TRANSFER LEARNING FROM FOUNDATIONAL OPTIMIZATION EMBEDDINGS TO UNSUPERVISED SAT REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Foundational optimization embeddings, such as FORGE, have recently emerged as powerful pre-trained representations for mixed-integer programming (MIP) problems. Prior work showed these embeddings can transfer across optimization problem domains while reducing reliance on solver-generated labels. In this work, we investigate whether such representations generalize beyond optimization to decision problems, focusing on Boolean satisfiability (SAT). We adapt the foundational optimization architecture to SAT by mapping conjunctive normal form (CNF) formulas into the same bipartite constraint–variable graph representation used for MIPs. This enables us to study both direct transfer of MIP-pretrained representations and FORGE-SAT, a variant of the same architecture pretrained directly on SAT instances. Our results show that these embeddings capture structural regularities in SAT instances and support unsupervised tasks such as instance clustering and distribution identification. Beyond clustering, we analyze what the learned discrete codewords encode by generating semantic descriptions, validating them on held-out nodes, and testing whether the same interpretations transfer across SAT difficulty levels. We further evaluate the downstream utility of these representations for satisfiability prediction under held-out distribution shift, showing that frozen FORGE-SAT representations provide useful task-relevant signal. Our findings are a step toward a unified representational framework across optimization and decision problems.

1 INTRODUCTION

Mixed-integer programming (MIP) and Boolean satisfiability (SAT) are central formalisms for combinatorial reasoning across a wide range of optimization and decision problems. Recent work has shown that graph neural networks (GNNs) can learn useful structural representations for specific problem families. However, many such models rely on solver-generated labels, must be retrained for each new domain or task, and can generalize poorly across distributions and instance scales.

Foundational optimization embeddings, such as FORGE (Shafi & Kadioglu, 2026), offer an alternative direction. They learn generic representations of MIP instances through *unsupervised pretraining without relying on labels* on large, heterogeneous corpora. These representations have been shown to support transfer across optimization tasks such as cut generation and search guidance. This raises a natural question: ***can a model designed and trained purely for MIP structure provide useful representations for decision problems such as SAT, and if so, how far does this transfer extend?***

In this paper, we investigate transfer learning from foundational optimization embeddings to SAT. We consider several variants of the original FORGE architecture (Shafi & Kadioglu, 2026) that span different degrees of transfer. In FORGE-MIP, we directly reuse the original FORGE model and its MIP-specific features on SAT instances encoded as MIPs. In FORGE-MIP-SAT, we keep the pretrained weights and architecture but replace MIP node features with SAT-specific clause and variable features. Finally, in FORGE-SAT, we retain the architecture but discard the MIP-pretrained weights and pretrain from scratch on SAT instances using SAT features.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

This work provides an empirical study of these variants and systematically analyzes whether the resulting representations are structurally meaningful, semantically interpretable, and useful for downstream tasks. Our main contributions are:

1. **Architecture transfer to SAT**: We adapt the FORGE foundational architecture, originally developed for MIPs, to SAT through a SAT-to-MIP encoding pipeline and, in adapted variants, SAT-specific node features.
2. **Transfer from MIP to SAT:** We show that representations from the pretrained MIP model, FORGE-MIP, retain measurable structural information when applied to SAT, and that SAT-specific feature adaptation further improves this transfer.
3. **SAT-native representation model:** We introduce FORGE-SAT, a SAT-pretrained variant that reuses the FORGE training paradigm while specializing to SAT instances.
4. **Empirical evaluation on SAT benchmarks:** We evaluate all variants on unsupervised clustering and further study FORGE-SAT through codeword interpretation and downstream satisfiability prediction under held-out distribution shift. Together, these experiments provide evidence for transfer from MIP to SAT and characterize the utility and limitations of learned SAT representations.
5. **Open-source pretrained models and pipelines:** We provide pretrained models and training pipelines in an anonymized open-source repository, enabling embeddings for SAT instances, clauses, and variables to be generated out-of-the-box: `https://anonymous.4open.science/r/forge-sat-release-057C/README.md`.

2 BACKGROUND

We briefly review the SAT and MIP formalisms, their bipartite graph representations, and the standard transformation that encodes a SAT formula as a MIP instance.

Boolean Satisfiability (SAT): A propositional formula in conjunctive normal form (CNF) is a conjunction of clauses $F = C_1 \wedge \cdots \wedge C_m$, where each clause C_j is a disjunction of literals over Boolean variables $x_1, \ldots, x_n$. A literal is either a variable x_i or its negation $\neg x_i$. The SAT problem asks whether there exists an assignment $\mathbf{x} \in \{0,1\}^n$ such that all clauses are satisfied.

Mixed-Integer Programming (MIP): A mixed-integer program consists of linear constraints over continuous and integer variables: $\min_{\mathbf{x}} c^\top \mathbf{x}$ s.t. $A\mathbf{x} \leq b$, $\mathbf{x}_\mathcal{I} \in \mathbb{Z}^{|\mathcal{I}|}$, $\mathbf{x}_\mathcal{C} \in \mathbb{R}^{|\mathcal{C}|}$, where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and the variable set is partitioned into integer variables $\mathcal{I}$ and continuous variables $\mathcal{C}$. For feasibility problems such as the SAT encoding, the objective can be taken to be constant.

MIP Bipartite Graph: Following Gasse et al. (2019), it is common to represent a MIP instance as a bipartite graph with constraint nodes and variable nodes, and an edge between constraint i and variable j whenever $A_{ij} \neq 0$. In FORGE (Shafi & Kadioglu, 2026), node features encode properties of constraints (e.g., sense, right-hand side) and variables (e.g., type, bounds, objective coefficient).

Encoding SAT as MIP: A CNF formula can be encoded as a 0–1 MIP by introducing a binary variable x_i for each Boolean variable and a linear inequality for each clause. For a clause $C_j = (\ell_{j1} \vee \cdots \vee \ell_{jk})$, we define $\sum_{t=1}^{k} y_{jt} \geq 1$, where each literal ℓ_{jt} is mapped to a term $y_{jt} = x_i$ if $\ell_{jt} = x_i$, and $y_{jt} = 1 - x_i$ if $\ell_{jt} = \neg x_i$. The resulting system of linear constraints over binary variables is feasible if and only if the original CNF formula is satisfiable.

This SAT-to-MIP encoding allows us to reuse the bipartite constraint–variable graph representation and the foundational FORGE architecture for SAT instances.

3 METHODOLOGY

3.1 FOUNDATIONAL ARCHITECTURE

FORGE is a foundational representation-learning framework that learns generic embeddings of mixed-integer programs (MIPs) through *unsupervised pretraining* on large, heterogeneous corpora.

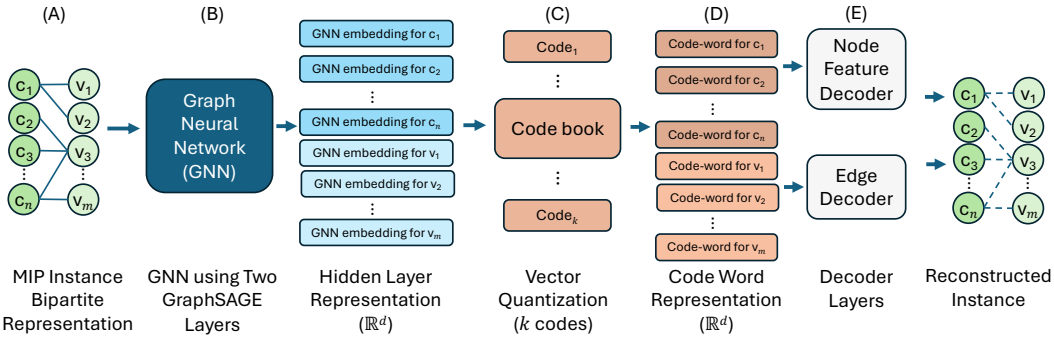

Figure 1: Overview of FORGE architecture Shafi & Kadioglu (2026): Starting with the bipartite representation of MIPs and its GNN embedding (A-B), FORGE uses vector quantized graph autoencoder (C-D) to reconstruct node features and edges (E). This enables learning generic MIP representations with a pre-trained model.

As illustrated in Figure 1, a MIP instance is represented as a bipartite graph of constraint and variable nodes, each equipped with MIP-related features. A two-layer GraphSAGE (Hamilton et al., 2017) encoder produces latent node embeddings, which are discretized through a vector-quantization (VQ) codebook (Yang et al., 2024). A decoder reconstructs node features and incident edges from these codewords. The training objective combines node-feature reconstruction, edge reconstruction, and VQ commitment losses.

Once pretrained over a diverse MIP corpus, FORGE provides:

- **Instance embedding:** a global representation of an instance, obtained either from the distribution of codewords assigned to all nodes or by combining node representations.
- **Constraint & variable embeddings:** representations of individual constraint and variable nodes, along with the discrete codeword assigned to each node.

FORGE is pretrained *entirely without solver labels*, making it a natural candidate for transfer beyond its original MIP domain.

3.2 FORGE APPLIED TO SAT

To apply FORGE to SAT, we first encode each CNF formula as a MIP using the SAT-to-MIP transformation described in Section 2. The resulting bipartite graph has the same structure expected by FORGE, allowing us to reuse the architecture directly. We explore three variants that differ in how features and pretrained weights are used.

FORGE-MIP (zero-change transfer): This variant, as shown in Figure 2 (upper side), directly applies the pretrained FORGE model to SAT instances encoded as MIPs. Both the architecture and the MIP-specific node features are kept unchanged. This represents the most direct transfer setting from optimization to satisfiability.

FORGE-MIP-SAT (feature-adapted transfer): This variant also reuses the pretrained MIP weights but replaces the MIP node features with SAT-specific features with the same input dimensionality as the pretrained model. SAT-specific constraint node features are described by clause-level properties: `width` (number of literals in the clause), `pos_count` (number of positive literals), `neg_count` (number of negative literals), and `pos_neg_ratio` (ratio of positive to negative literals). SAT-specific variable node features are described by degree-based properties: `degree` (total number of clauses the variable appears in), `pos_deg` (number of clauses where it appears positively), `neg_deg` (number of clauses where it appears negated), `pos_neg_ratio` (ratio of positive to negative appearances), `pos_deg_norm` (positive degree normalized by the mean positive degree across all variables), and `neg_deg_norm` (negative degree normalized by the mean negative degree across all variables). The graph structure and pretrained weights remain unchanged, isolating the effect of feature specialization.

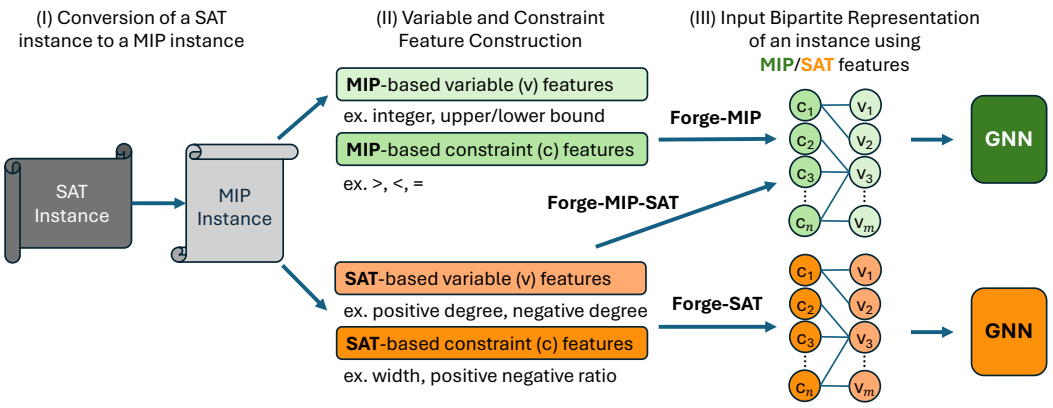

Figure 2: Our transfer learning setup for FORGE applied to SAT to obtain unsupervised SAT representations. (I) convert SAT instances into MIP form. (II) construct either MIP-specific or SAT-specific node features. (III) the bipartite representation, with MIP or SAT node features, is fed into FORGE architecture trained on MIP or SAT instances. Overall, we obtain SAT representations (embeddings) at the instance, clause, and variable levels.

FORGE-SAT (SAT-native model): This variant, as shown in Figure 2 (lower side), retains the FORGE architecture but discards the MIP-pretrained weights. Using the SAT-specific features above, we pretrain the full FORGE model from scratch on SAT instances from the SAT Competition 2022 Anniversary Track. This represents architecture-level transfer: the foundational training paradigm is reused, but the representation is learned entirely from SAT data.

Across these variants, we study two axes of transfer learning:

- **Weight-level transfer:** whether MIP-pretrained parameters provide useful structure for SAT, evaluated via FORGE-MIP and FORGE-MIP-SAT.
- **Architecture-level transfer:** whether the FORGE training paradigm generalizes to SAT when trained from scratch via FORGE-SAT.

Our experiments evaluate whether foundational optimization representations transfer from MIP to SAT, what structure they capture, and whether they provide downstream predictive signal. We evaluate unsupervised clustering of unseen SAT formulas (Section 4), codeword semantics and transfer across difficulty levels (Section 5), and frozen FORGE-SAT representations for satisfiability prediction under held-out distribution shift (Section 6).

4 UNSUPERVISED CLUSTERING

We cluster instance-level embeddings of unseen SAT formulas and evaluate how well the resulting clusters recover problem distributions and joint problem-type/SAT-status categories.

SAT Competition Anniversary Dataset. We use 2,827 instances from the SAT Competition 2022 Anniversary Track (Iser et al., 2024), a curated collection drawn from application, crafted, and main tracks of SAT competitions from 2002–2021, to pretrain FORGE-SAT.

G4SATBench Dataset. We evaluate on G4SATBench (Li et al., 2024), which contains seven problem types spanning random, pseudo-industrial, and combinatorial SAT. These include SR (Selsam et al., 2019), 3-SAT (Lauria et al., 2017), Community Attachment (CA) (Giráldez-Cru & Levy, 2015), Popularity–Similarity (PS) (Giráldez-Cru & Levy, 2017), k-Clique, k-Dominating Set, and k-Vertex Cover. G4SATBench provides Easy, Medium, and Hard difficulty levels with balanced SAT/UNSAT instances.

For clustering, we use all 1,400 Hard test instances: 200 from each problem type, evenly split between SAT and UNSAT, yielding 14 ground-truth (problem type, feasibility) groups. We focus on Hard for clarity; Easy and Medium show similar trends.

4.1 COMPARISONS

We compare the clustering quality obtained from the instance-level embeddings produced by our three transfer-learning variants introduced in Section 3: FORGE-MIP, FORGE-MIP-SAT, and FORGE-SAT. The first two variants reuse the pretrained FORGE (Shafi & Kadioglu, 2026) weights learned on MIP instances, while FORGE-SAT is pretrained from scratch on the SAT Competition Anniversary 2022 set, which contains 2827 instances.

Control Baseline: We include two additional baselines. STATIC-SAT uses the same SAT-specific clause and variable features described in FORGE-MIP-SAT setup in the Methodology section. Clause and variable features are mean-pooled at the instance level and concatenated to form a fixed, non-learned instance embedding. We also include FORGE-RANDOM, which is a randomly initialized GNN using the identical Forge architecture.

Evaluation Metrics: We evaluate clustering quality using *Normalized Mutual Information (NMI)* and *Purity*. NMI measures the agreement between predicted clusters and the 14 ground-truth categories, ranging from 0 to 1, with higher values indicating stronger alignment. Purity is computed per cluster as the fraction of its members belonging to its dominant ground-truth group, then averaged unweighted across the 14 clusters. We report mean $\pm$ standard deviation over repeated KMeans runs on fixed embeddings. These do not capture pretraining-seed variability.

4.2 CLUSTERING RESULTS

Figure 3 shows the instance-level embeddings projected into 2D using PaCMAP (Wang et al., 2021). Across methods, FORGE-SAT exhibits the clearest organization of the problem-type/SAT-status groups, with several groups forming compact and visually distinct regions. In some problem families, such as PS, SAT and UNSAT instances are also visibly separated, while other families, such as k-Clique, show substantial overlap.

The other methods also recover considerable problem structure. STATIC-SAT forms several coherent regions despite using only hand-designed structural features, while both FORGE-MIP and FORGE-MIP-SAT produce structured embeddings despite relying on weights pretrained entirely on MIP instances. FORGE-MIP-SAT generally yields cleaner organization than FORGE-MIP, suggesting that SAT-specific input features make the transferred MIP representation more compatible with SAT structure.

Table 1 reports the corresponding quantitative results. STATIC-SAT already provides a strong baseline, achieving an NMI of 0.74 compared with 0.54 for FORGE-RANDOM. Both MIP-pretrained variants improve on STATIC-SAT in NMI, with FORGE-MIP reaching 0.76 and FORGE-MIP-SAT reaching 0.77. FORGE-MIP-SAT also improves Purity over FORGE-MIP, increasing from 0.61 to 0.66. Although the gains over the strong STATIC-SAT baseline are modest, these results show that representations learned entirely from MIP data retain measurable structure when transferred to SAT, and that SAT-specific feature adaptation can further improve this transfer.

Pretraining directly on SAT yields the strongest NMI: FORGE-SAT reaches 0.81 NMI and 0.66 Purity, further improving the organization of unseen instances. At the same time, the embeddings primarily reflect problem-family structure; satisfiability is visibly separated for some distributions but is not a uniformly separable axis across all seven problem types.

To the best of our knowledge, this is the first study to directly transfer a MIP-pretrained foundational representation model to SAT. This provides evidence that pretrained optimization representations can retain useful structure on SAT without SAT-specific pretraining.

5 INTERPRETING EMBEDDINGS

Clustering shows that FORGE-SAT learns structured SAT representations, but not what its discrete codewords represent. We therefore adapt automated feature interpretation for sparse autoencoders (Paulo et al., 2025) to the FORGE-SAT codebook. Figure 4 summarizes the process: for each codeword, we construct examples that characterize its membership, infer a natural-language description from those examples, validate whether that description predicts membership on unseen nodes, and finally test whether the same interpretation transfers to Hard instances.

Table 1: NMI and Purity scores across methods on Hard G4SATBench test instances.

Metric	FORGE-RANDOM	STATIC-SAT	FORGE-MIP	FORGE-MIP-SAT	FORGE-SAT
NMI	0.54 ± 0.014	0.74 ± 0.002	0.76 ± 0.001	0.77 ± 0.003	$\mathbf{0.81 \pm 0.004}$
Purity	0.43 ± 0.006	0.63 ± 0.002	0.61 ± 0.002	$\mathbf{0.66 \pm 0.001}$	0.66 ± 0.009

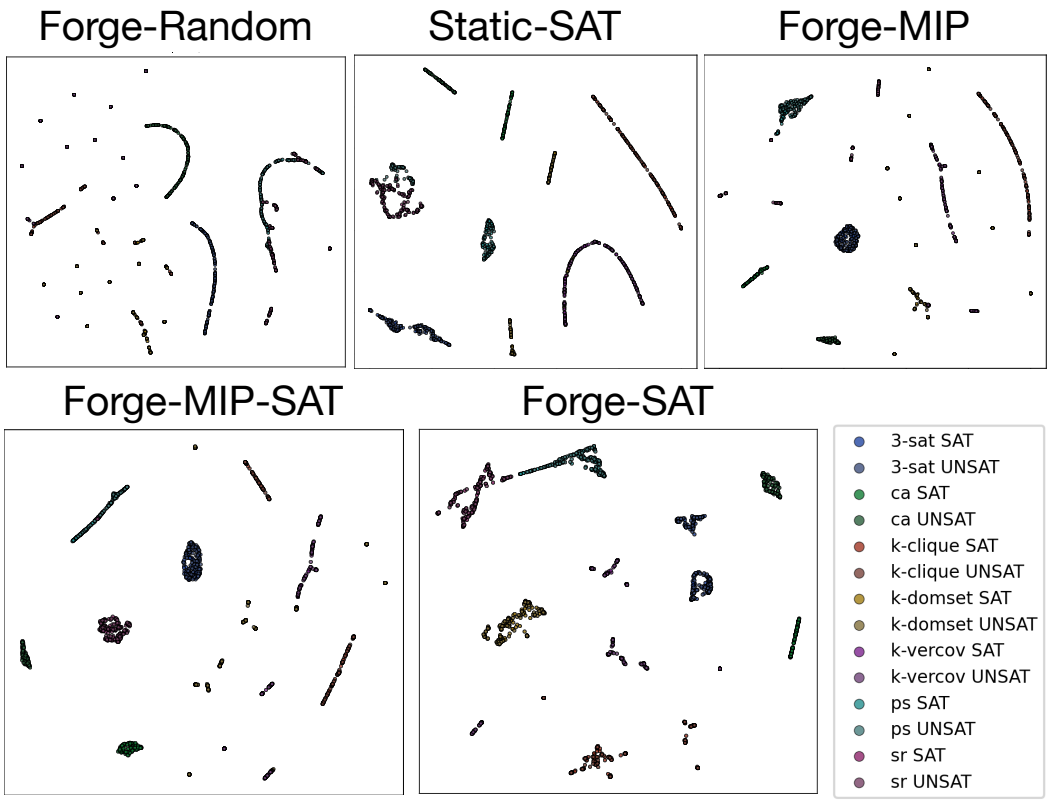

Figure 3: PaCMAP projections of Hard G4SATBench instance embeddings across baselines and FORGE variants. Colors indicate the 14 problem-type/SAT-status groups.

For each active codeword, we collect three types of examples from Easy training instances. Prototypes are the 30 assigned nodes with highest cosine similarity to the codeword. Borderline positives are 15 assigned nodes with the smallest positive margin over the runner-up codeword. Matched negatives are 15 same-type nodes assigned to another codeword but selected to be structurally similar to the prototypes using standardized, human-readable node and graph-context features. Descriptions are generated from nodes in Easy training graphs and validated on nodes from disjoint Easy validation graphs; Hard test graphs are also disjoint from both sets.

A domain-blind CLAUDE-OPUS-5 (Anthropic, 2026a) explainer uses these examples to infer a natural-language description of the structural pattern associated with the codeword. We then test that description on held-out Easy nodes using a separate CLAUDE-SONNET-5 (Anthropic, 2026b) validator, which receives only the description and the same human-readable structural features and predicts codeword membership. Neither model observes SAT/UNSAT labels or problem-family identities. The concise names shown in Table 2 summarize these descriptions for readability; all validation is performed using the full generated descriptions. We measure description quality using area under the receiver operating characteristic curve (AUROC) (Fawcett, 2006). As an additional check, we train simple structural probes on node-level features, graph-context features, and their combination to measure how directly codeword membership is recoverable from the observable features.

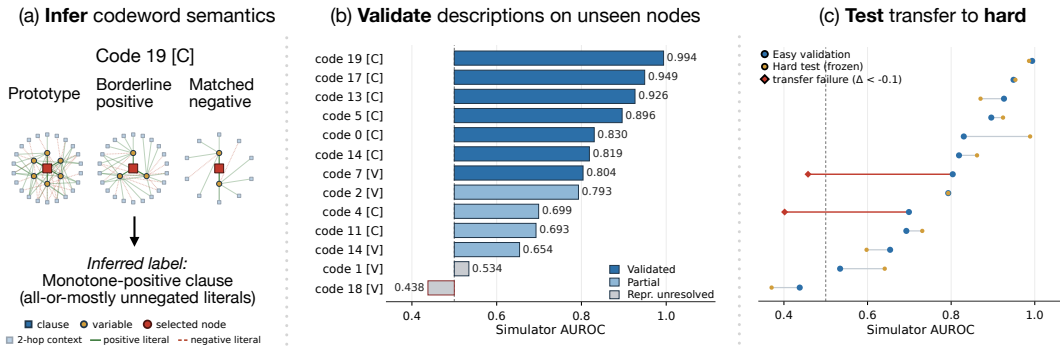

Figure 4: **Interpreting and validating FORGE-SAT codewords.** (a) An explainer infers a natural-language codeword description from prototypes, borderline positives, and matched negatives; a concise interpretation label is shown for readability. (b) The full description is validated by predicting codeword membership on unseen Easy nodes. (c) The same frozen description is evaluated on Hard nodes to measure semantic transfer.

Table 2: Interpreted FORGE-SAT codeword units with short-form names from the domain-blind explainer. Easy AUROC evaluates the description on held-out Easy nodes; Hard AUROC applies the same frozen description to Hard test nodes. Full descriptions are in Table 5. Reported AUROCs correspond to the canonical generated descriptions.

Code	Proposed name	AUROC	
		Easy	Hard
19-C	Monotone-positive clause (all-or-mostly unnegated literals)	0.994	0.986
17-C	Monotone-positive hub clause over sign-committed variables	0.949	0.954
13-C	Majority-positive 3-clause in a uniform-width graph (globally conditioned, locally unseparable)	0.926	0.870
5-C	Balanced mid-width clause anchored to a high-degree / degree-heterogeneous variable set	0.896	0.924
0-C	Hub-attached clause (touches a very high-degree variable, large two-hop footprint)	0.830	0.989
14-C	Negative-majority ternary clause in a high-degree, degree-heterogeneous region	0.819	0.862
7-V	Balanced high-degree variable in a low-expansion clause block	0.804	0.458
2-V	High-degree hub variable (bimodal: balanced-polarity in uniform clauses, or single-positive-occurrence in width-2 chains) — not separable from feature-matched negatives	0.793	0.793
4-C	Overlap-saturated clause (variables share nearly the same clause set)	0.699	0.402
11-C	All-negative (negation-monotone) clause	0.693	0.731
14-V	Catch-all mixed-polarity variable code (width-heterogeneous neighbourhood), with an extreme-degree hub tail	0.654	0.598
1-V	Balanced high-degree hub variable (polarity-symmetric, many-clause occurrence)	0.534	0.641
18-V	Balanced high-degree hub variable (mixed-width clause neighbourhood)	0.438	0.370

We use the following predictive validation criteria: codewords are treated as validated when both the description and combined structural probe achieve AUROC at least 0.80, partially interpreted when description AUROC is at least 0.60 but below 0.80, and representation unresolved when description AUROC is below 0.60 and the combined probe is below 0.80. Here, validation indicates that the inferred description predicts codeword membership; it does not imply that the codeword is determined by a single local structural feature, as some units may depend on broader graph context. Because codeword 14 is assigned to both clause and variable nodes, we evaluate its two node types separately. Finally, we freeze each Easy-derived description and evaluate it unchanged on Hard test nodes to test whether the inferred semantics remain stable under distribution shift.

On held-out Easy nodes, 7 of 13 codeword units meet our predictive validation criterion in the canonical run, 4 are partially interpreted, and 2 remain unresolved. Strong interpretations include monotone-positive clauses, hub-attached clauses, and majority-positive ternary clauses, while codes 1 and 18 remain representation-unresolved. The Easy-derived descriptions often transfer to Hard

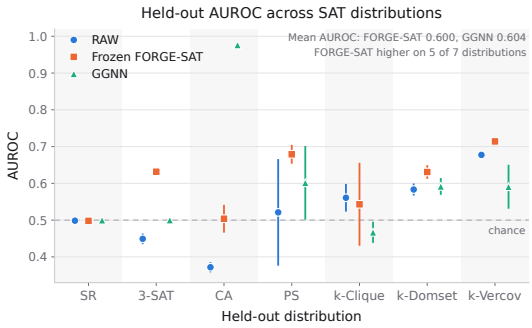

Figure 5: Held-out test satisfiability prediction across SAT distributions. We compare RAW structural features, frozen FORGE-SAT representations, and a task-trained GGNN over three seeds. FORGE-SAT and GGNN achieve similar mean AUROC (0.600 vs. 0.604), with FORGE-SAT higher on 5 of 7 distributions; the dashed line denotes chance.

instances: median AUROC changes from 0.804 to 0.793, with 8 of 13 units remaining within 0.05 of their Easy performance or improving. However, codes 7 and 4 fall from 0.804 to 0.458 and from 0.699 to 0.402, respectively, showing that not all learned semantics remain stable under distribution shift. Because description generation is stochastic, units near the validation thresholds may vary across repeated generations. Some codewords depend on broader graph context rather than a single local property; however, regenerating six affected descriptions without graph identifiers changes Easy AUROC by only 0.013 on average (Appendix A.2).

6 DOWNSTREAM APPLICATION: SUPERVISED SAT PREDICTION

We next ask whether the learned SAT representations provide reusable signal for a downstream task without updating the pretrained encoder. We consider satisfiability prediction and evaluate whether a lightweight supervised MLP trained on frozen FORGE-SAT representations can generalize to an unseen SAT distribution. For each leave-one-domain-out split, we train on Easy instances from six G4SATBench problem distributions and evaluate on the held-out seventh distribution. We repeat this procedure for all seven problem types and report AUROC over three random seeds. Hyperparameters and model selection use only the training and validation splits; the reported held-out-domain AUROC is computed on the untouched test split. For each fold, the prediction head is refit using the selected hyperparameters and seed before a single evaluation on the held-out test data. For FORGE-SAT, the encoder remains frozen throughout. We form a graph-level representation by pooling the pre-quantization node embeddings over all nodes, variable nodes, and clause nodes, and concatenate these pooled representations as input to a lightweight supervised MLP. Only this prediction head is trained using SAT/UNSAT labels, isolating the utility of the pretrained representation from task-specific representation learning.

We compare three approaches. **RAW** is a non-learned representation built from the same SAT-specific node features as STATIC-SAT, retaining all-node, clause, and variable means plus three instance-size descriptors ($\log n_{\text{var}}$, $\log n_{\text{cls}}$, $n_{\text{cls}}/n_{\text{var}}$). It uses the same MLP architecture as frozen FORGE-SAT, so the comparison isolates the input representation. Frozen FORGE-SAT uses the frozen graph-level representation described above as input to a lightweight MLP prediction head. **GGNN** is a task-trained graph neural network based on the G4SATBench setup Li et al. (2024), trained end-to-end for satisfiability prediction under the same leave-one-domain-out protocol. Together, these comparisons distinguish the value of pretrained SAT representations from both hand-designed structural features and end-to-end supervised graph representation learning.

Figure 5 shows held-out test AUROC across the seven G4SATBench distributions. Averaged across domains, frozen FORGE-SAT and GGNN achieve similar AUROC of 0.600 and 0.604, respectively, while RAW reaches 0.523. FORGE-SAT improves over RAW on 5 of 7 held-out distributions and exceeds GGNN on 5: 3-SAT, PS, k-Clique, k-Dominating Set, and k-Vertex Cover. The aggregate comparison is strongly influenced by CA, where GGNN substantially outperforms FORGE-SAT (0.976 vs. 0.504). On test instances from the six distributions seen during training, FORGE-

SAT achieves 0.777 AUROC, compared with 0.750 for RAW and 0.760 for GGNN. Overall, frozen FORGE-SAT provides reusable predictive signal, with similar mean held-out AUROC to GGNN despite substantially different per-domain behavior.

7 DISCUSSION AND LIMITATIONS

Our results suggest that shared bipartite constraint–variable structure enables transfer from MIP to SAT, while the improvement from FORGE-MIP to FORGE-MIP-SAT shows that domain-specific node features remain important.

Our first limitation is that our study focuses on SAT as a single decision-problem domain; it remains unclear how broadly these findings extend to other constraint satisfaction problems. Second, our SAT node features are simple statistical descriptors rather than learned or heavily engineered features, and a more systematic feature ablation could identify which properties are most important for transfer and representation quality. Third, FORGE-SAT is pretrained on a relatively small SAT corpus, and larger and more diverse collections may yield stronger and more robust representations. Finally, although the learned codeword interpretations often transfer from Easy to Hard instances, several codewords show substantial degradation, indicating that not all learned semantics remain stable under distribution shift.

Future work could scale SAT pretraining, extend the representations to additional downstream tasks such as assignment prediction, UNSAT-core prediction, and solver-guided heuristics, and study whether the same framework transfers to broader classes of constraint satisfaction problems.

8 RELATED WORK

Graph neural networks are widely used for learning SAT representations. NeuroSAT (Selsam et al., 2019) showed that message-passing networks trained for satisfiability can learn useful search behavior, motivating many solver-integrated approaches. NLocalSAT (Zhang et al., 2020), NeuroGlue (Han, 2020), and Graph-Q-SAT (Kurin et al., 2020) use neural predictions to guide initialization, glue-variable selection, and branching, while NeuroCore (Selsam & Bjørner, 2019), NeuroBack (Wang et al., 2024), and Chen et al. (Chen & Wang, 2025) move neural computation offline to guide core prediction, phase selection, or preprocessing. *Across these methods, representations are typically learned for a specific prediction objective or solver-facing task.* A parallel line studies more general SAT representations. Cameron et al. (2020) learn permutation-invariant representations for satisfiability prediction, while Duan et al. (2022) use contrastive pretraining with CDCL-augmented instances. GraSS (Zhang et al., 2024) learns supervised graph embeddings for solver selection. Cardillo et al. (2025) encode k-CNF formulas as MILP-style weighted bipartite graphs and study both satisfiability prediction and theoretical properties of this representation. More recently, Cameron et al. (2025) propose a foundation model for Boolean logic trained across multiple logical tasks and study transfer to held-out tasks. Recent work on SAT GNN interpretability analyzes latent representations and MaxSAT relaxations (Mojžíšek et al., 2026), as well as SAT-relevant algorithmic concepts emerging in NeuroSAT (Shoham et al., 2026).

Unlike Cameron et al. (2025), which pretrain directly across Boolean tasks, we ask whether an optimization-pretrained representation framework can transfer to SAT without SAT-specific supervision. We study MIP-pretrained transfer, SAT-specific feature adaptation, SAT-native pretraining, codeword interpretability, and frozen downstream representations under held-out distribution shift.

9 CONCLUSION

In this paper, we show that a structural representation-learning framework developed for optimization can extend to decision problems such as SAT. It would be interesting to explore how these ideas extend to Constraint Satisfaction Problems (CSPs) and Constraint Programming (CP). It remains an open question whether a single foundational pretrained model can serve all of these paradigms. However, our results show that the same foundational architecture can support both MIP-pretrained transfer to SAT and SAT-native pretraining. This opens an exciting direction toward foundational models that unify optimization and satisfaction and support both supervised and unsupervised tasks.

AI USE STATEMENT

In this work, we used generative AI tools for research ideation and experiment execution, retrieval and discovery of related literature, drafting and polishing portions of the manuscript, and checking code and experimental outputs. We did not use AI to generate pretraining data since that was already available online. We reviewed AI-assisted work by testing outputs across multiple agents. We also manually checked that code matched our proposed pipeline and confirmed that the cited papers actually exist. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses publicly available SAT benchmark datasets and does not involve human subjects, personal data, or sensitive attributes. We do not identify any specific ethical risks arising directly from the methods or experiments presented in this work.

REPRODUCIBILITY STATEMENT

The paper and appendix describe the datasets, model variants, training setup, evaluation protocols, and metrics used in our experiments. Code, trained models, experiment configurations, and scripts supporting the reported results are provided in the anonymized repository: `https://anonymous.4open.science/r/forge-sat-release-057C/README.md`

REFERENCES

Anthropic. Introducing claude opus 5, July 2026a. URL `https://www.anthropic.com/news/claude-opus-5`.

Anthropic. Introducing claude sonnet 5, June 2026b. URL `https://www.anthropic.com/news/claude-sonnet-5`.

Chris Cameron, Rex Chen, Jason Hartford, and Kevin Leyton-Brown. Predicting propositional satisfiability via end-to-end learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3324–3331, 2020.

Chris Cameron, Xing Jin, Taylor Lundy, Jason Hartford, and Kevin Leyton-Brown. Foundation models for boolean logic, 2025. URL `https://openreview.net/forum?id=qeY25DwmKO`.

Franco Alberto Cardillo, Hamza Khyari, and Umberto Straccia. Milp-sat-gnn: Yet another neural sat solver, 2025. URL `https://arxiv.org/abs/2507.01825`.

Guanting Chen and Jia Wang. Enhancing modern sat solver with machine learning method. In *Proceedings of the Great Lakes Symposium on VLSI 2025*, GLSVLSI '25, pp. 886–892, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714962. doi: 10.1145/3716368.3735251. URL `https://doi.org/10.1145/3716368.3735251`.

Haonan Duan, Pashootan Vaezipoor, Max B Paulus, Yangjun Ruan, and Chris Maddison. Augment with care: Contrastive learning for combinatorial problems. In *International Conference on Machine Learning*, pp. 5627–5642. PMLR, 2022.

Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Neural Information Processing Systems*, 32:15554–15566, 2019.

Jesús Giráldez-Cru and Jordi Levy. A modularity-based random sat instances generator. 2015.

Jesús Giráldez-Cru and Jordi Levy. Locality in random sat instances. In *IJCAI*, volume 17, pp. 638–644, 2017.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Neural Information Processing Systems*, pp. 1024–1034, 2017.

Jesse Michael Han. Enhancing sat solvers with glue variable predictions, 2020. URL https://arxiv.org/abs/2007.02559.

Ashlin Iser, Marijn Heule, Matti Järvisalo, Martin Suda, and Tomas Balyo. Benchmarks of sat competition 2022: Anniversary track, May 2024. URL https://doi.org/10.5281/zenodo.11175170.

Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Can q-learning with graph networks learn a generalizable branching heuristic for a sat solver? In *Advances in Neural Information Processing Systems 32*, 2020.

Massimo Lauria, Jan Elffers, Jakob Nordström, and Marc Vinyals. Cnfgen: A generator of crafted benchmarks. In *International Conference on Theory and Applications of Satisfiability Testing*, pp. 464–473. Springer, 2017.

Zhaoyu Li, Jinpei Guo, and Xujie Si. G4SATBench: Benchmarking and advancing SAT solving with graph neural networks. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=7VB5db72lr.

David Mojžíšek, Jan Hůla, Ziwei Li, Ziyu Zhou, and Mikoláš Janota. Neural approaches to sat solving: Design choices and interpretability. *International Journal of Approximate Reasoning*, 190: 109609, 2026. ISSN 0888-613X. doi: https://doi.org/10.1016/j.ijar.2025.109609. URL https://www.sciencedirect.com/science/article/pii/S0888613X25002506.

Gonçalo Santos Paulo, Alex Troy Mallen, Caden Juang, and Nora Belrose. Automatically interpreting millions of features in large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=EemtbhJOXc.

Daniel Selsam and Nikolaj Bjørner. Guiding high-performance sat solvers with unsat-core predictions. In Mikoláš Janota and Inês Lynce (eds.), *Theory and Applications of Satisfiability Testing – SAT 2019*, pp. 336–353, Cham, 2019. Springer International Publishing. ISBN 978-3-030-24258-9.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJMC_iA5tm.

Zohair Shafi and Serdar Kadioglu. Forge: Foundational optimization representations from graph embeddings, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=7Uo1yRWUpo.

Elad Shoham, Hadar Cohen, Khalil Wattad, Havana Rika, and Dan Vilenchik. Concept learning for algorithmic reasoning: Insights from sat-solving gnns. *Information Sciences*, 726:122754, 2026. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2025.122754. URL https://www.sciencedirect.com/science/article/pii/S0020025525008904.

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

Yingfan Wang, Haiyang Huang, Cynthia Rudin, and Yaron Shaposhnik. Understanding how dimension reduction tools work: an empirical approach to deciphering t-sne, umap, trimap, and pacmap for data visualization. *Journal of Machine Learning Research*, 22(201):1–73, 2021.

Ling Yang, Ye Tian, Minkai Xu, Zhongyi Liu, Shenda Hong, Wei Qu, Wentao Zhang, Bin Cui, Muhan Zhang, and Jure Leskovec. Vqgraph: Rethinking graph representation space for bridging gnns and mlps. In *International Conference on Learning Representations*, 2024.

Wenjie Zhang, Zeyu Sun, Qihao Zhu, Ge Li, Shaowei Cai, Yingfei Xiong, and Lu Zhang. Nlocalsat: Boosting local search with solution prediction. In Christian Bessiere (ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 1177–1183. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/164. URL https://doi.org/10.24963/ijcai.2020/164. Main track.

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*, pp. 6301–6311, 2024.

A Additional Experimental Details

A.1 Downstream evaluation details

GGNN is evaluated here under leave-one-distribution-out shift, rather than the in-distribution setting of the original benchmark. For each of the seven G4SATBench Easy distributions in turn, the model is trained on the remaining six and evaluated on the held-out distribution; results are averaged over three seeds (123, 124, 125). This differs from the benchmark's standard protocol, where train, validation, and test instances are drawn from the same generator, so the resulting performance numbers are not directly comparable.

Table 3: Held-out test AUROC under leave-one-distribution-out evaluation. Results are mean ± standard deviation over three seeds.

Held-out distribution	RAW	Frozen FORGE-SAT	GGNN
SR	0.4985 ± 0.0014	0.4980 ± 0.0004	0.4997 ± 0.0064
3-SAT	0.4491 ± 0.0146	$\mathbf{0.6316 \pm 0.0097}$	0.5000 ± 0.0000
CA	0.3719 ± 0.0139	0.5038 ± 0.0378	$\mathbf{0.9763 \pm 0.0041}$
PS	0.5211 ± 0.1449	$\mathbf{0.6791 \pm 0.0257}$	0.6016 ± 0.1002
k-Clique	$\mathbf{0.5608 \pm 0.0376}$	0.5430 ± 0.1128	0.4668 ± 0.0287
k-Dominating Set	0.5832 ± 0.0164	$\mathbf{0.6309 \pm 0.0187}$	0.5916 ± 0.0229
k-Vertex Cover	0.6774 ± 0.0033	$\mathbf{0.7141 \pm 0.0055}$	0.5908 ± 0.0598
Mean over 7 held-out	0.5231 ± 0.0252	0.6001 ± 0.0197	$\mathbf{0.6038 \pm 0.0088}$

Under distribution shift, GGNN reaches 0.6038 ± 0.0088 AUROC averaged across the seven held-out distributions, compared with 0.6001 ± 0.0197 for frozen FORGE-SAT representations and 0.5231 ± 0.0252 for RAW features. Performance varies substantially by held-out distribution: GGNN reaches 0.9763 ± 0.0041 on CA but is at or below chance on 3-SAT (0.5000 ± 0.0000) and k-Clique (0.4668 ± 0.0287), while frozen FORGE-SAT is above chance on six of seven distributions.

Table 4: Test AUROC on distributions represented during training. Results are mean ± standard deviation over three seeds.

	RAW	Frozen FORGE-SAT	GGNN
Seen-distribution test	0.7503 ± 0.0025	$\mathbf{0.7769 \pm 0.0011}$	0.7600 ± 0.0308

On test instances from distributions represented during training, RAW, frozen FORGE-SAT, and GGNN reach 0.7503 ± 0.0025, 0.7769 ± 0.0011, and 0.7600 ± 0.0308 AUROC, respectively, indicating that GGNN's weak performance on several held-out distributions occurs specifically in the cross-distribution evaluation setting. The reported GGNN results use the same configuration across all seed runs; full hyperparameters and checkpoint-selection details are provided with the anonymized repository.

A.2 Interpretation split and robustness to explainer identifiers

Interpretation examples are drawn exclusively from Easy training graphs, while validation uses disjoint Easy validation graphs and Hard evaluation uses graphs disjoint from both. Example graph identifiers provided to the explainer are anonymized independently within each codeword and have no correspondence to validation identifiers. As a robustness check, for six descriptions containing graph references we compared three generations using the grouped identifiers with three generations in which all graph-grouping information was removed. The mean Easy-AUROC difference was 0.013, compared with a mean within-condition standard deviation of 0.045; five of six units had overlapping score ranges. Removing graph identifiers therefore did not materially affect predictive validation, although generated descriptions themselves vary across runs.

B FULL CODEWORD INTERPRETATIONS

Table 5: Canonical descriptions for all 13 interpreted FORGE-SAT codeword units. Descriptions are the verbatim output of the domain-blind explainer. Easy AUROC is the held-out simulator score on Easy validation nodes; Hard AUROC applies the same description, frozen, to Hard test nodes. Type C denotes a clause unit and V a variable unit; code 14 is reported separately for each node type.

Code	Type	Description	Easy AUROC	Hard AUROC
code 19	C	The dominant, unmistakable axis of this code is literal polarity of the clause itself: essentially every prototypical positive is fully monotone positive (pos_frac = 1.0, neg = 0), and the borderline positives are the polarity-diluted cases (pos_frac 0.667-0.857, i.e. one or two negated literals out of 3-7). Width is not the trigger — the code fires at widths 2 through 7 — and neighbourhood degree ranges over two orders of magnitude (nbr_deg_mean 6.5 to 67.7, 2hop 10 to 271), so neither local degree nor two-hop reach is what is being encoded. The code appears to be a clause-level polarity detector: 'this clause imposes only positive requirements on its variables'. The borderline set shows the code's boundary is graded in pos_frac: as soon as the clause drops to roughly 2/3 positive it is on the verge of being routed elsewhere, and clauses whose variables themselves are polarity-balanced (nbr_posfrac ˜0.46-0.55) are the marginal ones, while the safest positives are those whose variables are also predominantly positive elsewhere (nbr_posfrac 0.87-0.90).	0.994	0.986
code 17	C	Every prototypical positive is a clause with pos_frac = 1.0 (no negative literals at all) that is also wide for its graph (width 8-15) and sits in a very large two-hop neighbourhood (142-523 distinct co-clauses, versus ˜96-153 for the feature-matched negatives). Crucially, its variables are not polarity-balanced: nbr_posfrac is either near 0 (0.023-0.045) or near 1 (0.91-0.92), never ˜0.5. In the low case the arithmetic is telling — a variable of degree ˜40 with positive fraction ˜0.025 has essentially exactly one positive occurrence, namely in this clause, while all its other occurrences are negative. So the motif is: a wide all-positive clause that supplies the *only* (or nearly only) positive occurrence of each of a large set of otherwise-negatively-used, high-degree variables. The mirror case (G13/G14) is the same idea with the surrounding polarity flipped: an all-positive clause whose variables are also almost exclusively positive elsewhere. Either way the defining signal is a monotone clause embedded in a polarity-pure variable neighbourhood, plus uniformly high and tight neighbour degrees (nbr_deg_min is close to nbr_deg_max, nbr_deg_std small relative to the mean).	0.949	0.954

continued on next page

Table 5: *(continued)*

Code	Type	Description	Easy AUROC	Hard AUROC
code 13	C	Every prototypical positive is an exact structural clone: width 3, two positive and one negative literal (pos_frac = 0.667), variable degrees in the mid-teens (mean ˜14-19, min ˜7-19, max ˜14-21), 2hop in the 25-44 range, and neighbour positive-occurrence fractions tightly around 0.5. So on its face the code names the motif 'width-3 clause with a 2:1 positive:negative split embedded in a homogeneous, moderately dense, polarity-balanced neighbourhood'. The problem is that the feature-matched negatives are indistinguishable on every one of those columns (same width, same 2/1 split, nbr_deg_mean 15-18, 2hop 34-47, nbr_posfrac 0.46-0.59). The only visible regularity is provenance: all 30 prototypical positives come from graphs G00-G08, while all 15 negatives come from G11-G31, and the borderline positives are much wider clauses (width 6-12, degrees 24-42, 2hop 68-152) drawn from that same high-index group. That pattern is what you would expect if the code's decision boundary is set partly by graph-level context (overall clause-width distribution and degree scale) rather than by the clause's own local features: in graphs where width-3 is the uniform width, the 2:1 three-clause lands here; in graphs containing a mix of narrow and wide clauses, the cell is instead occupied by mid-to-wide clauses with roughly balanced polarity (pos_frac 0.33-0.73, mostly 0.44-0.60), and the width-3 clauses there are routed elsewhere.	0.926	0.870
code 5	C	On every feature displayed, the prototypical positives and the contrastive negatives occupy essentially the same region: width 5-6, polarity as balanced as the width allows (3/3, 2/3, 3/2, pos_frac 0.4-0.6), neighbour variables of mean degree ˜26-40, 2hop neighbourhood ˜70-140, and nbr_posfrac clustered tightly around 0.45-0.53 (i.e. the variables the clause touches are themselves polarity-neutral overall). So the code is clearly *situated* in the 'wide, polarity-balanced clause in a dense, polarity-neutral formula' region, but that description alone does not separate it from the negatives. The only systematic, if small, differences are in the degree profile of the clause's own variables: positives run a little higher in nbr_deg_mean (˜31 vs ˜29) and, more noticeably, in nbr_deg_max and nbr_deg_std — several positives contain a variable of degree 44-51 (G18 group) while no matched negative exceeds 39, and the positive set also contains the largest 2hop values (132, 138). The borderline positives reinforce a 'degree-heterogeneity' reading rather than a 'width' reading: they are width-2 balanced clauses in much sparser graphs, but every one of them pairs one low-degree variable with a markedly higher-degree one (nbr_deg_std 5-9.5, min 5-8 vs max 16-25). The common thread across prototypes and borderlines is therefore: perfectly (or near-perfectly) balanced polarity, plus at least one variable that is a hub relative to its clause-mates, with the clause's variables being individually polarity-neutral. Caveat: the positives and negatives also split cleanly by graph id (positives from G17/18/24/26-29, negatives from G13-16/19-23/25/30), so much of what the code responds to may be a graph-level context signal that the node-level columns shown here do not expose.	0.896	0.924

continued on next page

Table 5: *(continued)*

Code	Type	Description	Easy AUROC	Hard AUROC
code 0	C	The prototypes are all width-2, purely negative clauses (pos=0, pos_frac=0) whose two variables include at least one very high-degree hub: nbr_deg_max is 49-57 in every prototype, nbr_deg_mean 31-53, and the two-hop clause neighbourhood spans 60-104 distinct clauses. The contrastive negatives are structurally identical on the obvious features - also width 2, also 0 positive / 2 negative literals, also variables that occur overwhelmingly negated - but they live in a flat, degree-homogeneous neighbourhood: every neighbour variable has degree 9-12 (nbr_deg_max = 12 for all of them, std ¡= 2) and the two-hop reach is only 18-22 clauses. So the code is not keying on width or on polarity (both groups are all-negative binary clauses); it is keying on the magnitude and skew of the neighbour degree distribution, i.e. on whether the clause is attached to a hub variable and therefore has a large, degree-asymmetric two-hop context. The borderline positives confirm this reading in the direction that matters: they abandon the all-negative, width-2 signature entirely (widths 2-8, pos_frac 0-0.5, nbr_posfrac ˜0.46-0.53) yet still carry elevated neighbour degrees (mean 11.5-33.4, max 16-42) and mostly large two-hop counts (20-125). Polarity is clearly the secondary, softer axis - among the prototypes the neighbour variables are almost purely negative-occurring (nbr_posfrac 0.019-0.049) versus 0.083-0.104 for the matched negatives - but that gap is tiny next to the 3-5x gap in neighbour degree.	0.830	0.989
code 14	C	Every prototypical positive is a width-3 clause with exactly one positive and two negative literals (pos_frac = 1/3), so the code is anchored on the 1-pos/2-neg triple motif. But the contrastive negatives have *identical* width and polarity, so polarity alone cannot be what separates them. The separating signal is the neighbourhood degree context: positives sit on variables that occur in many clauses (nbr_deg_mean typically 30-90, max frequently 50-100, nbr_deg_std often 10-30, 2hop 75-180), whereas the feature-matched negatives are uniformly in a tight, homogeneous low-degree regime (nbr_deg_mean 21-30, min˜19-24, max˜25-37, std ˜1.5-5, 2hop ˜52-77). So the code reads as: a negative-leaning short clause attached to at least one high-occurrence / hub-like variable, i.e. embedded in a locally dense and degree-heterogeneous part of the incidence graph. The borderline positives relax the width/polarity requirement (widths 3-8, pos_frac 0 to 0.75) while keeping the same 'moderately dense two-hop context' signature, which suggests degree context dominates the code vector and polarity/width only fine-tune the position within the cell.	0.819	0.862

continued on next page

Table 5: *(continued)*

Code	Type	Description	Easy AUROC	Hard AUROC
code 7	V	Every prototypical positive is a variable with unusually high degree (16-63 occurrences, far above what a typical variable in these graphs appears to have), an almost exactly even split of positive and negative occurrences (pos_frac clustered in 0.42-0.58, mode 0.50), and neighbour clauses of near-uniform width (nbr_w_std = 0 for the majority, width 4 or 5). The most striking feature is the ratio degree/2hop: a variable occurring in 17 clauses touches only 3-6 distinct other variables, and one with degree 63 touches only 14. So the clauses containing this variable are drawn from a tiny, repeatedly reused variable pool — a closed, highly redundant block of clauses rather than a spread-out neighbourhood. The neighbour clauses are themselves polarity-balanced (nbr_posfrac ˜0.45-0.54). The code therefore looks like a detector for 'hub variable inside a small, polarity-symmetric, uniform-width clause cluster with almost no outward expansion'. The borderline positives relax exactly the parts you would expect: they either keep the balance but have mixed clause widths (nbr_w_std 3-13, wide max-width clauses) or skew polarity (pos_frac 0.62-0.67), i.e. they sit at the edge of the balance / uniform-width region.	0.804	0.458
code 2	V	Every prototypical positive is a high-degree variable (degree 25–43, far above what a typical variable would have) and they fall into two sharply distinct regimes. Regime A: degree ˜26–33, almost exactly balanced polarity (pos_frac 0.43–0.55), all incident clauses of identical width (4 or 5, nbr_w_std = 0), tiny two-hop neighbourhood (2hop = 4–16, i.e. far smaller than degree$\times$(width-1)), and neighbour clauses that are themselves polarity-balanced (nbr_posfrac ≈ 0.49). Regime B: degree ˜27–43, one single positive occurrence and all the rest negative (pos_frac $\approx$ 0.02–0.04), incident clauses almost all of width 2 with one wide clause (nbr_w_mean ≈ 2.3, nbr_w_max 10–15), and 2hop $\approx$ degree-1, i.e. a star/implication-hub whose binary clauses each introduce a fresh distinct variable. Borderline positives extend regime A to much higher degrees (up to 81) and to mixed-width neighbourhoods, suggesting the code's core is simply 'very high degree variable' with the two regimes sharing that trait. Critically, the contrastive negatives are numerically identical to the positives on every feature shown (e.g. G15:n1575 matches G19:n949 exactly; G02:n203 matches G11:n222 exactly). On the evidence available I cannot state any property that separates code 2 from the competing codes; the visible signal is only the high-degree hub character, which the negatives share.	0.793	0.793

continued on next page

Table 5: *(continued)*

Code	Type	Description	Easy AUROC	Hard AUROC
code 4	C	Within the feature-matched comparison, width (5), polarity split (2 pos / 3 neg, pos_frac 0.40) and mean neighbour degree (˜23-33) are essentially identical for positives and negatives, so those cannot be what the code encodes. The one quantity that separates them cleanly is the two-hop count: prototypical positives have 2hop of 27-36 while their five variables each sit in ˜26-31 clauses (sum of neighbour degrees ˜130-155), i.e. the distinct-clause neighbourhood is only about 1.0-1.2x the mean neighbour degree — the five variables occur in almost exactly the same set of clauses. Contrastive negatives, with the same degrees, reach 2hop of 51-69, roughly 2x the mean neighbour degree, i.e. their variables' clause sets overlap far less. Positives also have very low nbr_deg_std (0.49-1.20) versus 1.5-3.7 for negatives, so their variables are near-degree-regular as well. The code therefore appears to fire on a clause whose variable neighbourhood is maximally redundant/nested: a tightly closed, degree-homogeneous block where reaching two hops away buys almost no new clauses. The borderline members are width-2 clauses whose absolute 2hop is likewise small (12-32); they share the 'small, contained two-hop footprint' property but not the degree homogeneity, which is presumably why they sit near a decision boundary.	0.699	0.402
code 11	C	Every prototypical positive is a width-3 clause whose literals are all negated (pos=0, pos_frac=0.00, exactly 3 negatives). Neighbourhood statistics (variable degrees ˜11-18, 2hop ˜28-45, neighbour positive-fraction ˜0.31-0.55) span a wide range and are clearly not what the code is selecting on: the code fires across that whole range. The borderline positives extend the same motif to wider clauses (width 7-13) that remain strongly negative-dominant (pos_frac 0.13-0.36, i.e. at most a couple of positive literals among many negatives), which suggests the underlying axis is 'clause polarity strongly skewed toward negated occurrences', with the cleanest exemplar being the fully negative 3-clause. Importantly, the contrastive negatives are also all-negative width-3 clauses with statistically indistinguishable degree/2hop/nbr_posfrac profiles, so within the features shown I cannot separate positives from negatives at all; the residual discrimination must come from context not exposed here (the negatives all come from a different block of graphs, G27-G52, than the prototypical positives, G00-G26, hinting at a graph-level scale or degree-distribution effect that shifts identical local motifs to a neighbouring code).	0.693	0.731

continued on next page

Table 5: *(continued)*

Code	Type	Description	Easy AUROC	Hard AUROC
code 14	V	On everything visible here the prototypical positives and the feature-matched negatives are essentially the same population: degrees mostly 4-40, pos_frac clustered near 0.2-0.7 (median ˜0.5), clause widths ranging from a floor of 3 up to one very wide clause (nbr_w_max 12-39, large nbr_w_std), and 2hop counts that are close to nbr_w_max, i.e. most of their two-hop neighbourhood comes from the single widest clause they sit in. nbr_posfrac is ˜0.4-0.56 for both groups, so the code is not tracking the polarity climate of its clauses either. The only visible asymmetries are at the tails: (a) the positives include several very-high-degree hubs (151, 157, 211 occurrences) with near-balanced polarity, which do not appear among the matched negatives (max 37); and (b) the borderline positives are a quite different, internally near-identical family - degree 13-20, pos_frac 0.05-0.08 (almost purely negative occurrences), narrow clauses (width 2-5, nbr_w_mean ˜2.2), and low nbr_posfrac ˜0.07-0.10. That suggests code 14 behaves like a broad region of the codebook covering 'ordinary' mixed-polarity variables embedded in a width-heterogeneous clause neighbourhood, with its boundary abutting a distinct code for near-monotone-negative variables in narrow clauses, and absorbing degree outliers at the other end.	0.654	0.598
code 1	V	Every node carrying this code is a high-occurrence variable: degrees run from ˜10 to ˜64 (median around 28-30), far above the typical variable in these graphs, and the positive/negative split is almost always near even (pos_frac concentrated in 0.40-0.55, only two prototypes outside 0.37-0.65). The clauses they sit in have ordinary positive-literal fractions (nbr_posfrac ˜0.44-0.60, i.e. no polarity bias in their neighbourhood either). The code therefore looks like a detector for 'hub variable with symmetric polarity usage embedded in polarity-neutral clauses' rather than for anything about clause width per se: both narrow-uniform (width 3-5, std 0) and wide-mixed (nbr_w_mean 5-7, nbr_w_max up to 25, std 2-5) neighbourhoods occur among positives. Crucially, the feature-matched negatives reproduce the positives almost exactly on every column shown - G02:n67 (14, 7/7, w=3, 2hop 12, nbr_posfrac 0.548) is a negative while G02:n70 with identical numbers is a positive; G16:n202 (35, 16/19, w=5) is negative while G12:n262 with the same vector is positive. So on the observable quantities I cannot separate the two sets. The one asymmetry is that all 15 contrastive negatives have nbr_w_std = 0 (single-width neighbourhoods), whereas a substantial share of prototypes have mixed-width neighbourhoods (std 1.1-5.2); no feature-matched negative could be produced for those, which weakly suggests that mixed-width hub variables are captured by this code exclusively, while among uniform-width formulas the code competes with at least one other code on grounds not visible here (likely finer two-hop polarity structure or clause-overlap pattern).	0.534	0.641

continued on next page

Table 5: *(continued)*

Code	Type	Description	Easy AUROC	Hard AUROC
code 18	V	Every positive is a variable of unusually high degree (10–67 clause occurrences, most 15–35) whose literal polarity is close to balanced: pos_frac is concentrated in 0.33–0.65 with a median near 0.50, and pos/neg counts differ typically by only 1–4 even at degree 30+. The clause neighbourhoods are also polarity-neutral on average (nbr_posfrac ˜0.41–0.64, clustering near 0.5), i.e. the variable sits inside clauses that are themselves not polarity-skewed. A large share of the prototypes additionally have heterogeneous neighbour widths (nbr_w_mean 4.5–7.6, min 2, max 9–26, std 1.7–5.8), so the code covers hubs that bridge short and long clauses. The 2hop count (9–34) is modest relative to degree, meaning the same neighbour variables recur across many of the hub's clauses — a dense, locally re-entrant hub rather than a hub spread over disjoint clause groups. So the motif is: a heavily-occurring variable, polarity-symmetric, embedded in a dense and width-diverse clause cluster.	0.438	0.370