Learning Unified Graph and Language Representations for SMT Algorithm Selection

Zhengyang Lu
University of Waterloo, Canada

Paul Sarnighausen-Cahn
University of Göttingen and CIDAS, Göttingen, Germany

Jiahao Chen
Georgia Institute of Technology, Atlanta, GA, USA

Arie Gurfinkel
University of Waterloo, Canada

Florin Manea
University of Göttingen and CIDAS, Göttingen, Germany

Vijay Ganesh
Georgia Institute of Technology, Atlanta, GA, USA

Abstract

Algorithm selection is important in satisfiability and constraint solving, since no single solver performs best across all instances. Traditional learning-based approaches represent problem instances using expert-designed features to predict solver performance, while recent work explores graph representations derived from ASTs. However, most existing approaches overlook high-level contextual information, such as the application domain or the benchmark origin. In practice, such cues often help practitioners choose an appropriate solver.

We present SMT-Select, a multimodal framework for SMT algorithm selection. It learns graph representations from formula ASTs and textual representations from natural-language context descriptions. These representations are then combined to guide solver selection. Evaluated across nine SMT logics, SMT-Select consistently outperforms existing selectors and SMT-COMP winning solvers. Across all evaluated logics, it closes at least 30% of the performance gap between the competition winner and the virtual best solver (VBS), and nearly matches the VBS in two logics.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Logic and verification; Computing methodologies → Machine learning approaches

Keywords and phrases satisfiability modulo theories, algorithm selection, multimodal learning, neuro-symbolic AI

Digital Object Identifier 10.4230/LIPIcs.CP.2026.41

Related Version A preliminary version of this work was presented at FoIKS 2026 as a poster paper: *Preliminary Version*: `https://doi.org/10.1007/978-3-032-21540-6_23` [32]

Supplementary Material *Software (Source Code)*: `https://github.com/JohnLyu2/smt-select` archived at `swh:1:dir:ba33c8a1704f6413c4636b8713822b5396d65c15`

Funding Florin Manea and Paul Sarnighausen-Cahn are supported by the German Research Foundation (DFG) through the Heisenberg project 466789228 and research project 562495345.
Zhengyang Lu: NSERC Canada Graduate Scholarship – Doctoral (CGS-D)

Acknowledgements We thank Po-Chun Chien and Nian-Ze Lee for their help in checking the correctness of our enriched natural language descriptions for the SV-COMP benchmarks.

1 Introduction

Satisfiability Modulo Theories (SMT) solvers decide the satisfiability of first-order logic formulas [12], and they serve as core engines for various tools in software engineering [10, 9], formal verification [38, 17], and planning [6, 11]. Due to the inherent difficulty and diversity of SMT applications, no single algorithm performs best across all SMT instances; instead, different solvers exhibit complementary strengths across different problem families and application fields. This motivates algorithm selection [40], which seeks to exploit solver complementarity by selecting the most suitable solver for each problem instance. Machine-learning (ML)-based methods have emerged as an effective data-driven approach for this problem: SATZILLA [59] pioneered ML-based algorithm selection in the SAT domain, and MACHSMT [46] extended the idea to SMT solving.

As in many ML applications, the effectiveness of algorithm selection critically depends on how problem instances are represented. Many algorithm selectors rely on handcrafted, domain-specific features intended to capture instance properties that affect solver performance. In the SMT domain, MACHSMT defined a set of syntactic features, primarily based on counts of theory symbols, to characterize problem instances. Subsequent systems [37, 33, 34, 26] adopted similar features to guide solver or heuristic selection.

Beyond handcrafted features, there has been a growing interest in leveraging ML advances to automate feature design. Given the inherent abstract syntax tree (AST) structure of SMT formulas, graph-based representations provide a natural foundation for such approaches. Hůla et al. [20] were the first to explore the use of graph neural networks (GNNs), specifically Graph Convolutional Networks (GCNs) [23], for SMT algorithm selection. Subsequently, SIBYL [29] used a more advanced GNN architecture, Graph Attention Networks (GATs) [53], and evaluated the approach across a broader range of SMT application scenarios.

More recently, rapid advances in transformer-based models for text representation have opened up promising opportunities for learning expressive features directly from textual inputs [52]. In this direction, Pellegrino et al. [36] proposed a transformer-based approach to feature learning for algorithm selection in combinatorial optimization. Their method employs a BERT encoder [14] to embed optimization problems expressed in the ESSENCE [15] language into fixed-dimensional representations for algorithm selection tasks.

Inspired by this line of work, we investigate whether semantic natural-language descriptions can enhance SMT algorithm selection. Existing approaches such as MACHSMT and SIBYL primarily rely on syntactic representations of formulas. However, practitioners often consider not only the syntactic structure of a formula but also high-level semantic context when choosing solvers. For example, they may take into account the application domain, such as software verification or cryptographic analysis, and draw on prior experience regarding which solvers are designed for such applications or particularly effective for such tasks. This motivates us to explore whether incorporating contextual descriptions alongside structural features can provide complementary signals and improve selection performance.

To this end, we present SMT-SELECT, a multimodal learning-based framework for SMT algorithm selection that integrates structural and semantic representations of SMT formulas. In SMT-SELECT, the formula AST is encoded by a GNN to obtain a structural representation. We additionally incorporate high-level natural-language descriptions, such as those provided in SMT-LIB headers, which capture information about the formula's semantics, application domain, or source, and are typically available to solver users. These descriptions are embedded using a transformer-based encoder. The resulting graph and text representations are then

combined to guide solver selection. Furthermore, SMT-Select provides multiple variants with different representation and learning choices, enabling systematic trade-offs across data availability and computational resources.

Our Contributions

Our main contributions are as follows:

- **SMT-Select, the first multimodal framework for SMT algorithm selection.** We present SMT-Select, a neural architecture that learns SMT instance representations from both the AST structure and natural-language descriptions. The combined representation enables effective solver selection and achieves state-of-the-art performance across diverse SMT logics.
- **A systematic study of representation and architecture choices.** We investigate different feature representations and algorithm selection architectures, leading to multiple SMT-Select variants tailored to different data-availability and computational-resource scenarios. These variants range from lightweight syntactic-feature-based models to full multimodal graph-and-text approaches, all built on an effective cost-sensitive pairwise selection mechanism.
- **Thorough empirical evaluation on SMT-COMP benchmarks.** We evaluate SMT-Select on SMT-COMP benchmarks across nine logics where complementary solvers exist. All SMT-Select variants consistently outperform existing selectors Sibyl and MachSMT, as well as competition-winning solvers. Notably, even the lightweight variant based only on simple syntactic features achieves strong performance, benefiting from the cost-sensitive pairwise selection framework. Building on this foundation, the full multimodal SMT-Select that combines graph and text representations achieves the best performance overall and substantially narrows the gap to the virtual best solver (VBS).

2 Preliminaries

2.1 SMT and Algorithm Selection

Satisfiability Modulo Theories (SMT) solvers determine the satisfiability of first-order logic formulas under background theories [25]. SMT-LIB [2] is an international initiative that supports the development and evaluation of SMT solvers by providing a standardized input language and maintaining a large, curated benchmark library. Many SMT-LIB benchmarks also include *natural-language descriptions* in their headers, which typically summarize the problem origin, application domain, or encoding method. Although these descriptions are not part of the formal logical constraints, they provide contextual information associated with each benchmark instance.

SMT-COMP [3] is an annual competition in which state-of-the-art SMT solvers compete on SMT-LIB benchmarks. For each logic, a winner solver is determined based on the number of benchmarks solved within the specified time limit. However, it is often observed that the winner solver is outperformed by other solvers on certain problem families. This complementarity motivates the problem of algorithm selection, which aims to predict the best solver for each instance.

We briefly introduce the standard formulation of machine learning (ML)-based algorithm selection [24, 22]. Let $\mathcal{A}$ be a portfolio of off-the-shelf SMT solvers. For an SMT instance x and a solver $\alpha \in \mathcal{A}$, we write $u_\alpha(x)$ as the performance of α on x (e.g., runtime). We assume a distribution D over SMT instances that reflects the target domain of interest, such

as instances generated by a symbolic execution engine in software analysis. The goal of algorithm selection is to construct a selector f that maps a given instance x to a solver $f(x) \in \mathcal{A}$ so as to maximize the expected performance $\mathbb{E}_{x\sim D}\big[u_{f(x)}(x)\big]$.

Two common reference baselines are the *virtual best solver (VBS)* and the *single best solver (SBS)*. The VBS is an idealized oracle that always selects the best solver for each instance, representing an upper bound on the achievable performance of algorithm selection. In contrast, the SBS is one single solver in $\mathcal{A}$ that performs best on average.

In the ML setting, a training instance set S is sampled from D, and the performance of each solver on each training instance is collected. Based on this training dataset, a learning algorithm trains a selector $\hat{f}$ that predicts the most suitable solver for a given instance from its feature representation. The learned selector is evaluated on a separate test set T drawn independently from D.

Two common learning approaches for the algorithm selection problem are:

- *Empirical hardness model (EHM)* [59] is a regression-based algorithm selection method. During training, EHM learns a regression model for each solver to predict its performance on problem instances. At inference time, the solver with the predicted best performance is selected.
- *Pairwise classification (PWC)* [58] reduces the multiclass problem to a set of binary classification tasks. During training, a classifier is constructed for each pair of solvers in $\mathcal{A}$ to predict which solver performs better on a given instance. At inference time, each classifier casts a vote for the preferred solver in its pair, and the solver receiving the most votes is selected.

2.2 Graph Neural Networks

Graph Neural Networks (GNNs) are a class of neural architectures designed for learning graph representations. They broadly follow a recursive neighborhood aggregation (or message passing) scheme, where each node aggregates feature vectors of its neighbors to compute its new feature vectors. Let $G = (V, E)$ be a graph where V represents the set of nodes and E represents the set of edges, with initial node features X_v for each $v \in V$. At each layer k, the representation $\mathbf{h}_v^{(k)}$ of node v is updated as:

$$\mathbf{h}_v^{(k)} = \mathrm{COMBINE}^{(k)}\left(\mathbf{h}_v^{(k-1)}, \mathrm{AGGREGATE}^{(k)}\left(\left\{\mathbf{h}_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)\right),$$

where $\mathcal{N}(v)$ denotes the neighborhood of v, $\mathbf{h}_v^{(0)} = \mathbf{X}_v$, $\mathrm{AGGREGATE}^{(k)}$ is a permutation-invariant function, and $\mathrm{COMBINE}^{(k)}$ is a learnable transformation. After k iterations, each node representation $\mathbf{h}_v^{(k)}$ captures the structural information within the node v's k-hop neighborhood. The representation of the entire graph $\mathbf{h}_G$ can be obtained through pooling [60], for example, by summing the representation vectors of all nodes in the graph.

Different GNN variants, such as GraphSAGE [18], Graph Convolutional Networks (GCN) [23], and Graph Attention Networks (GAT) [53], vary in their neighborhood aggregation and graph-level pooling mechanisms. The expressive power of a GNN for graph representation depends largely on those choices. Xu et al. [57] propose a theoretical framework for analyzing GNNs' representation power, characterizing it in terms of their ability to distinguish non-isomorphic graphs. They show that the discriminative capability of a GNN is upper-bounded by the Weisfeiler–Lehman (WL) graph isomorphism test [54], and that commonly used architectures such as GraphSAGE and GCN are strictly less powerful than this bound.

Motivated by this theoretical analysis, they introduce the Graph Isomorphism Network (GIN), a GNN architecture designed to match the discriminative power of the WL test, i.e., achieving *maximal discriminative power* within the class of message-passing GNNs. GIN adopts sum aggregation together with a multilayer perceptron (MLP) as the update function. The node representation at layer k is updated as

$$\mathbf{h}_v^{(k)} = \mathrm{MLP}^{(k)}\left((1+\epsilon^{(k)})\mathbf{h}_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} \mathbf{h}_u^{(k-1)}\right),$$

where $\epsilon^{(k)}$ is either a learnable parameter or a fixed scalar.

2.3 Transformer-based Text Representation

Transformer-based language models [52] have become the dominant architecture for learning contextual representations from text. They are built upon self-attention mechanisms that enable each token to attend to all other tokens in a sequence, allowing the model to capture long-range dependencies and contextual relationships.

Given an input text sequence of tokens $\tau = (w_1, w_2, \ldots, w_n)$, a transformer encoder produces contextualized token embeddings $\mathbf{z}_1, \ldots, \mathbf{z}_n$, where $\mathbf{z}_i \in \mathbb{R}^{d_T}$. For downstream tasks, a pooling operation is applied to obtain a sentence-level embedding $\mathbf{h}_\tau \in \mathbb{R}^{d_T}$, for example by averaging token embeddings or using a special classification token.

Pretrained transformer models, such as BERT [14] and its variants, are trained on large-scale corpora to learn semantic representations of text. Through this pretraining process, the embedding space is shaped so that semantically similar texts are mapped to nearby vectors, while semantically different texts tend to be farther apart. This geometric property makes transformer embeddings effective for classification, clustering, and retrieval tasks. These pretrained models can either be used directly as feature extractors or further fine-tuned to a downstream task, where model parameters are updated using task-specific data.

3 SMT-Select: Multimodal Learning-Based Algorithm Selection

3.1 System Overview

SMT-Select is a unified multimodal algorithm selection framework that integrates graph and language representations to select a suitable solver for a given SMT instance. Given an input SMT formula together with its natural language description, the framework encodes both inputs into vector representations, fuses them into a joint embedding, and applies a selection layer to determine the solver. An overview of the framework is shown in Figure 1.

The architecture consists of three components: a graph encoder, a language encoder, and a multimodal solver selection layer. The graph encoder extracts a structural representation from the SMT formula, while the language encoder produces a semantic representation from the textual description. These representations are jointly processed by the selection layer to produce the final solver decision.

3.2 Graph Encoder

3.2.1 Graph Construction

We represent each SMT formula as a directed acyclic graph (DAG) $G = (V, E)$ following the AST representation used in Z3 [13]. Nodes $v \in V$ correspond to subexpressions, including function applications, quantifiers, variables, and constants, while directed edges $(u, v) \in E$

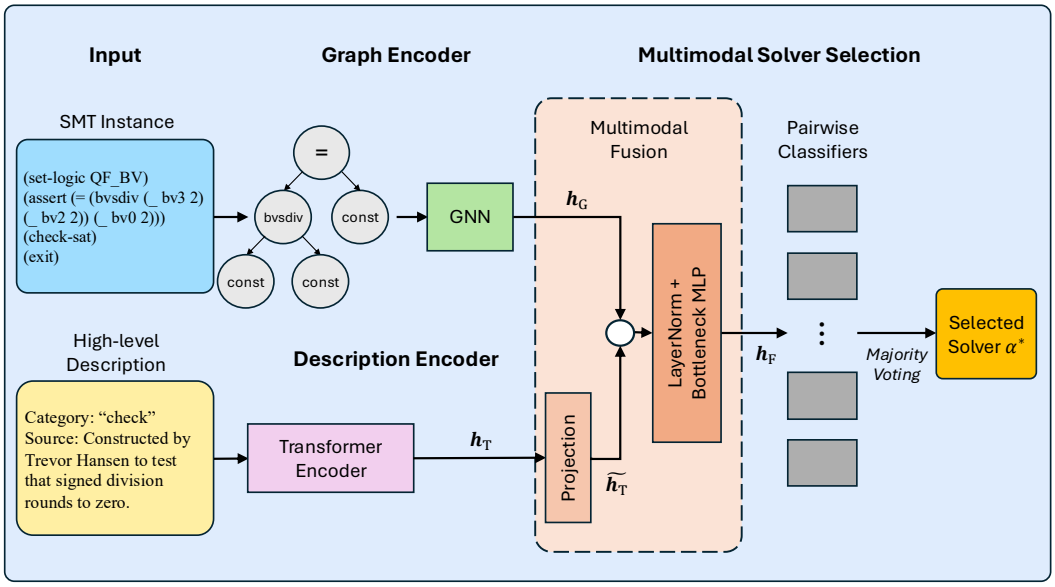

Figure 1 Overview of the SMT-SELECT architecture.

connect each parent expression to its child subterms. Each top-level assertion in the SMT formula is treated as a root node, so a formula with multiple assertions is represented as a DAG with multiple roots.

Each node v is assigned a type $t(v)$ according to its syntactic role. Binding constructs, including `forall`, `exists`, and `lambda`, are labeled by their specific kind. Built-in operator applications are labeled by their operator name. We additionally distinguish uninterpreted function applications, constants, free variables, and bound variables.

3.2.2 Graph Neural Encoding

We encode the formula graph G using a three-layer Graph Isomorphism Network (GIN) with hidden dimension d_G. Each node v is initialized with a learnable embedding $\mathbf{h}_v^{(0)} \in \mathbb{R}^{d_G}$ based on its type $t(v)$. Let $\mathbf{h}_v^{(\ell)}$ denote the representation of node v at layer ℓ. At each layer, node representations are updated through neighborhood aggregation followed by a two-layer MLP with ReLU activation.

Algorithm selection is a graph-level prediction task that requires expressive whole-graph embeddings capable of distinguishing structurally different formulas. We adopt GIN due to its maximal discriminative power and simple architecture. These properties allow the GIN encoder to efficiently capture hierarchical and compositional relationships among subexpressions that influence solver performance.

After the final layer, node representations are aggregated using a global mean pooling to obtain a graph embedding $\mathbf{h}_G \in \mathbb{R}^{d_G}$, which serves as the structural representation of the SMT instance and is passed to the multimodal solver selection module.

3.3 Description Encoder

For each SMT instance, we assume that an associated natural-language description τ is available, for example from SMT-LIB headers. The description provides high-level contextual information, such as the encoded problem, application domain, or problem origin. In practice, such information is often known to users.

By default, description-based features are obtained by encoding this text using the pretrained all-mpnet-base-v2 model [49, 39]. It is a lightweight sentence embedding model with approximately 0.1B parameters and is widely used for generating high-quality text embeddings in classification tasks. In our study, we also compared it with larger alternative models and experimented with fine-tuning all-mpnet-base-v2. However, these variants yielded only marginal performance differences. We therefore mainly use the pretrained all-mpnet-base-v2 for generating description embeddings.

Let d_T denote the dimensionality of the text embedding space ($d_T = 768$ for all-mpnet-base-v2). Given a description text τ, the encoder produces a vector $\mathbf{h}_T \in \mathbb{R}^{d_T}$, which serves as a semantic contextual representation of the instance.

3.4 Multimodal Solver Selection Layer

The multimodal solver selection layer takes as input the graph representation $\mathbf{h}_G$ and the description representation $\mathbf{h}_T$, and produces a solver prediction $\alpha \in \mathcal{A}$ for the given SMT instance. The layer consists of two stages: multimodal fusion and pairwise solver selection.

3.4.1 Multimodal Fusion

Since $\mathbf{h}_T$ typically has much higher dimensionality than $\mathbf{h}_G$, direct concatenation may introduce a dimensional imbalance that biases the fusion layer toward the higher-dimensional $\mathbf{h}_T$. We therefore apply a learnable projection that maps $\mathbf{h}_T \in \mathbb{R}^{d_T}$ to $\tilde{\mathbf{h}}_T \in \mathbb{R}^{d_G}$, aligning it with $\mathbf{h}_G$.

The projected embedding $\tilde{\mathbf{h}}_T$ is then concatenated with $\mathbf{h}_G$ and passed through a layer normalization step followed by a bottleneck fusion layer. This operation models cross-modal interactions and produces a compact joint representation $\mathbf{h}^F \in \mathbb{R}^{d_F}$, which is subsequently fed into the pairwise classification heads. In our implementation, we set $d_F = d_G$.

3.4.2 Pairwise Solver Selection

Let $\mathcal{A} = \{\alpha_1, \ldots, \alpha_K\}$ denote the set of candidate solvers. We adopt the pairwise classification (PWC) scheme and construct a binary classifier $f_{i,j}(\cdot)$ for each unordered solver pair (α_i, α_j) with $i < j$, resulting in a total of $K(K-1)/2$ classifiers.

Each classifier takes the fused representation $\mathbf{h}^F$ as input and produces a logit $\ell_{i,j} = f_{i,j}(\mathbf{h}^F)$, where $f_{i,j}$ is implemented as a two-layer MLP. This logit is converted into a binary decision indicating which solver in the pair is predicted to perform better. The final solver prediction is obtained by aggregating the pairwise decisions via majority voting and selecting the solver that receives the most votes.

3.5 Training Strategy

We adopt a two-stage training procedure with a cost-sensitive pairwise objective as follows:

Stage 1: graph-only pretraining. We first train the graph encoder together with pairwise classification heads using only structural information from the formula. For each unordered solver pair (α_i, α_j) and instance $x \in D$, we define the binary training label

$$y_x^{(i,j)} = \begin{cases} 1 & \text{if } r_{\alpha_i}(x) < r_{\alpha_j}(x), \\ 0 & \text{otherwise,} \end{cases}$$

where $r_\alpha(x)$ denotes the PAR-2 runtime[1] of solver α on instance x. To reflect the importance of performance differences, each training example is assigned a weight

$$w_x^{(i,j)} = \left| r_{\alpha_i}(x) - r_{\alpha_j}(x) \right|.$$

The pairwise classifier $f_{i,j}(x)$ is then trained by minimizing the weighted binary cross-entropy (BCE) loss

$$\ell_{i,j} = \sum_{x \in D} w_x^{(i,j)} \cdot \mathrm{BCE}\big(f_{i,j}(x), y_x^{(i,j)}\big).$$

Stage 2: multimodal training. In the second stage, we train the multimodal fusion module and its pairwise heads while keeping the pretrained graph encoder fixed. The same cost-sensitive pairwise objective is used.

This two-stage design allows the graph encoder to learn stable graph-level representations before introducing semantic information. Freezing the pretrained graph encoder during multimodal training also reduces computational cost. We additionally experimented with fine-tuning the graph encoder during the multimodal stage. However, this did not yield noticeable performance improvements while increasing training cost. We therefore adopt the frozen-graph-encoder design.

An important design choice is the *cost-sensitive training* of each pairwise classifier. Training samples are weighted according to the PAR-2 performance difference between the two solvers. For example, misclassifying a solver that solves an instance while another times out results in a substantial performance loss and thus incurs a high penalty. In contrast, if both solvers solve the instance and their runtimes differ only by a negligible margin, confusing them has little impact. This weighting scheme encourages the model to prioritize distinctions that most affect overall performance. As shown in our experiments, this cost-sensitive PWC scheme contributes significantly to the strong performance of SMT-Select.

3.6 SMT-Select Variants

We introduce several variants of the proposed framework.

SMT-Select-Graph+Text denotes the full multimodal model described above, which integrates graph representations with natural-language description embeddings and is trained using the two-stage procedure.

SMT-Select-Graph refers to the GNN-only variant, trained using only Stage 1 (graph-only pretraining) without incorporating description information.

SMT-Select-Lite is a lightweight variant, which relies on simple syntactic features instead of graph representation. The feature vector has 215 dimensions and is consistent with the meta-information provided in the SMT-LIB catalog [45, 44]. Specifically, it includes counts of 9 SMT-LIB commands, such as assertions, function declarations, constant declarations, and sort declarations; counts of 204 predefined theory symbols, such as `bvadd`, `mod`, and `str.substr`; as well as the maximum term depth and file size. We refer the reader to the SMT-LIB catalog for the complete feature specification. SMT-Select-Lite uses the same cost-sensitive PWC algorithm selection pipeline, but replaces the MLP models with lightweight SVM classifiers.

[1] PAR-2 is a penalized runtime metric that assigns each solved benchmark its actual runtime and each unsolved benchmark twice the timeout.

Table 1 Meta-information for the selected SMT-COMP 2024 logics: benchmark count, number of competition solvers, SBS and VBS solved counts, description rate, and average file size.

Logic	Benchmarks	Solvers	SBS solved	VBS solved	Desc.%	Avg. file size (KB)
ABV	2487	3	837 (cvc5)	971	100.0	4.1
ALIA	1537	3	298 (iProver v3.9)	390	100.0	5.0
BV	1040	5	952 (cvc5)	1002	98.6	303.2
QF_BV	10703	6	10489 (Bitwuzla)	10560	81.5	1528.4
QF_IDL	1208	5	1047 (Z3-alpha)	1101	79.1	1637.6
QF_LIA	4825	5	4514 (OpenSMT)	4726	92.9	861.4
QF_NRA	3104	5	2813 (Z3-alpha)	2933	98.7	670.9
QF_SLIA	23722	4	23276 (Z3-Noodler)	23700	83.5	35.1
UFNIA	6279	3	3688 (cvc5)	3759	99.6	75.5

SMT-Select-Lite+Text is an extension of SMT-Select-Lite by additionally incorporating description embeddings. In this variant, the syntactic feature vector is concatenated with the text embedding to form a joint representation, and the same cost-sensitive pairwise SVM framework is applied to this combined feature vector.

SMT-Select-Text is a variant that relies solely on description features. It adopts the same cost-sensitive pairwise SVM framework.

Due to their training and computational efficiency, SMT-Select-Lite and SMT-Select-Lite+Text serve as fallback selectors for SMT-Select-Graph and SMT-Select-Graph+Text, respectively, when graph construction fails within the specified timeout.

4 Experimental Setup

We evaluate SMT-Select through a rigorous and comprehensive experimental study. Beyond performance gains, we also examine practical factors such as training and inference overhead.

4.1 Benchmark, Solver, and Performance

Our experiments use the open SMT benchmarks from SMT-LIB and retrieve solver performance data from the accompanying metadata catalog [45]. We conduct experiments on 9 selected logics from the single-query track of SMT-COMP 2024. These logics are selected based on the following criteria:

1. **VBS must solve at least 50 more benchmarks than the competition winner (SBS)**. This ensures sufficient headroom for algorithm selection to demonstrate measurable improvements.
2. **Natural-language description coverage in SMT-LIB must be at least 75%**, as our method relies on textual descriptions as part of the feature representation.

The selected logics and their meta-information are listed in Table 1. For each logic, we partition the competition benchmarks into 75% for training and 25% for testing. Based on the training dataset, we train the algorithm selector to choose among the competition solvers. The competition runs were executed on the VerifierCloud cluster operated by SoSy-Lab at LMU Munich, with a wall-clock timeout of 20 minutes per benchmark-solver pair. Further details about the competition setup can be found on the official SMT-COMP website [8].

4.2 SMT-Select Implementation

4.2.1 Feature Extraction

Syntactic features and formula ASTs. We use Klammerhammer [43] to extract the syntactic features used by SMT-Select-Lite, and the AST parser of Z3-4.15.0 [13] to construct formula DAGs for the GNN models. We impose a 5-second feature extraction timeout. Syntactic features extraction is efficient, with only three QF_BV instances and one QF_NRA instance exceeding this timeout; in such cases, SMT-Select-Lite falls back to the training-set SBS solver. Graph construction for SMT-Select-Graph is substantially more expensive due to AST parsing, node labeling, and format conversion. When graph construction fails within the time limit, SMT-Select-Graph fall back to SMT-Select-Lite. Detailed graph-construction failure statistics are reported in Table 10 in the Appendix.

Natural language descriptions. For natural language descriptions, we concatenate the `category`, `application`, and `description` fields defined in the SMT-LIB catalog, extracted by Klammerhammer from benchmark headers. In our dataset, the `category` field takes one of three values: "industrial", "crafted", and "random". The `application` field includes entries such as "Software verification", "Symbolic Execution of Python Programs", or "Rewrite rule verification". The `description` field often contains richer information about the benchmark and its source, for example: "The instance is generated by a runtime predictive analysis system called RVPredict. This problem was added to SMT-LIB by Bohan Li." Note that not all fields are present for every benchmark. For benchmarks without descriptions, we assign a simple placeholder. For example, for QF_IDL instances from the `asp` family that lack textual descriptions, we use: "This is a QF_IDL instance from the asp family."

We also observe that descriptions in ABV and ALIA exhibit very limited variability. All benchmarks in ABV and the vast majority of those in ALIA are generated using Ultimate Automizer [19] to encode software verification tasks from SV-COMP [4, 5]. Their SMT-LIB descriptions are nearly identical and only indicate this origin. To address this, we enrich these descriptions with information from the corresponding SV-COMP benchmark, including a brief summary of the program and, when available, the property being checked; for example, "The original SV-COMP benchmark checks unreachability and memory cleanup in a circular doubly linked list program."

Description embeddings. Descriptions are encoded into vector representations using pretrained sentence embedding models. For our main results, we use all-mpnet-base-v2, a widely used sentence embedding model introduced in 2021 that remains a strong baseline for text representation. It supports a maximum input length of 384 tokens; longer descriptions are truncated (see Appendix Section A.1 for truncation statistics).

We additionally compare against the recent embedding model, Qwen3-Embedding-0.6B [61] in an ablation study on description encoders. Compared to all-mpnet-base-v2, Qwen3-Embedding-0.6B has roughly six times more parameters, a 32K-token context window, and a 1024-dimensional embedding space.

We further explore whether task-specific fine-tuning improves performance. Specifically, we fine-tune all-mpnet-base-v2 using the SetFit framework [50] on the algorithm selection task, where each description in the training set is labeled with the best solver for the corresponding instance. SetFit trains the encoder through supervised contrastive learning under a multiclass classification objective. The fine-tuned model is then used to generate description embeddings in the same way as the pretrained model.

4.2.2 ML Model Training

SVM. SMT-Select-Lite, SMT-Select-Text, and SMT-Select-Lite+Text use SVMs as their pairwise classifiers. For each unordered solver pair, we train a binary SVM with an RBF kernel. Input features are standardized prior to training.

GNN. For graph-based variants, we instantiate the graph encoder described in Section 3.2.2 with hidden dimension $d_G = 64$ and dropout rate 0.1. In Stage 1 (Section 3.5), this GIN backbone and the pairwise classification heads are trained jointly. For each unordered solver pair, we use a two-layer MLP, ReLU activation, and dropout rate 0.1 as the binary classification head. Models are trained using Adam with a learning rate 10^{-3} and batch size 64 for up to 200 epochs. Early stopping is applied based on validation loss.

Multimodal layer. In Stage 2, we freeze the pretrained GIN backbone and train only the multimodal fusion module together with its pairwise classification heads. The description encoder remains fixed and provides precomputed description embeddings. We use the same cost-sensitive pairwise objective as in Stage 1 and optimize models using Adam with the same learning rate and batch size for 200 epochs.

For graph-based models, when the training set of a logic contains more than 3000 instances (namely QF_BV, QF_LIA, QF_SLIA, and UFNIA), we exclude instances that are solved by all solvers within 24 seconds. This filtering reduces training time on large datasets and removes instances that are unlikely to be informative for distinguishing solver performance.

4.3 Baselines

We evaluate the SMT-Select variants against the following baselines, using the recommended default training procedures and hyperparameter settings for each method.

MachSMT. MachSMT [47] is an ML-based SMT algorithm selection framework that relies on 189 handcrafted syntactic features, which are similar to the feature set used in SMT-Select-Lite. Its primary approach is the Empirical Hardness Model (EHM), where an AdaBoost regressor is trained to predict the PAR-2 runtime for each solver. It also supports a PWC variant, which we include in our evaluation. We evaluate MachSMT at the latest commit e524a61.

Sibyl. Sibyl [29] adopts a *pairwise margin ranking* training objective. For each solver α, it learns a GAT-based scoring function $g_\alpha(x)$ for instance x, where higher scores indicate better predicted performance. The scoring models are trained using a margin ranking loss. For each instance x, solvers are ranked according to their PAR-2 runtimes $r_\alpha(x)$. For any solver pair (α_i, α_j) with $r_{\alpha_i}(i) < r_{\alpha_j}(x)$ and an instance $x \in D$, the loss is defined as

$$\ell_{i,j}(x) = \max\left(0,\, m_x^{(i,j)} - \left(g_{\alpha_i}(x) - g_{\alpha_j}(x)\right)\right),$$

where the margin $m_x^{(i,j)}$ reflects the *rank difference* between α_i and α_j on instance x. The overall objective sums this loss over all solver pairs and training instances. We evaluate Sibyl at the latest commit 133d33f.

Sibyl uses PySMT [16] to construct formula ASTs, but the current implementation does not support certain operators appearing in UFNIA and QF_SLIA benchmarks. We therefore exclude these two logics from the Sibyl evaluation. Additional experimental details are provided in Appendix Section A.5.

Since our objective is to study SMT feature representations in the setting of offline supervised algorithm selection, where a single solver is selected before solving, we do not include approaches that focus on online learning, such as MedleySolver [37], or solver scheduling, such as SMTgazer [35], both of which rely on the same feature set as MachSMT.

4.4 Computational Environment

All experiments were conducted on a server equipped with dual AMD EPYC 7763 processors (128 cores, 256 threads in total) and 1 TB RAM. GNN and multimodal model training, as well as all-mpnet-base-v2 fine-tuning, were performed on an NVIDIA GeForce RTX 3090 GPU with 24 GB memory.

The system runs Ubuntu 22.04 with Linux kernel 6.8.0. Our tools and experiments were implemented in Python 3.11. GNN models were built using PyTorch 2.10.0 with CUDA 12.8 and PyTorch Geometric 2.7.0. Language embedding models were used through Sentence Transformers 5.2.2. SVM classifiers were implemented using scikit-learn 1.8.0.

4.5 Evaluation Measures

We evaluate algorithm selectors on the held-out test set. For each test benchmark, the selector chooses a solver, and its performance is retrieved from the competition dataset. We additionally account for the selector's overhead, including both feature extraction and model inference time, when computing the final performance.

We evaluate performance using two metrics: the average PAR-2 score and the number of test benchmarks solved within the timeout. We also report results as the fraction of the SBS–VBS performance gap closed by the selector. Since the VBS represents an upper bound on achievable performance and the competition winner (SBS) provides a practical baseline, this metric quantifies how much of the potential improvement over the winner is realized by the selector. A value of 100% indicates performance equivalent to VBS, while negative values indicate performance worse than the competition winner solver.

Througout the experiments, we observe non-trivial variability across different train-test splits. To account for this, we generate five independent random train–test splits using different random seeds and report the average performance across these splits.

5 Experimental Results

Table 2 and Table 3 summarize the experimental results in terms of PAR-2 score and number of solved instances, respectively. The PAR-2 results in Table 2 are reported as the proportion of the SBS–VBS gap closed by the algorithm selector.

5.1 Effectiveness of the Cost-sensitive PWC Mechanism

A key observation is that all SMT-Select variants consistently outperform existing algorithm selectors and competition winners. This improvement is largely attributed to the cost-sensitive PWC mechanism adopted by all SMT-Select variants.

Despite relying on similar syntactic features and a simpler ML model, SMT-Select-Lite substantially outperforms MachSMT. A likely reason is the difference in the underlying selection framework: MachSMT uses EHM by default, whereas SMT-Select-Lite adopts cost-sensitive PWC training. Although MachSMT also provides a PWC variant, it does not incorporate cost-sensitive weighting. We additionally evaluated MachSMT-PWC in Table 9 of the Appendix, where it performs worse than MachSMT-EHM.

Table 2 Experimental results on the held-out test set, measured by the PAR-2 SBS–VBS gap closed (%). Results are averaged over five random train–test splits.

	Without Description				With Description		
	MachSMT	Sibyl	SMT-Select		SMT-Select		
			Lite	Graph	Text	Lite+Text	Graph+Text
ABV	7.83	17.97	71.45	76.56	62.83	67.15	**79.44**
ALIA	27.73	33.34	55.30	55.07	43.68	56.42	**67.65**
BV	17.11	12.00	38.22	42.43	32.31	49.20	**53.82**
QF_BV	7.35	-29.45	35.40	47.68	55.63	55.64	**57.52**
QF_IDL	6.06	-30.47	64.53	69.66	44.92	71.69	**81.82**
QF_LIA	84.70	76.66	89.01	**92.56**	91.95	91.19	91.65
QF_NRA	17.25	-3.50	44.61	50.61	44.94	50.72	**55.17**
QF_SLIA	76.96	—	90.70	92.82	87.34	**93.50**	**93.50**
UFNIA	-129.22	—	14.79	24.48	-3.95	23.95	**31.84**

Table 3 Difference in instances solved vs. competition winner (SBS) on the held-out test set. Results are averaged over five random train–test splits.

	Ref.	Without Description				With Description		
	VBS	MachSMT	Sibyl	SMT-Select		SMT-Select		
				Lite	Graph	Text	Lite+Text	Graph+Text
ABV	+34.8	+2.2	+6.4	+25.0	+26.6	+22.0	+23.2	**+27.6**
ALIA	+20.6	+4.4	+7.4	+10.4	+11.4	+9.2	+11.0	**+13.8**
BV	+13.6	+0.8	-0.6	+5.2	+5.0	+3.4	+6.4	**+6.8**
QF_BV	+15.8	+0.2	+2.2	+5.6	+7.6	**+9.8**	+9.4	+9.2
QF_IDL	+14.6	+1.0	-2.2	+10.8	+11.4	+7.4	+11.6	**+12.8**
QF_LIA	+52.4	+41.6	+38.0	+47.2	+48.0	**+48.2**	+47.2	+47.0
QF_NRA	+32.6	+6.6	-0.2	+15.6	+17.8	+16.0	+18.0	**+18.2**
QF_SLIA	+112.4	+87.6	—	+102.8	+106.0	+106.4	**+107.2**	+106.0
UFNIA	+16.4	-22.8	—	+1.8	+4.4	-0.6	+4.4	**+5.8**

Similarly, SMT-Select-Graph outperforms the GNN-based selector Sibyl. While both methods employ pairwise training, Sibyl uses solver ranking differences as training margins for solver utility functions, whereas our approach uses PAR-2 runtime differences to weight the training of pairwise classifiers.

5.2 SMT-Select-Lite versus SMT-Select-Graph

In terms of both PAR-2 and number of instances solved, SMT-Select-Graph outperforms SMT-Select-Lite in eight of the nine evaluated logics, with a negligible difference in the remaining one. Nevertheless, SMT-Select-Lite remains competitive, closing a non-trivial portion of the SBS–VBS gap across all logics and achieving performance close to SMT-Select-Graph in most cases. Given that SMT-Select-Lite requires substantially less training time (see Table 5) and significantly fewer computational resources, including GPU and RAM, it serves as a strong, lightweight alternative for practical SMT algorithm selection.

Table 4 Performance comparison of SMT-Select (Graph) under different natural-language description modes, measured by PAR-2 SBS–VBS gap closed (%) on the test set (mean over five random train–test splits).

	Without Description	Family-only Description	Full Description
ABV	76.56	77.52	**79.44**
ALIA	55.07	62.15	**67.65**
BV	42.43	49.55	**53.82**
QF_BV	47.68	49.83	**57.52**
QF_IDL	69.66	80.42	**81.82**
QF_LIA	**92.56**	90.79	91.65
QF_NRA	50.61	52.81	**55.17**
QF_SLIA	92.82	93.26	**93.50**
UFNIA	24.48	26.60	**31.84**

Table 5 Average training time (minutes) for SMT-Select variants.

	Lite	Text	Lite+Text	Graph	Graph+Text
ABV	≤ 0.1	≤ 0.1	≤ 0.1	0.2	0.3
ALIA	≤ 0.1	≤ 0.1	≤ 0.1	≤ 0.1	0.3
BV	≤ 0.1	≤ 0.1	≤ 0.1	1.1	1.6
QF_BV	1.6	0.3	1.9	312.7	321.0
QF_IDL	0.2	≤ 0.1	0.2	65.1	65.6
QF_LIA	0.4	≤ 0.1	0.4	156.4	158.8
QF_NRA	≤ 0.1	≤ 0.1	≤ 0.1	38.1	39.3
QF_SLIA	0.4	0.8	1.5	14.4	17.8
UFNIA	≤ 0.1	≤ 0.1	≤ 0.1	12.3	12.8

5.3 Role of Description Features

We first observe that SMT-Select-Text, which relies purely on description features, already serves as a reasonably strong selector, even outperforming SMT-Select-Graph on the QF_BV logic. However, its effectiveness strongly depends on the quality and diversity of the descriptions. For example, in UFNIA, many benchmarks share nearly identical descriptions, limiting the usefulness of textual information alone.

Encouragingly, incorporating description features consistently benefits both SMT-Select-Lite and SMT-Select-Graph, reducing PAR-2 runtime in eight of the nine evaluated logics for both variants. Overall, the full multimodal version SMT-Select-Graph+Text, which integrates graph and description features, achieves the best PAR-2 performance in eight of the nine logics among all variants.

To further examine how textual descriptions contribute to performance, we conduct an ablation study comparing SMT-Select using descriptions containing only the benchmark family information (e.g., “This is a QF_IDL instance from the asp family.”) against using the full SMT-LIB descriptions. As shown in Table 4, incorporating the full descriptions consistently outperforms the family-information-only setting, suggesting that finer-grained semantic information provides additional benefits for algorithm selection.

5.4 Description Encoder Choice and Fine-tuning

The ablation study results for different description encoders are shown in Table 8 in the Appendix. We observe no clear pattern indicating that either the larger encoder Qwen3-Embedding-0.6B or the fine-tuned all-mpnet-base-v2 (methods described in Section 4.2.1) provides consistent performance gains. Arguably, this is because the selector performance depends primarily on the downstream pairwise training stage, which can learn robust decision boundaries given sufficiently informative embeddings. Thus, stronger embedding models do not necessarily yield noticeable performance gains.

5.5 Training Cost and Inference Overhead

Table 5 reports the training times of different SMT-Select variants. Training lightweight SVM-based models is inexpensive. In contrast, training SMT-Select-Graph is substantially more costly, especially for logics containing many large benchmarks. For example, training on QF_BV and QF_LIA takes an average of 5.2 and 2.6 hours, respectively. Since SMT-Select-Graph+Text reuses the pretrained GNN to generate graph embeddings, training the additional fusion MLP introduces only a small overhead beyond GNN pretraining.

For selection overhead, which includes feature extraction and model inference time, the SVM-based selectors incur little cost, less than 0.1 seconds on average. For graph-based variants, the overhead remains modest with the graph construction time capped at 5 seconds. While QF_IDL has the largest average overhead of 1.7 seconds, all other logics remain below 1 second. Note that the selection overhead is accounted for in our performance evaluation.

5.6 Performance Variability

We observe non-trivial variability across the five train–test splits, as shown by the variability statistics in Table 6 and Table 7 in the Appendix. However, this variability appears to stem primarily from the intrinsic difficulty differences across splits, rather than from changes in the relative competitiveness of the algorithm selectors.

6 Related Work

SATzilla [59] is an early work applying ML-based algorithm selection to SAT solvers. It introduced a set of handcrafted features designed to capture SAT instance hardness, including statistics such as the clause-to-variable ratio, and probing features obtained from short runs of solvers. Similarly, expert-designed features have been used for algorithm selection in the constraint programming context, in systems such as CPHydra [7], SUNNY [1], and Ulrich-Oltean et al. [51].

MachSMT [46] applied algorithm selection to SMT solvers. It constructed a large feature set for SMT formulas, mainly consisting of counts of theory symbols. This feature set has also been adopted by the online selector MedleySolver [37] and scheduler SMTGazer [35]. Other work, such as SMTQuery [26], developed handcrafted theory-specific features.

In the exploration of automated feature extraction, Hůla et al. [20] first applied GNNs to SMT algorithm selection. Specifically, they followed the EHM approach and trained one independent GCN model per solver to predict runtime. In subsequent work, Sibyl [29] employed a shared GAT backbone for graph encoding and attached solver-specific scoring heads trained with a pairwise margin ranking objective. Our GNN variant follows a similar shared-backbone design, enabling the model to learn a unified representation of SMT formulas while reducing redundant computation across solvers. However, our approach differs from

Sibyl in several key aspects, including the selection architecture (pairwise classifiers vs. solver-specific scoring functions), training objective (runtime differences as weights vs. rank differences as margins), and GNN architecture (GIN vs. GAT). Graph learning methods have also been applied in algorithm selection for SAT [62] and verification [41, 31, 42, 28].

AS-LLM [56] introduced a large language model-enhanced method for algorithm selection. However, its focus is on learning representations for algorithms instead of problem instances. Pellegrino et al. [36] proposed a transformer-based approach to learn instance features for algorithm selection, training a BERT-style encoder with a context window of 2048 tokens. While this window size suits their optimization problems expressed in the high-level modeling language Essence, raw SMT instances are typically much longer. For example, QF_BV benchmarks in SMT-COMP 2024 have an average file size of approximately 1.5MB, which far exceeds the typical context window of standard text embedding models. Instead of directly encoding raw SMT formulas, we leverage natural-language contextual descriptions, and combine them with formula graph representations.

Taken together, prior work shows a clear progression from expert-designed features toward automated representation learning for algorithm selection. While recent work has increasingly focused on graph-based methods and has begun to explore language-model-based techniques, we are the first to integrate structural graph representations with natural-language descriptions within a unified neural framework for SMT algorithm selection.

7 Conclusion, Limitations, and Future Work

We present SMT-Select, a multimodal SMT algorithm selection framework that combines graph representations of SMT formulas with their natural-language descriptions. Using a GNN as the graph encoder together with description features derived from pretrained language models, SMT-Select demonstrates consistent state-of-the-art performance across diverse SMT logics. We also introduce lightweight variants that require substantially lower training cost and computational resources while remaining competitive in performance. Our experimental results demonstrate that a cost-sensitive PWC formulation provides a strong and effective selection mechanism, and that incorporating contextual natural-language descriptions consistently improves performance beyond structural formula features alone.

One limitation of our framework is that the pairwise training cost may increase substantially as the number of candidate solvers grows. Future work could explore reducing this cost through techniques such as dyadic feature representations [48] or training on frugal subsets of instances [27]. Moreover, our work focuses on the setting where a selector chooses and commits to a single solver for a given instance based on offline training data. Alternative strategies, such as solver scheduling [20, 35], online adaptation [37, 30], or parallel portfolio execution [21, 55], fall outside the scope of this work, though the proposed representation and learning framework may be extendable to these settings. More broadly, the framework could potentially generalize to objectives beyond solver runtime optimization and to domains outside SMT, such as Constraint Answer Set Programming (CASP).

References

1 Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. SUNNY: a lazy portfolio approach for constraint solving. *Theory Pract. Log. Program.*, 14(4-5):509–524, 2014. doi:10.1017/S1471068414000179.

2 Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org, 2016.

3 Clark W. Barrett, Leonardo Mendonça de Moura, and Aaron Stump. SMT-COMP: satisfiability modulo theories competition. In *Computer Aided Verification, 17th International Conference, CAV 2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 20–23. Springer, 2005. doi:10.1007/11513988_4.

4 Dirk Beyer. Automatic verification of C and Java programs: SV-COMP 2019. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 133–155. Springer International Publishing, 2019. doi:10.1007/978-3-030-17502-3_9.

5 Dirk Beyer. Competition on software verification and witness validation: SV-COMP 2023. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 495–522. Springer Nature Switzerland, 2023. doi:10.1007/978-3-031-30820-8_29.

6 Miquel Bofill, Joan Espasa, and Mateu Villaret. Relaxed exists-step plans in planning as SMT. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 563–570, 2017. doi:10.24963/ijcai.2017/79.

7 Derek Bridge, Eoin O'Mahony, and Barry O'Sullivan. *Case-Based Reasoning for Autonomous Constraint Solving*, pages 73–95. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-21434-9_4.

8 Martin Bromberger, François Bobot, and Martin Jonáš. SMT-COMP 2024. https://smt-comp.github.io/2024/, 2024. Accessed: 2025-12-22.

9 Cristian Cadar, Daniel Dunbar, and Dawson Engler. KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 209–224, 2008. URL: http://www.usenix.org/events/osdi08/tech/full_papers/cadar/cadar.pdf.

10 Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. EXE: Automatically generating inputs of death. *ACM Trans. Inf. Syst. Secur.*, 12(2), December 2008. doi:10.1145/1455518.1455522.

11 Michael Cashmore, Daniele Magazzeni, and Parisa Zehtabi. Planning for hybrid systems via Satisfiability Modulo Theories. *Journal of Artificial Intelligence Research*, 67:235–283, February 2020. doi:10.1613/jair.1.11751.

12 Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011. doi:10.1145/1995376.1995394.

13 Leonardo de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *2008 Tools and Algorithms for Construction and Analysis of Systems*, pages 337–340. Springer, Berlin, Heidelberg, March 2008. doi:10.1007/978-3-540-78800-3_24.

14 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. doi:10.48550/arXiv.1810.04805.

15 Alan M. Frisch, Warwick Harvey, Christopher Jefferson, Bernadette Martínez Hernández, and Ian Miguel. Essence : A constraint language for specifying combinatorial problems. *Constraints An Int. J.*, 13(3):268–306, 2008. doi:10.1007/S10601-008-9047-Y.

16 Marco Gario and Andrea Micheli. PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In *SMT Workshop 2015*, 2015.

17 Arie Gurfinkel, Temesghen Kahsai, Anvesh Komuravelli, and Jorge A. Navas. The SeaHorn verification framework. In *Computer Aided Verification*, pages 343–361, Cham, 2015. Springer International Publishing. doi:10.1007/978-3-319-21690-4_20.

18 Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

19 Matthias Heizmann, Yu-Fang Chen, Daniel Dietsch, Marius Greitschus, Jochen Hoenicke, Yong Li, Alexander Nutz, Betim Musa, Christian Schilling, Tanja Schindler, and Andreas Podelski. Ultimate Automizer and the search for perfect interpolants. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 447–451. Springer International Publishing, 2018. doi:10.1007/978-3-319-89963-3_30.

20 Jan Hůla, David Mojžíšek, and Mikoláš Janota. Graph neural networks for scheduling of SMT solvers. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 447–451, 2021. doi:10.1109/ICTAI52525.2021.00072.

21 Haniye Kashgarani and Lars Kotthoff. Automatic parallel portfolio selection. In *ECAI 2023 - 26th European Conference on Artificial Intelligence*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, pages 1215–1222. IOS Press, 2023. doi:10.3233/FAIA230398.

22 Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evol. Comput.*, 27(1):3–45, March 2019. doi:10.1162/evco_a_00242.

23 Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=SJU4ayYgl.

24 Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Mag.*, 35(3):48–60, September 2014. doi:10.1609/aimag.v35i3.2460.

25 Daniel Kroening and Ofer Strichman. *Decision Procedures - An Algorithmic Point of View, Second Edition*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016. doi:10.1007/978-3-662-50497-0.

26 Mitja Kulczynski, Kevin Lotz, Florin Manea, Danny Bøgsted Poulsen, and Paul Sarnighausen-Cahn. SMTQuery: Analysing SMT-LIB string benchmarks. In *Formal Methods: Foundations and Applications*, pages 22–34, Cham, 2025. Springer Nature Switzerland. doi:10.1007/978-3-031-78116-2_2.

27 Erdem Kus, Özgür Akgün, Nguyen Dang, and Ian Miguel. Frugal algorithm selection (short paper). In *30th International Conference on Principles and Practice of Constraint Programming, CP 2024*, volume 307 of *LIPIcs*, pages 38:1–38:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CP.2024.38.

28 Will Leeson and Matthew B. Dwyer. Algorithm selection for software verification using graph neural networks. *ACM Trans. Softw. Eng. Methodol.*, 33(3), March 2024. doi:10.1145/3637225.

29 Will Leeson, Matthew B Dwyer, and Antonio Filieri. Sibyl: Improving software engineering tools with SMT selection. In *Proceedings of the 45th International Conference on Software Engineering*, pages 2185–2197. IEEE Press, 2023. doi:10.1109/ICSE48619.2023.00184.

30 Zhengyang Lu. AlphaSMT: A reinforcement learning guided SMT solver. Master's thesis, University of Waterloo, 2023.

31 Zhengyang Lu, Po-Chun Chien, Nian-Ze Lee, Arie Gurfinkel, and Vijay Ganesh. Btor2-Select: machine learning based algorithm selection for hardware model checking. In *Computer Aided Verification*, pages 296–311, Cham, 2025. Springer Nature Switzerland. doi:10.1007/978-3-031-98668-0_15.

32 Zhengyang Lu, Paul Sarnighausen-Cahn, Jiahao Chen, Arie Gurfinkel, Florin Manea, and Vijay Ganesh. Learning SMT algorithm selection with high-level natural-language descriptions. In *Foundations of Information and Knowledge Systems - Poster Track*, pages 353–358, Cham, 2026. Springer Nature Switzerland. doi:10.1007/978-3-032-21540-6_23.

33 Zhengyang Lu, Stefan Siemer, Piyush Jha, Joel Day, Florin Manea, and Vijay Ganesh. Layered and staged Monte Carlo Tree Search for SMT strategy synthesis. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 1907–1915, 2024. doi:10.24963/ijcai.2024/211.

34 Zhengyang John Lu, Joel Day, Piyush Jha, Paul Sarnighausen-Cahn, Stefan Siemer, Florin Manea, and Vijay Ganesh. Novel tree-search method for synthesizing SMT strategies. *Acta Informatica*, 62(3):28, 2025. doi:10.1007/s00236-025-00495-x.

35 Chuan Luo, Shaoke Cui, Jianping Song, Xindi Zhang, Wei Wu, Chanjuan Liu, Shaowei Cai, and Chunming Hu. SMTgazer: Learning to schedule SMT algorithms via Bayesian optimization. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1273–1285, 2025. doi:10.1109/ASE63991.2025.00109.

36 Alessio Pellegrino, Özgür Akgün, Nguyen Dang, Zeynep Kiziltan, and Ian Miguel. Transformer-Based Feature Learning for Algorithm Selection in Combinatorial Optimisation. In *31st International Conference on Principles and Practice of Constraint Programming (CP 2025)*, volume 340 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:22, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CP.2025.31.

37 Nikhil Pimpalkhare, Federico Mora, Elizabeth Polgreen, and Sanjit A. Seshia. MedleySolver: Online SMT algorithm selection. In *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 453–470. Springer, 2021. doi:10.1007/978-3-030-80223-3_31.

38 Elizabeth Polgreen, Kevin Cheang, Pranav Gaddamadugu, Adwait Godbole, Kevin Laeufer, Shaokai Lin, Yatin A. Manerkar, Federico Mora, and Sanjit A. Seshia. UCLID5: Multi-modal formal modeling, verification, and synthesis. In *Computer Aided Verification: 34th International Conference, CAV 2022*, pages 538–551, Berlin, Heidelberg, 2022. Springer-Verlag. doi:10.1007/978-3-031-13185-1_27.

39 Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, November 2019. doi:10.48550/arXiv.1908.10084.

40 John R. Rice. The algorithm selection problem. *Adv. Comput.*, 15:65–118, 1976. doi:10.1016/S0065-2458(08)60520-3.

41 Cedric Richter, Eyke Hüllermeier, Marie-Christine Jakobs, and Heike Wehrheim. Algorithm selection for software validation based on graph kernels. *Automated Software Engg.*, 27(1–2):153–186, June 2020. doi:10.1007/s10515-020-00270-x.

42 Cedric Richter and Heike Wehrheim. Attend and represent: a novel view on algorithm selection for software verification. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ASE '20, pages 1016–1028, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3324884.3416633.

43 Hans-Jörg Schurr and François Bobot. Klammerhammer. https://github.com/SMT-LIB/SMT-LIB-db/tree/main/klhm, 2025. Accessed: 2026-02-26.

44 Hans-Jörg Schurr, François Bobot, Mathias Preiner, Aina Niemetz, Clark Barrett, Pascal Fontaine, and Cesare Tinelli. Exploring the SMT-LIB benchmark library. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 150–169, Cham, 2026. Springer Nature Switzerland. doi:10.1007/978-3-032-22752-2_8.

45 Hans-Jörg Schurr, Mathias Preiner, Aina Niemetz, Clark W. Barrett, Pascal Fontaine, and Cesare Tinelli. SMT-LIB catalog 2025 (version 1), July 2025. Accessed on 2025-12-16. doi:10.5281/zenodo.16290040.

46 Joseph Scott, Aina Niemetz, Mathias Preiner, Saeed Nejati, and Vijay Ganesh. MachSMT: A machine learning-based algorithm selector for SMT solvers. In *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021*, volume 12652 of *Lecture Notes in Computer Science*, pages 303–325, 2021. doi:10.1007/978-3-030-72013-1_16.

47 Joseph Scott, Aina Niemetz, Mathias Preiner, Saeed Nejati, and Vijay Ganesh. Algorithm selection for SMT. *Int. J. Softw. Tools Technol. Transf.*, 25(2):219–239, 2023. doi:10.1007/S10009-023-00696-0.

48 Alexander Tornede, Marcel Wever, and Eyke Hüllermeier. Extreme algorithm selection with dyadic feature representation. In *Discovery Science*, pages 309–324, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-61527-7_21.

49 Sentence Transformers. all-mpnet-base-v2, 2021. URL: https://huggingface.co/sentence-transformers/all-mpnet-base-v2.

50 Lewis Tunstall, Nils Reimers, Unso Eun Seo Jo, Luke Bates, Daniel Korat, Moshe Wasserblat, and Oren Pereg. Efficient few-shot learning without prompts, 2022. doi:10.48550/arXiv.2209.11055.

51 Felix Ulrich-Oltean, Peter Nightingale, and James Alfred Walker. Learning to select SAT encodings for pseudo-boolean and linear integer constraints. *Constraints An Int. J.*, 28(3):397–426, November 2023. doi:10.1007/s10601-023-09364-1.

52 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

53 Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL: https://openreview.net/forum?id=rJXMpikCZ.

54 Boris Weisfeiler and A. A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16, 1968.

55 Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo de Moura. A concurrent portfolio approach to SMT solving. In *Computer Aided Verification*, pages 715–720, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-02658-4_60.

56 Xingyu Wu, Yan Zhong, Jibin Wu, Bingbing Jiang, and Kay Chen Tan. Large language model-enhanced algorithm selection: Towards comprehensive algorithm representation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 5235–5244. International Joint Conferences on Artificial Intelligence Organization, 2024. doi:10.24963/ijcai.2024/579.

57 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=ryGs6iA5Km.

58 L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Evaluating component solver contributions to portfolio-based algorithm selectors. In *Proc. SAT*, LNCS 7317, pages 228–241. Springer, 2012. doi:10.1007/978-3-642-31612-8_18.

59 Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *J. Artif. Intell. Res.*, 32:565–606, 2008. doi:10.1613/jair.2490.

60 Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 4805–4815, Red Hook, NY, USA, 2018. Curran Associates Inc. URL: https://proceedings.neurips.cc/paper/2018/hash/e77dbaf6759253c7c6d0efc5690369c7-Abstract.html.

61 Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025. doi:10.48550/arXiv.2506.05176.

62 Zhanguang Zhang, Didier Chételat, Joseph Cotnareanu, Amur Ghose, Wenyi Xiao, Hui-Ling Zhen, Yingxue Zhang, Jianye Hao, Mark Coates, and Mingxuan Yuan. GraSS: Combining graph neural networks with expert knowledge for SAT solver selection. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024*, pages 6301–6311. ACM, 2024. doi:10.1145/3637528.3671627.

A Additional Experimental Results and Details

A.1 Truncation Statistics of the all-mpnet-base-v2 Encoder

The all-mpnet-base-v2 model supports a maximum input length of 384 tokens; longer descriptions are be truncated. In five of the nine evaluated logics, none or fewer than 1% of descriptions exceed this limit. Truncation occurs more frequently in ABV (83.43%), ALIA (68.18%), BV (11.35%), and QF_NRA (13.76%).

For ABV, ALIA, and BV, fewer than 10 tokens are removed on average among the truncated benchmarks. In QF_NRA, truncated benchmarks lose an average of 102.6 tokens, mostly consisting of citation information.

A.2 Result Variability

Table 6 and Table 7 report the mean and standard deviation of the test-set PAR-2 scores and the number of solved instances, respectively, for all SMT-Select variants over five train–test splits.

Table 6 Test set PAR-2 (seconds) by SMT-Select variants. Mean ± std over five random train–test splits.

	Lite	Text	Lite+Text	Graph	Graph+Text
ABV	1488.83 ± 34.94	1500.67 ± 31.25	1495.93 ± 32.98	1482.26 ± 29.39	**1478.44 ± 26.22**
ALIA	1875.39 ± 31.96	1890.27 ± 22.06	1873.77 ± 26.38	1875.64 ± 32.09	**1859.08 ± 32.04**
BV	189.35 ± 25.65	200.73 ± 30.27	175.24 ± 27.00	188.65 ± 62.19	**172.25 ± 48.24**
QF_BV	54.37 ± 5.56	50.84 ± 4.83	50.84 ± 4.66	52.22 ± 5.12	**50.51 ± 4.86**
QF_IDL	301.51 ± 15.53	323.71 ± 18.44	295.14 ± 14.39	298.32 ± 15.73	**286.71 ± 19.99**
QF_LIA	72.68 ± 12.34	68.17 ± 9.96	69.36 ± 9.57	**67.20 ± 12.62**	68.83 ± 12.39
QF_NRA	194.02 ± 13.10	193.51 ± 17.92	187.93 ± 19.15	187.83 ± 17.45	**183.64 ± 17.50**
QF_SLIA	8.10 ± 0.88	9.61 ± 0.96	**6.84 ± 0.44**	7.16 ± 1.31	**6.84 ± 0.87**
UFNIA	1020.63 ± 28.92	1026.68 ± 25.65	1018.36 ± 27.82	1018.35 ± 28.00	**1016.73 ± 26.47**

Table 7 Test set number of instances solved by SMT-Select variants. Mean ± std over five random train–test splits.

	Lite	Text	Lite+Text	Graph	Graph+Text
ABV	236.8 ± 9.1	233.8 ± 8.1	235.0 ± 8.6	238.4 ± 7.7	**239.4 ± 6.9**
ALIA	86.4 ± 5.3	85.2 ± 3.5	87.0 ± 4.5	87.4 ± 5.8	**89.8 ± 5.4**
BV	241.8 ± 3.2	240.0 ± 3.3	243.0 ± 3.1	241.6 ± 6.8	**243.4 ± 5.2**
QF_BV	2628.8 ± 5.3	**2633.0 ± 4.9**	2632.6 ± 4.8	2630.8 ± 4.9	2632.4 ± 4.9
QF_IDL	269.4 ± 1.6	266.0 ± 2.1	270.2 ± 1.6	270.0 ± 1.4	**271.4 ± 2.5**
QF_LIA	1178.2 ± 5.8	**1179.2 ± 4.5**	1178.2 ± 4.6	1179.0 ± 5.9	1178.0 ± 5.7
QF_NRA	718.8 ± 4.3	719.2 ± 5.8	721.2 ± 6.3	721.0 ± 5.4	**721.4 ± 5.8**
QF_SLIA	5916.2 ± 2.6	5919.8 ± 2.0	**5920.6 ± 1.0**	5919.4 ± 3.5	5919.4 ± 2.3
UFNIA	924.8 ± 19.5	922.4 ± 17.5	927.4 ± 18.5	927.4 ± 18.5	**928.8 ± 17.5**

A.3 Description Encoder Choice and Fine-tuning

Table 8 Performance comparison of different description encoders within SMT-Select-Text, SMT-Select-Lite+Text, and SMT-Select-Graph+Text, measured by the PAR-2 SBS–VBS gap closed (%) on the held-out test set. Results are averaged over five random train–test splits, and the best-performing encoder for each variant is shown in bold.

	SMT-Select-Text			SMT-Select-Lite+Text			SMT-Select-Graph+Text		
	mpnet	Qwen3	finetune	mpnet	Qwen3	finetune	mpnet	Qwen3	finetune
ABV	62.83	62.45	**63.26**	67.15	65.20	**70.99**	79.44	77.76	**79.50**
ALIA	43.68	**44.50**	37.59	56.42	**59.69**	51.93	**67.65**	66.17	60.78
BV	32.31	34.65	**36.14**	49.20	**53.44**	52.25	53.82	55.00	**57.01**
QF_BV	55.63	54.62	**58.65**	55.64	56.23	**57.73**	**57.52**	57.07	56.06
QF_IDL	44.92	**48.18**	46.86	71.69	70.67	**74.43**	**81.82**	81.64	78.32
QF_LIA	**91.95**	91.58	91.34	91.19	91.63	**91.66**	**91.65**	91.25	90.74
QF_NRA	44.94	**50.09**	42.12	50.72	**54.00**	53.40	**55.17**	51.16	51.81
QF_SLIA	**87.34**	87.33	87.09	**93.50**	93.35	93.30	93.50	94.00	**95.23**
UFNIA	**-3.95**	-3.97	**-3.95**	23.95	**24.62**	23.94	31.84	**32.87**	30.95

Table 8 compares the performance of SMT-Select-Text, SMT-Select-Lite+Text, and SMT-Select-Graph+Text using different description encoders, including all-mpnet-base-v2, Qwen3-Embedding-0.6B, and SetFit-fine-tuned all-mpnet-base-v2.

A.4 Performance of MachSMT and SMT-Select-Lite

Table 9 reports the PAR-2 SBS–VBS gap closed by MachSMT-EHM (default), MachSMT-PWC, and SMT-Select-Lite on the test set. As indicated in the original MachSMT paper [46], MachSMT-PWC performs worse than the EHM version while incurring a higher training cost.

Table 9 PAR-2 SBS–VBS gap closed (%) on the test set: MachSMT-EHM, MachSMT-PWC vs. SMT-Select-Lite. Mean over five random train–test splits.

	MachSMT-EHM	MachSMT-PWC	SMT-Select-Lite
ABV	7.83	31.48	**71.45**
ALIA	27.73	32.34	**55.30**
BV	17.11	4.47	**38.22**
QF_BV	7.35	-61.55	**35.40**
QF_IDL	6.06	-8.32	**64.53**
QF_LIA	84.70	61.69	**89.01**
QF_NRA	17.25	-5.95	**44.61**
QF_SLIA	76.96	25.21	**90.70**
UFNIA	-129.22	-305.52	**14.79**

Table 10 Average failure rate (%) of graph construction for Sibyl and SMT-Select-Graph.

Logic	Sibyl	SMT-Select-Graph
ABV	0.0	0.0
ALIA	0.0	0.0
BV	0.9	0.7
QF_BV	11.9	16.2
QF_IDL	14.2	16.0
QF_LIA	4.3	6.8
QF_NRA	2.8	1.2
QF_SLIA	59.8	3.6
UFNIA	83.8	2.3

A.5 Sibyl Experimental Details

Sibyl does not natively provide a fallback mechanism when graph construction fails. During our evaluation, we therefore introduce a simple fallback strategy: if the PySMT graph parser fails to construct the formula graph within a 10-second time limit, the system defaults to selecting the single best solver (SBS) from the training set. We use a longer graph construction time limit for Sibyl because the Python-based PySMT parser is slower than the C++-based Z3 parser used in SMT-Select-Graph. This adjustment keeps the graph-construction failure rate comparable between the two systems, as shown in Table 10.

In addition, the current PySMT implementation does not support certain theory-specific operators, including the integer division operator `div` and several string-theory operators such as `str.to_re`. Consequently, graph construction fails for 83.8% of UFNIA benchmarks and 59.8% of QF_SLIA benchmarks, and we therefore exclude these two logics from the Sibyl evaluation.

We acknowledge that the fallback solver used by Sibyl (the train-set SBS) may be weaker than the fallback solver used by SMT-Select-Graph (SMT-Select-Lite). However, for the two logics with higher graph-construction failure rates, QF_BV and QF_IDL, Table 2 and Table 3 show that Sibyl performs no better than SBS overall. Therefore, this fallback choice does not affect the relative comparison between Sibyl and SMT-Select-Graph.