A Machine Learning–Enhanced Decision Layer for SAT-Based Combinatorial Interaction Testing

MSc Artificial Intelligence and Data Engineering Research Project (24/25)

Anonymous Author

ABSTRACT

Overview at the end.

1 INTRODUCTION

What is the context, the research problem and why is it important?

2 BACKGROUND

This section outlines the foundational concepts of combinatorial interaction testing, its encoding as a *Satisfiability* (SAT) problem, and learning-based heuristics. Particular focus will be given to graph neural networks (GNNs), proximal policy optimization (PPO), and meta-learning. These components collectively inform the development of a novel adaptive decision layer tailored to combinatorial tests, specifically utilizing SAT-based representations for their efficient generation and optimization.

2.1 Combinatorial Interaction Testing

The testing of modern software systems presents significant challenges, not only due to the large number of configuration parameters but also because of the complex interactions among them. An increasing body of research indicates that failures often arise from subtle interactions among small subsets of parameters rather than from individual parameters in isolation. This observation forms the core reasoning for *Combinatorial Interaction Testing* (CIT), a software testing technique that aims to reduce the cost of exhaustive testing while preserving high fault detection effectiveness by generating test suites that cover all desired interactions.

A foundational empirical study by Kuhn et al. [6] demonstrated that almost all observed software failures were triggered by four-way interactions or fewer. Their analysis, conducted in multiple industrial systems, provided compelling evidence that low-strength interaction tests, such as pairwise (2-way) or three-way tests, can detect the vast majority of faults. They introduced the notion of *pseudo-exhaustive coverage*, where testing all t-way combinations for small t provides coverage close to exhaustive testing, especially when parameters are discrete or can be grouped into equivalence classes. However, they also cautioned that the appropriate selection of t should be informed by the system's fault profile, as higher-order interactions, though rare, may still be critical.

Expanding on this insight, Bures and Ahmed [3] conducted a real-world evaluation of CIT using model-based mutation testing. Their findings reinforced the importance of interaction strength: three-way test suites consistently uncovered defects missed by pairwise testing, demonstrating the limitations of low-strength CIT in certain contexts. At the same time, they highlighted a key trade-off: higher interaction strengths lead to larger and more computationally expensive test suites. Despite this, their results showed that modest values of t, such as 2 or 3, often strike a balance between effectiveness and feasibility, offering substantial improvements over ad hoc or random test strategies.

Despite its empirical strengths, CIT faces a fundamental computational challenge. Both Kuhn et al.[6] and Bures and Ahmed[3] acknowledge that generating minimal t-way test suites under realistic constraints is NP-hard. Even when the optimal interaction strength t is known, constructing a suite that ensures full t-wise coverage without prohibitive growth in size remains a non-trivial optimization problem. This scalability barrier accentuates the need for more intelligent, constraint-aware strategies for test suite generation.

2.2 SAT-Based Formulations for CIT Generation

The generation of minimal t-way test suites in the presence of complex configuration constraints has led researchers to encode CIT problems as Boolean satiability (SAT) problems. This enables the use of modern, highly optimized SAT solvers to construct test suites that guarantee t-way coverage while strictly respecting system constraints [4, 15].

In this formulation, each parameter-value assignment within the system under test is encoded as a distinct Boolean variable. Both the system constraints and the desired coverage requirements are expressed as clauses in Conjunctive Normal Form (CNF) [15]. Although many real-world systems involve non-Boolean or multi-valued configuration options, these can be accommodated through a transformation known as *flattening* [5]. Flattening converts multi-valued parameters into sets of mutually exclusive Boolean variables and their associated constraints, thus producing models compatible with Boolean SAT solvers. Although this transformation can increase the number of variables and clauses, it enables the use of advanced SAT-solving techniques such as *Conflict-Driven Clause Learning* (CDCL), *Boolean Constraint Propagation* (BCP), and dynamic backtracking, which significantly improve solver efficiency on large CIT encodings.

2.2.1 *The Capabilities of SAT-Based CIT.*

The adoption of SAT solvers for CIT offers several compelling benefits, particularly in constraint-rich and resource-sensitive testing environments:

- **Efficient Constraint Solving:** SAT solvers excel at navigating complex constraint satisfaction problems, efficiently pruning the search space using mechanisms such as clause learning and propagation. Cohen et al. [4] showed that incremental SAT solving could reduce generation time while preserving full t-way coverage.
- **Provably Optimal Test Suites:** Yamada et al. [15] demonstrated that SAT-based methods can generate minimal test

suites that provably meet t-way requirements, critical in resource-limited settings.

- **Expressive Constraint Modeling:** The CNF encoding enables precise modeling of complex inter-parameter dependencies. This expressiveness supports real-world systems with rich configuration logic [4].
- **Faster Processing with Flattened Models:** Flattening introduces more variables but improves solver performance by aligning models with Boolean expectations. Henard et al. [5] found that flattened CNF models solved faster than equivalent SMT encodings on various benchmarks.

2.2.2 Limitations and Open Challenges in SAT-Based CIT. Despite their strengths, SAT solvers exhibit several limitations when applied to CIT, particularly in terms of adaptability:

- **Rigid Heuristics in CDCL Solvers:** Strategies such as VSIDS [8], CHB, and LRB [9] are statically designed and applied uniformly across all instances of SAT, regardless of the domain of the problem or structural properties. This rigid decision layer does not exploit the structural patterns and domain-specific properties present in CIT formulations, potentially leading to inefficiencies or suboptimal solver performance.

2.3 Machine Learning for Enhanced SAT Solving Heuristics

Machine learning (ML) has been increasingly applied to SAT solving to improve solver performance by learning effective heuristics. In recent years, research has moved toward embedding ML directly within the solver loop, guiding decisions like variable branching, clause deletion, and restart scheduling. For example, Bergin et al. [1] used Random Forest classifiers to predict the satisfiability of partial assignments to inform branching decisions.

Static heuristics like VSIDS are agnostic to instance-specific properties, making them brittle in domains with complex or irregular structure. As Oh [9] observes, the progress in classical heuristic design has plateaued, motivating the exploration of alternative data-driven strategies. In contrast, ML-based heuristics can learn from distributions of SAT instances and dynamically adapt solver behavior to optimize decision-making on a per-instance basis. This adaptability is especially valuable in CIT, where instance structures vary substantially due to diverse parameter interactions and constraint encodings.

2.3.1 Graph Neural Networks. GNNs operate on graph-structured data by iteratively updating node embeddings based on local neighborhoods. SAT formulas in CNF can be naturally modeled as bipartite graphs, where variables and clauses form disjoint node sets, with edges denoting literal participation. This structure enables message passing to capture both local and global dependencies in the formula.

A notable example is **Graph-Q-SAT** by Kurin et al. [7], which integrates GNN-derived embeddings into a Q-learning policy for branching. Variable-clause bipartite graphs are encoded, and a GNN learns contextual node representations passed to a Q-function that estimates the expected reward of branching on each variable. This approach surpasses classical heuristics like VSIDS on synthetic benchmarks and generalizes to larger instances, demonstrating the adaptability of learned, structure-aware policies.

Other models also exploit the representational power of GNNs. **NeuroSAT** [12] uses message passing to predict satisfiability, while **NeuroBack** [14] and **RLAF** [13] apply GNNs within CDCL solvers to learn variable selection and feedback strategies. In each case, GNNs function as expressive encoders, learning latent structural features that inform solver decision-making without relying on defaulted heuristics.

2.3.2 Proximal Policy Optimization. Proximal Policy Optimization (PPO) [11] is a widely adopted policy gradient method designed to improve training stability and sample efficiency in *reinforcement learning* (RL). Its suitability for structured environments like SAT solving arises from its ability to constrain policy updates, thus avoiding erratic behavior during long-horizon or tree-structured decision processes.

PPO optimizes a clipped surrogate objective that limits the deviation between successive policies. The objective function is:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right],$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ denotes the probability ratio between the new and old policies, and $\hat{A}_t$ is the estimated advantage at the time step t. The clipping parameter ϵ limits the magnitude of the update, encouraging stable but effective learning.

In SAT solving, PPO has been successfully deployed in **RDC-SAT** [16], which uses a *distributed PPO* (DPPO) variant to train a GNN-based actor-critic model for variable decomposition in SAT divide-and-conquer solving. The environment captures tree-structured transitions, and delayed rewards are computed by solving time metrics stored in a binary tree. This design enables PPO to model long-term dependencies and generalize across both random and industrial CNF instances.

3 APPROACH

We propose a novel adaptive decision layer for SAT-based CIT, combining GNNs, PPO, and meta-learning. Our approach integrates this learned module into the **CaDiCaL 2.0** SAT solver [2] to dynamically guide variable branching decisions. This section presents the research questions, describes the system architecture, details the training methodology, and explains the reinforcement learning strategy used to implement adaptive variable selection.

3.1 Research Questions

As discussed in Sections 2.2.2 and 2.3, SAT-based CIT solvers typically rely on static branching heuristics that are insensitive to instance-specific structure. This rigidity can degrade performance on configuration-rich or structurally diverse problems. We address this gap by investigating the following research questions:

- **RQ1:** Can GNN-based branching heuristics outperform classical strategies on SAT-encoded CIT instances in terms of solving efficiency and test suite compactness?
- **RQ2:** Does reinforcement learning via PPO lead to more effective variable selection than static or GNN-only approaches, by enabling policy adaptation during solving?

- **RQ3:** Can meta-learning improve generalization across diverse CIT instance distributions, reducing adaptation time and improving performance on previously unseen configurations?

3.2 Adaptive Decision Layer Architecture

To improve the adaptability of the solver, we replace the static branching heuristic of **CaDiCaL** with a learned policy network driven by the structure of the instance. This adaptive decision layer consists of three integrated components:

- **GNN-Based Instance Encoding:** SAT formulas are modeled as bipartite graphs with clause and variable nodes. A GNN computes embeddings for each unassigned variable by aggregating local and global structural information from the CNF formula.
- **Reinforcement Learning Policy:** A PPO-trained policy network receives the GNN embeddings and selects the next variable to branch on. Rewards are derived from solver performance metrics such as conflict counts, backtracks, and total solving time.
- **Meta-Learning Initialization:** We use *Model-Agnostic Meta-Learning* (MAML) to pre-train the policy across a range of CIT instances. This enables rapid adaptation to new instances with few gradient steps, enhancing generalization across diverse configurations.

These components jointly define a modular, instance-sensitive decision layer that interacts with the solver during the search. Unlike static heuristics, this layer evolves with experience and adjusts to the structural properties of the current formula.

3.3 Reinforcement Learning for Variable Selection

We frame branching in the SAT solver as a sequential decision-making process. At each step, the solver must choose an unassigned variable whose assignment will most efficiently lead to satisfiability. Our formulation of *reinforcement learning* (RL) guides this decision using GNN-based state representations and a PPO policy network.

3.3.1 CNF-to-Graph Representation and GNN Architecture. Each CIT instance is first encoded as a CNF formula where satisfying assignments correspond to valid test cases with full t-way interaction coverage. The CNF is converted into a bipartite graph (detailed in Section 2.3.1).

The initial node features include literal polarity counts, clause lengths, and activity scores. These features are updated as solving progresses through assignments, propagations, and clause learning.

We experiment with *graph convolutional networks* (GCNs) and *graph attention networks* (GATs) to compute variable embeddings. These GNNs perform multiple rounds of message passing, enabling each variable to aggregate structural context from its neighboring clauses and variables. The resulting embeddings capture the relevance of each unassigned variable within the current formula state.

3.3.2 Reinforcement Learning Formulation, State Definition, and Action Space. The sequential decision-making process inherent in SAT solvers, particularly in selecting branching variables, naturally aligns with a RL framework. Within this setting, the SAT solver functions as the environment, exposing the dynamic state of the CNF formula, while the agent, instantiated as a GNN-based policy network, learns to select branching variables in a way that minimizes overall solving effort. At each decision point, the agent receives an observation comprising the current solver configuration, which includes the partial assignment of variables, the set of active clauses yet to be satisfied or falsified, and the learned clauses accumulated from conflict analysis. These components are processed to form graph-based embeddings of variables and clauses, encoding both structural and semantic information about the instance.

These embeddings serve as input to the policy network, providing a rich and instance-specific abstraction of the problem state.

The action space consists of the set of currently unassigned Boolean variables. The policy network outputs a probability distribution in this space, representing the likelihood of selecting each variable for branching.

3.3.3 Reward Design and Credit Assignment. An effective reward function is critical for aligning the agent's behavior with the goal of efficient SAT solving. In our formulation, we adopt a hybrid reward scheme that balances immediate feedback from solver operations with sparse signals tied to overall task success. Immediate rewards provide fine-grained guidance: a small penalty ($R_{\text{conflict}} = -0.1$) is applied when a decision leads to a conflict, while successful unit propagations yield a minor positive reward ($R_{\text{propagate}} = +0.01$). Additional positive feedback is given to reduce the number of unassigned variables or unsatisfied clauses, reflecting incremental progress. Sparse rewards reinforce long-term objectives, issuing a substantial reward ($R_{\text{solve}} = +1.0$) upon successful CNF resolution, and a strong penalty ($R_{\text{timeout}} = -1.0$) when the solver fails to complete a given resource budget.

Reward signals are attributed directly to branching actions, consistent with established practices in RL for symbolic search. Long-term dependencies are captured through the evolving solver state and the cumulative return, avoiding the need for explicit backtracking of credit across earlier decisions. For reward accumulation, we explore two mechanisms: an average credit assignment strategy, which computes the mean reward per action and suits stationary environments; and an exponentially recency-weighted method with decay rate $\alpha \in [0.01, 0.99]$, which prioritizes recent transitions and is more responsive in non-stationary scenarios where problem difficulty may shift dynamically. To reduce variance and improve training stability, reward updates are batch-aligned with key solver events such as clause learning, propagation cascades, and instance termination, ensuring that credit assignment is synchronized with meaningful structural transitions in the solving process.

3.3.4 Policy Optimization with PPO. As introduced in Section 2.3.2, we adopt PPO to train our GNN-based branching policy. In our implementation, PPO operates on an actor-critic architecture tuned to the SAT solving domain.

The **actor network**, comprising the GNN encoder and a *multilayer perceptron* (MLP), outputs a probability distribution over unassigned variables at each decision point. The **critic network**, implemented as a separate MLP, estimates the state value function. This enables advantage estimation and stabilizes training through variance reduction.

During training, the agent interacts with the SAT solver to generate rollouts consisting of solver states, selected branching actions, and corresponding rewards. These trajectories are used to compute advantage estimates via *Generalized Advantage Estimation* (GAE), and both networks are updated using the clipped PPO objective.

3.3.5 Branching Variable Selection and Integration. Once the GNN encodes the current SAT state and the policy network produces a probability distribution over unassigned variables, the agent selects the next branching variable.

During training, variable selection follows a stochastic policy: a variable is sampled from the predicted distribution to encourage exploration and facilitate policy improvement. At evaluation time, a greedy strategy is used, selecting the variable with the highest predicted probability.

The selected variable is returned to the SAT solver to guide the branching decision, replacing the default heuristic. This selection process is embedded within the solver's decision loop, allowing real-time adaptive control throughout the solving process.

3.4 Generalization Strategy via Meta-Learning

To support generalization across diverse and unseen CIT instances, we integrate the MAML framework into our training pipeline.

The goal is to enable rapid adaptation of the GNN-based branching policy to new SAT-encoded CIT problems with minimal additional training. In this setting, each meta-learning task corresponds to solving a SAT instance derived from a distinct CIT configuration—defined by unique parameters such as interaction strength, number of factors, and value domains.

MAML optimizes the model parameters (of both the GNN encoder and the policy network) so that they are easily adaptable. This is achieved through a bi-level training process:

- **Inner loop (task adaptation):** For each CIT-derived SAT instance in a meta-batch, the policy is fine-tuned over a few gradient steps using rewards obtained during solving.
- **Outer loop (meta-update):** After adaptation to all tasks in the batch, the original model parameters are updated based on post-adaptation performance across tasks. This encourages the model to learn representations and policies that generalize well to new instance distributions.

This meta-learning setup enhances scalability by avoiding the need to retrain heuristics from scratch for each new CIT configuration. Instead, the model takes advantage of structural regularities in all instances to achieve a fast and effective adaptation.

3.5 Experimental Setup and Evaluation

3.5.1 Dataset Selection and CNF Generation. We evaluate our method using established CIT benchmarks that cover a diverse range of realistic configuration scenarios. Two primary sources are used:

- Ten feature models used by Henard et al. [5], originally introduced by Petke et al. [10], representing software product lines and configuration systems.
- Five real-world systems and thirty synthetic benchmarks from Cohen et al. [4], including Apache, GCC, SPIN, and Bugzilla.

The models are converted to CNF using a flattening process (see Section 2.2). The final CNF formulas are emitted in DIMACS format for compatibility with our SAT solver pipeline.

3.5.2 Tooling and Implementation. Our experimental framework is implemented in Python for modularity and reproducibility. SAT solving and CNF manipulation are handled using the PySAT toolkit (v3.2.2) with **CaDiCaL 2.0** as the CDCL backend. CaDiCaL's *ExternalPropagator* interface is used to integrate the learned branching policy into the decision loop, allowing dynamic guidance without disrupting its core solution pipeline, including inprocessing and clause learning.

GNN-based components are implemented using PyTorch (v2.7.1) and PyTorch Geometric (v2.7.0).

All components are containerized with fixed seeds and locked dependencies. Experiments are orchestrated through reproducible scripts and deterministic logs to ensure consistent solver-learning behavior across CIT benchmarks.

3.5.3 Baselines. We evaluate our method against strong, established baselines to assess both solving performance and test suite compactness.

Glucose 4.1 and **CaDiCaL 2.0** with default heuristics. These represent high-performance CDCL solvers that use industry-standard static branching strategies such as VSIDS. They provide a competitive benchmark for assessing the effectiveness of our learned heuristic.

3.5.4 Performance Metrics. We evaluate the proposed adaptive heuristic using a comprehensive set of performance metrics that span both SAT solving efficiency and combinatorial testing effectiveness.

SAT Solver Metrics.

- **Solving Time (s):** Wall-clock time required to solve each CNF instance or determine unsatisfiability, with a timeout threshold.
- **Conflict Count:** Number of conflicts encountered during search; lower values indicate improved pruning heuristic efficiency.
- **Decision Count:** Total number of branching decisions made; fewer decisions suggest a more targeted search trajectory.
- **Solved Instances (%):** Proportion of instances successfully solved within the timeout limit, reflecting robustness and general applicability.

Combinatorial Testing Metrics.

- **Test Suite Size:** Number of test cases generated to achieve complete t-way interaction coverage. This reflects both solver scalability and solution quality.
- **Coverage Validity:** Verification that the resulting test suites meet the full required t-wise interaction coverage, ensuring functional correctness.

4 EVALUATION

What are the results of your work? What is the significance of the results? What can be learned from it?

5 RELATED WORK

How does your work compare to other work in the area?

6 CONCLUSION

REFERENCES

[1] Ruth Bergin, Marco Dalla, Andrea Visentin, Barry O'Sullivan, and Gregory Provan. 2023. Using Machine Learning Classifiers in SAT Branching [Extended Abstract]. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, Vol. 16. 169–170. https://doi.org/10.1609/socs.v16i1.27298

[2] Armin Biere, Tobias Faller, Karcsi Fazekas, Mathias Fleury, Nils Froleyks, and Fredrik Pollitt. 2024. CaDiCaL 2.0. In *Computer Aided Verification (CAV 2024) (Lecture Notes in Computer Science, Vol. 14681)*, Arie Gurfinkel and Vijay Ganesh (Eds.). Springer, Cham. https://doi.org/10.1007/978-3-031-65627-9_7

[3] Miroslav Bures and Bestoun S. Ahmed. 2017. On the Effectiveness of Combinatorial Interaction Testing: A Case Study. In *Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security (Companion Volume)*. IEEE, 69–76. https://doi.org/10.1109/QRS-C.2017.20

[4] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. 2008. Constructing Interaction Test Suites for Highly-Configurable Systems in the Presence of Constraints: A Greedy Approach. *IEEE Transactions on Software Engineering* 34, 5 (2008), 633–650. https://doi.org/10.1109/TSE.2008.50

[5] Christopher Henard, Mike Papadakis, and Yves Le Traon. 2015. Flattening or not of the combinatorial interaction testing models?. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 1–4. https://doi.org/10.1109/ICSTW.2015.7107443

[6] D. Richard Kuhn, Dolores R. Wallace, and Albert M. Gallo Jr. 2004. Software Fault Interactions and Implications for Software Testing. *IEEE Transactions on Software Engineering* 30, 6 (2004), 418–421. https://doi.org/10.1109/TSE.2004.24

[7] Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. 2020. Can *Q*-Learning with Graph Networks Learn a Generalizable Branching Heuristic for a SAT Solver? arXiv:1909.11830 [cs.LG] https://arxiv.org/abs/1909.11830

[8] Jia Hui Liang, Vijay Ganesh, Ed Zulkoski, Atulan Zaman, and Krzysztof Czarnecki. 2015. Understanding VSIDS Branching Heuristics in Conflict-Driven Clause-Learning SAT Solvers. arXiv:1506.08905 [cs.LO] https://arxiv.org/abs/1506.08905

[9] Chanseok Oh. 2016. *Improving SAT Solvers by Exploiting Empirical Characteristics of CDCL*. Ph.D. Dissertation. New York University, Department of Computer Science, New York University.

[10] Justyna Petke, Shin Yoo, Myra B. Cohen, and Mark Harman. 2013. Efficiency and early fault detection with lower and higher strength combinatorial interaction testing. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (Saint Petersburg, Russia) *(ESEC/FSE 2013)*. Association for Computing Machinery, New York, NY, USA, 26–36. https://doi.org/10.1145/2491411.2491436

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG] https://arxiv.org/abs/1707.06347

[12] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. 2019. Learning a SAT Solver from Single-Bit Supervision. arXiv:1802.03685 [cs.AI] https://arxiv.org/abs/1802.03685

[13] Jan Tönshoff and Martin Grohe. 2025. Learning from Algorithm Feedback: One-Shot SAT Solver Guidance with GNNs. arXiv:2505.16053 [cs.LG] https://arxiv.org/abs/2505.16053

[14] Wenxi Wang, Yang Hu, Mohit Tiwari, Sarfraz Khurshid, Kenneth McMillan, and Risto Miikkulainen. 2024. NeuroBack: Improving CDCL SAT Solving using Graph Neural Networks. arXiv:2110.14053 [cs.AI] https://arxiv.org/abs/2110.14053

[15] Akihisa Yamada, Takashi Kitamura, Cyrille Artho, Eun-Hye Choi, Yutaka Oiwa, and Armin Biere. 2015. Optimization of Combinatorial Testing by Incremental SAT Solving. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. 1–10. https://doi.org/10.1109/ICST.2015.7102599

[16] Shumao Zhai and Ning Ge. 2025. Learning Splitting Heuristics in Divide-and-Conquer SAT Solvers with Reinforcement Learning. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=uUsL07BsMA

A ACCESS TO SOURCE CODE

How can the source code be accessed?

B DECLARATIONS

This report is submitted as part of the requirement for the MSc. Artificial Intelligence and Data Engineering at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed, provided that the source is explicitly acknowledged.