <= 1) || not (v_sep_x3 >= 0) || not (v_sep_x3
     + -1*v_sep_x2 <= 0))',

  'simp-solution': '( not (v_sep_x1 <= 1) || not (v_sep_x3 >= 0)
     || not (v_sep_x2 + -1*v_sep_x3 >= 0)) && ( not (v_sep_x1
    <= 2) || not (v_sep_x3 <= 0) || not (v_sep_x2 >= 1))'
    ,

  'time': 0.014645099639892578
}
```

C Experiments with CVC5

In addition to Z3 and CSIsat, we also report a limited evaluation with CVC5 [3]. Technically, while CVC5 supports interpolation, it struggles with our benchmark suite. In our experiments, CVC5 failed to solve 85% of the problems within the timeout of 5 seconds. For the remaining 15%, the problems were so simple that the original interpolants already had complexity 1 and could not be further simplified.

Nonetheless, we observe a significant reduction (-28% with a conservative estimation) in solver time which is comparable to the 30% reduction observed for Z3 in Table 2. Table 5 shows the statistics on guiding CVC5 using the model trained on Z3's ground truths.

Table 5: Solver performance (mean $\pm$ 95% CI) for solver time.

Method	**Valid**	**Complexity**	**Solver time (s)**
Original CVC5	15.3%	1	0.301 ± 0.116
Guided CVC5	7.3%	1	0.107 ± 0.031