GraphBench: Next-generation Graph Learning Benchmarking

Timo Stoll [1] **Chendi Qian** [1] **Ben Finkelshtein** [2] **Ali Parviz** [3] **Darius Weber** [1] **Fabrizio Frasca** [4] **Hadar Shavit** [1] **Antoine Siraudin** [1] **Arman Mielke** [5 6] **Erik Müller** [1] **Marie Anastacio** [1] **Maya Bechler-Speicher** [7] **Michael Bronstein** [2 8] **Mikhail Galkin** [9] **Holger Hoos** [1] **Mathias Niepert** [6] **Bryan Perozzi** [9] **Jan Tönshoff** [10] **Christopher Morris** [10]

Abstract

Machine learning on graphs has made substantial progress across domains such as molecular property prediction and chip design. Yet benchmarking practices remain fragmented, often relying on narrow, task-specific datasets and inconsistent evaluation protocols, hindering reproducibility and broader progress. With the recent popularity of graph foundation models, these weaknesses have become apparent, as existing benchmarks are insufficient for thorough evaluation. To address these challenges, we introduce GRAPHBENCH, a comprehensive benchmark suite spanning diverse real-world domains and task settings, including node-level, edge-level, graph-level, and generative tasks. GRAPHBENCH provides standardized evaluation protocols, including consistent dataset splits and metrics for assessing out-of-distribution generalization on selected tasks, as well as a unified hyperparameter tuning framework. We further evaluate GRAPHBENCH with recent message-passing neural networks and graph transformer models, establishing principled baselines for future research.

1. Introduction

Machine learning on graphs is central to many applications, including drug design (Wong et al., 2023), recommender systems (Ying et al., 2018), chip design (Zheng et al., 2025), combinatorial optimization (Cappart et al., 2021), and LLM trustworthiness (Frasca et al., 2026). Yet benchmarking remains fragmented, i.e., task-specific datasets, inconsistent protocols, narrow domain coverage, and misaligned metrics limit reproducibility and meaningful comparison (Bechler-Speicher et al., 2025). In addition, existing benchmarks often overrepresent 2D molecular graphs while underrepresenting other impactful domains such as relational databases, chip design, and combinatorial optimization. Many datasets poorly reflect real-world structural complexity, and reliance on single metrics can encourage overfitting rather than robust generalization. These issues are especially limiting for evaluating graph foundation models (GFMs), which require broad, realistic, and challenging benchmarks.

Shortcomings of current graph learning benchmarks Existing benchmarks have advanced the field but remain limited in scope, scale, and evaluation coverage. TUDATASETS (Morris et al., 2020) offers many graph-level tasks, but most datasets are small, molecular, and lack unified metrics. The *Open Graph Benchmark* (Hu et al., 2020a) and its extensions (Hu et al., 2021) provide larger datasets, yet focus mainly on molecules and citation graphs, are partially saturated, support only limited model comparisons, and lack generative tasks. Other benchmarks (Dwivedi et al., 2020; 2022b) target narrow prediction regimes, small-scale settings, restricted model sizes, and limited out-of-distribution evaluation. Graph generation benchmarks are particularly underdeveloped (Bechler-Speicher et al., 2025), often relying on low-practical-value molecular or synthetic datasets and overlooking large or structurally constrained graph generation. Recent efforts such as dataset quality assessment (Coupette et al., 2025) and GRAPHLAND (Bazhenov et al., 2025) broaden coverage, but scalability and application gaps remain. As GFMs expand across tasks and domains (Xia & Huang, 2024; Bechler-Speicher et al., 2026), GRAPHBENCH addresses these gaps by providing a unified suite of challenging, real-world domains ready for GFM evaluation.

Present work We introduce GRAPHBENCH, a next-generation benchmarking suite addressing key shortcomings in graph learning evaluation. GRAPHBENCH spans diverse domains—social networks, electronic circuits, chip design, combinatorial optimization, SAT solver prediction, algorithmic reasoning, and weather forecasting—and supports node-

[1]RWTH Aachen University [2]University of Oxford [3]Mila – Quebec AI Institute [4]Technion - Israel Institute of Technology [5]ETAS Research [6]University of Stuttgart [7]Meta AI [8]AITHYRA [9]Google Research [10]Microsoft Research. Correspondence to: Timo Stoll <timo.stoll@log.rwth-aachen.de>.

The Workshop on Graph Foundation Models at the 43rd International Conference on Machine Learning (ICML 2026), Seoul, South Korea.

, edge-, graph-level, and generative tasks. It provides standardized splits, domain-specific metrics, dataset-generation instructions, and hyperparameter-tuning scripts, along with out-of-distribution evaluation on selected datasets to assess generalization. We benchmark modern MPNNs and GTs, offering strong baselines and insights into architectural trade-offs. *By unifying heterogeneous tasks under one framework,* GRAPHBENCH *enables reproducible, robust, and impactful graph learning research, paving the way for graph foundation models.*

2. Overview of GraphBench

The goal of GRAPHBENCH is to support the next generation of graph learning research by addressing key shortcomings of existing benchmarks. Here, we provide an overview of the design and features of GRAPHBENCH. Figure 1 shows an overview of the framework. GRAPHBENCH is designed around the following core principles.

1. **Diverse tasks and domains** GRAPHBENCH supports node-, edge-, and graph-level prediction, as well as generative tasks, across domains such as social networks, chip and analog circuit design, algorithm performance prediction, and weather forecasting. We describe each domain in Section 3.
2. **Real-world impact** Unlike benchmarks dominated by small-scale citation or molecular graphs, GRAPHBENCH targets modern real-world applications, including large-scale chip design, weather forecasting, and combinatorial optimization, better reflecting industrial and emerging research needs.
3. **OOD generalization** GRAPHBENCH evaluates out-of-distribution generalization through domain-specific temporal shifts or problem-size shifts, moving beyond random train–test splits. Dataset-specific challenges and OOD settings are detailed in the corresponding subsections.
4. **Task-relevant evaluation** GRAPHBENCH provides standardized splits, domain-specific metrics beyond accuracy and mean-squared error, and state-of-the-art hyperparameter tuning scripts. Evaluation and tuning procedures are described in Section 3, Appendix D, and Appendix E.
5. **Data generation pipelines** For all domains, GRAPHBENCH documents the full dataset generation process in Appendix B, supporting replication, extension, and future expansion of the benchmark suite.

Benchmarks and evaluation To support fair and reproducible comparisons, GRAPHBENCH provides a standardized benchmarking protocol for each dataset: a clearly defined prediction or generation task, realistic domain-specific splits, standardized input features and labels, and evaluation scripts with task-specific metrics such as RMSE, ROC-AUC, and closed gap. State-of-the-art hyperparameter tuning scripts further reduce preprocessing effort and enable consistent comparison under realistic conditions.

Table 1. Comparison of graph benchmark datasets across task levels, real-world domains, and OOD evaluation.

Benchmark	Node-level	Edge-level	Graph-level	Real-world Domains	OOD
OGB (Hu et al., 2020a; 2021)	✓	✓	✓	✓	
LRGB (Dwivedi et al., 2022b)	✓	✓	✓	✓	
GraphLand (Bazhenov et al., 2025)	✓		✓	✓	
CLRS (Velickovic et al., 2022)			✓		
SALSA-CLRS (Minder et al., 2023)			✓		✓
Weatherbench2 (Rasp et al., 2024)	✓			✓	✓
GraphBench	✓	✓	✓	✓	✓

Baselines and experimental evaluation We evaluate a range of graph learning architectures, including MPNNs, graph transformers, and recent strong baselines. To assess the importance of graph structure, we include MLP and DeepSet (Zaheer et al., 2017) baselines, which ignore connectivity. We also include GIN as a standard MPNN baseline, additional GNNs for electronic circuit datasets, the recent GNNPlus models (Luo et al., 2025), and a graph transformer based on Stoll et al. (2025). For weather forecasting, we provide a specialized graph-transformer model inspired by (Lam et al., 2023). Our results highlight key challenges for current methods, including OOD generalization and efficient training on graph-structured data.

Software and accessibility GRAPHBENCH is released as an open-source PYTHON package for easy adoption and extension. It includes streamlined data loaders, PYTORCH GEOMETRIC-based data objects, predefined splits, and hyperparameter tuning scripts. Its modular design enables researchers to prototype and benchmark new graph learning models under a unified evaluation protocol, while supporting extension to new domains, datasets, and baselines, as illustrated in Appendix A. Furthermore, GRAPHBENCH provides all information and data necessary to evaluate GFMs directly either in a zero-shot scenario or with additional training.

3. Overview of GraphBench's datasets

GRAPHBENCH encompasses four domains, each further divided into seven subdomains, yielding a total of 38 datasets. We *highlight four subdomains* and describe them in more detail, with exemplary datasets highlighted. *In-depth descriptions of the remaining subdomains and datasets, along with discussions of related work for each, can be found in Appendix B.*

3.1. Social sciences

This GRAPHBENCH domain models human interactions and societal processes as graphs, with nodes representing individuals or entities and edges representing relationships

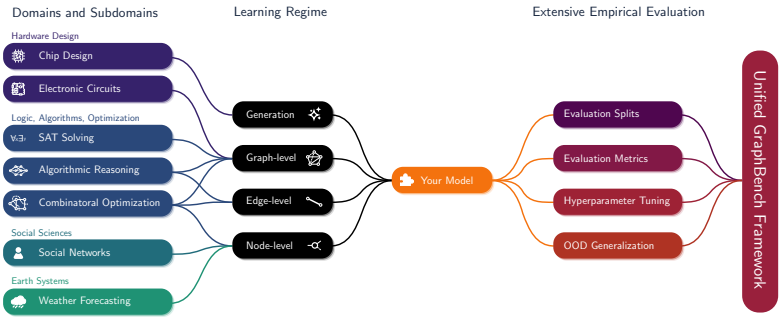

Figure 1. **Overview of the GraphBench framework.** GRAPHBENCH integrates datasets from diverse domains—ranging from social sciences and hardware design to logic, optimization, and earth systems—into different *learning regimes* (node-, edge-, graph-level prediction and generative tasks). GRAPHBENCH offers *extensive empirical evaluation* via standardized splits, metrics, hyperparameter tuning scripts, and out-of-distribution (OOD) generalization tests, all within a *unified evaluation pipeline*.

or communications. It supports tasks such as link prediction, community detection, and temporal forecasting, enabling analysis of real-world social systems.

3.1.1. SOCIAL NETWORKS: PREDICTING ENGAGEMENTS ON BLUESKY

Social networks—graphs with users as nodes and interactions as edges—are rich benchmarks for graph learning (Newman, 2010). They contain hubs, peripheral nodes, communities, evolving directed edges, and temporal dependencies driven by changing user interests. However, common datasets from SNAP (Leskovec & Sosic, 2016), TUDATASETS (Morris et al., 2020), and benchmarks such as FACEBOOK, GITHUB, TWITCH, LASTFM, and REDDIT (Rozemberczki et al., 2021; Hamilton et al., 2017) often collapse interactions into static, undirected graphs with categorical targets and limited or outdated node features. GRAPHLAND (Bazhenov et al., 2025) improves with temporal splits, but its social datasets largely rely on static, undirected[1] friendship relations and limited profile-based features.

In GRAPHBENCH, we use BLUESKY[2], a general, rapidly growing, and openly accessible social platform that exposes public posts and interactions. The task is to predict the number of engagements a user will receive on future posts, capturing near-term influence for applications such as user ranking, trend detection, influencer marketing, and proactive moderation. Compared to prior social graph benchmarks, our dataset provides realistic temporal splits, multi-relational time-evolving directed graphs, and content-grounded user representations from modern language model embeddings.

We next describe the learning task and datasets, with further details on data collection and processing.

[1]Only POKEC-REGIONS features *directed* friendship-based edges.

[2]https://bsky.app/

Description of the learning task We cast the problem as node-wise regression to predict an aggregated statistic on the engagements a user will receive on their *future* posts. For a time interval $\tau_{A,B} := (t_A, t_B] := \{t_A{+}1, \ldots, t_B\}$, we define: **(i)** an *Interaction graph* $G^{\iota}_{\tau_{A,B}} := (V, E^{\iota}_{\tau_{A,B}})$ where V, the node set, consists of social network users, and $E^{\iota}_{\tau_{A,B}}$ is the set of directed observed interactions in $\tau_{A,B}$; **(ii)** *Content* $T^{(u)}_{\tau_{A,B}} := \{\mathbf{c}_{u,p_t} \mid t \in \tau_{A,B}\}$, the set of the textual contents for posts p authored by user u in $\tau_{A,B}$; **(iii)** *Engagements* $E^{\kappa,(u)}_{\tau_{A,B}} := \{e^{\kappa}_{u,p_t} \mid t \in \tau_{A,B}\}$, collecting the number of engagements of kind κ received by the posts p authored by user u in $\tau_{A,B}$. We define an aggregate statistic $y^{\kappa,(u)}_{\tau_{A,B}} = \mu(E^{\kappa,(u)}_{\tau_{A,B}})$. Given timestamps $t_0 < t_1 < t_2$, user u and target engagement κ, the task is to predict value $y^{\kappa,(u)}_{\tau_{1,2}}$ given the overall interaction graph $G^{\iota}_{\tau_{0,1}}$ and post contents $\{T^{(u)}_{\tau_{0,1}} \mid u \in V\}$. We assess performance using the mean absolute error (MAE), the coefficient of determination R^2, and the Spearman correlation σ, which measure how well predictions preserve the relative ordering of users by future engagements, a key criterion for ranking influence in heavy-tailed social networks.

Details on the dataset We build upon the publicly available BLUESKY data curated by Failla & Rossetti (2024), referring the reader to their paper for full details on data collection and curation. Our dataset contains three distinct graph structures G^{ι}'s conveying different interaction kinds: a *directed* edge from node u to node v is drawn when u either *quoted*, *replied to*, or *reposted* v's posts. These interactions are timestamped, and we only construct edges from interactions observed in the interval $(t_0, t_1]$. For each user u, node features are derived from the content of their posts $T^{(u)}$ in the same interval, while distinct prediction targets are obtained as the *median* engagements counts $E^{\kappa,(u)}$ for its posts in time interval $(t_1, t_2]$, i.e., posterior to observed interactions and posts. Original post data span a time interval from t_{start} to t_{end}, and we fix three key time points t_{A}, t_{B}, t_{C} obtained so the proportion of posts in $\tau_{\text{start},A}, \tau_{A,B}, \tau_{B,C}, \tau_{C,\text{end}}$ amounts to, resp., $55\%, 15\%, 15\%, 15\%$. The dataset is accordingly

split into training, validation, and test sets. For each split we set $t_0 = t_{\text{start}}$ and choose t_1, t_2 as follows: *Training*: t_{A}, $t_2 = t_{\text{B}}$; *Val.*: $t_1 = t_{\text{B}}$, $t_2 = t_{\text{C}}$; *Test*: $t_1 = t_{\text{C}}$, $t_2 = t_{\text{end}}$. See Appendix B.1 for details on $t_{\text{start}}, t_{\text{end}}, t_0, \ldots, t_2$.

Challenges The temporal split changes the user set, content-based representations, and interaction graph across splits. Engagement counts may also depend on factors outside the training data, such as seasonality. By design, edges are accumulated from t_{start} to each split cutoff, making validation and test graphs denser than the training graph. This preserves interaction history, especially for users with sparse recent activity, but introduces structural shifts (see Table 7); sliding-window variants are left for future work.

Results Table 2 shows a consistent ranking from DeepSets to MLP to GNNs across all interaction types (quotes, replies, reposts), with GNNs achieving lower MAE and higher R^2 and Spearman (σ). Since DeepSets and MLPs ignore graph structure, this gap suggests that local, directed relational information helps predict short-term engagement. DeepSets underperforms MLP despite using global feature aggregation, indicating that global aggregation may wash out user-specific signals preserved by a structure-agnostic MLP. However, overall R^2 and σ remain modest, leaving room for stronger task-specific GNNs. GNNPlus architectures further improve performance across tasks. Due to the graph size, we omit positional encodings. Extended results and implementation details are provided in Appendix D.3.

3.2. Hardware design

The hardware design domain represents electronic circuits as graphs, with nodes denoting components such as transistors, resistors, or logic gates, and edges capturing electrical connections. These graphs support tasks such as circuit optimization and performance prediction, which are essential for efficient and reliable hardware development.

3.2.1. Electronic circuits: Predicting voltage conversion ratio

Analog circuit design is critical but remains labor-intensive in electronic design automation (Zhang et al., 2019; 2020; Zhao & Zhang, 2020). Unlike mature digital design flows, it still depends heavily on expert intuition due to device variation and complex constraints. This is especially challenging for custom power converters, where topology generation and parameter tuning must balance competing objectives across vast design spaces. Learning-based surrogate models can accelerate this search by predicting circuit performance (Fan et al., 2021; 2024b). Since power converters are naturally graph-structured, they provide a strong benchmark for testing robustness to structural sensitivity and extreme class imbalance. Figure 2 illustrates how an electronic circuit is transformed into a graph.

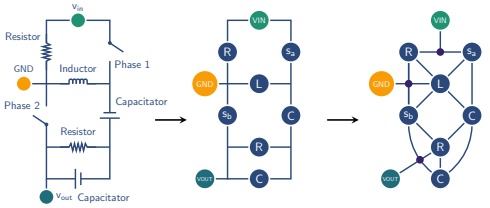

Figure 2. Transformation of an electronic circuit into a graph.

Description of the learning task Each circuit is represented as a graph $G = (V, E)$. The node set V contains device components: capacitors (C), inductors (L), and two phase-specific switches (S_a, S_b), and three external terminal ports: input (V_{in}), output (V_{out}) and ground (GND). Each component node $v \in V$ has a type $\tau(v)$ and (optional) parameter vector d_v and connects to other devices or ports via exactly two edges; each terminal node has a single outgoing edge. Edges E denote electrical interconnections between ports. Given a graph G, encoding a power converter, the prediction task is to estimate two continuous performance metrics: (1) the *voltage conversion ratio* (output-to-input voltage) and (2) the *power conversion efficiency* (fraction of input power delivered to the load). Performance is evaluated via the *relative squared error* (RSE), reported separately for each target,

$$\text{RSE}(y, \hat{y}) := \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2},$$

where y_i is the ground-truth value from high-fidelity simulation, $\hat{y}_i$ the model prediction, $\bar{y}$ the mean of $\{y_i\}_{i=1}^{N}$, and N the number of evaluation samples.

Details on the datasets Following the setting proposed by Fan et al. (2024b), we used datasets at three complexity levels, generated from random valid topologies with 5, 7, and 10 components. Topologies are sampled uniformly at random under connectivity constraints; isomorphic copies are removed to ensure uniqueness. Each instance is simulated with NGSPICE (Nenzi & Vogt, 2011) to obtain ground-truth voltage conversion ratio and efficiency; instances flagged as invalid during simulation are discarded. Unless otherwise specified, component and operating parameters are fixed as follows: capacitors $10\,\mu$F, inductors $100\,\mu$H, input voltage $100\,$V, switching frequency $1\,$MHz, an input resistor of $0\,1\,\Omega$ at V_{in}, and a $100\,\Omega$ load with a $10\,\mu$F output capacitor at V_{out}. Generated graphs consist of internal nodes (depending on component size) and three terminal nodes each.

Challenges Aside from domain-specific challenges in circuit design outlined in previous sections, major difficulties in the datasets arise with an increasing number of electronic components in a graph. As the number of components increases, the number of possible circuit arrangements and paths in each graph increases, leading to greater difficulty

Table 2. **Social Networks.** Performance metrics for each model across datasets. MAE, R^2, and Spearman correlation are reported with their standard deviation. For replies and reposts datasets GatedGCN+ ran out of memory in all tested configurations.

Dataset	Model	MAE ↓	R^2_{likes} ↑	R^2_{replies} ↑	R^2_{reposts} ↑	σ_{likes} ↑	σ_{replies} ↑	σ_{reposts} ↑
quotes	DeepSets	0.810± 0.005	0.140± 0.002	0.102± 0.002	0.138± 0.002	0.307± 0.004	0.178± 0.002	0.334± 0.003
	MLP	0.784± 0.001	0.145± 0.001	0.107± 0.001	0.141± 0.001	0.308± 0.003	0.176± 0.002	0.335± 0.002
	GNN	0.768± 0.002	0.165± 0.002	0.134± 0.002	0.175± 0.002	0.330± 0.003	0.192± 0.002	0.337± 0.022
	GIN+	0.737± 0.014	0.165± 0.017	0.126± 0.030	0.166± 0.021	0.382± 0.005	0.358± 0.010	0.405± 0.005
	GatedGCN+	0.734± 0.010	0.168± 0.016	0.132± 0.019	0.173± 0.018	0.384± 0.008	0.356± 0.008	0.411± 0.007
	GCN+	0.781± 0.021	0.146± 0.008	0.108± 0.010	0.145± 0.009	0.351± 0.009	0.296± 0.010	0.351± 0.009
replies	DeepSets	0.789± 0.033	0.086± 0.033	0.061± 0.024	0.104± 0.014	0.253± 0.003	0.130± 0.001	0.240± 0.006
	MLP	0.725± 0.004	0.131± 0.003	0.087± 0.003	0.122± 0.003	0.249± 0.004	0.127± 0.005	0.243± 0.003
	GNN	0.694± 0.002	0.158± 0.003	0.119± 0.002	0.159± 0.002	0.258± 0.009	0.132± 0.006	0.247± 0.011
	GIN+	0.669± 0.017	0.161± 0.020	0.112± 0.019	0.153± 0.018	0.364± 0.004	0.327± 0.009	0.365± 0.004
	GCN+	0.686± 0.010	0.139± 0.007	0.081± 0.010	0.128± 0.010	0.337± 0.004	0.287± 0.005	0.344± 0.004
reposts	DeepSets	0.918± 0.013	0.049± 0.005	0.031± 0.012	0.050± 0.009	0.229± 0.004	0.129± 0.007	0.205± 0.009
	MLP	0.874± 0.003	0.087± 0.002	0.051± 0.003	0.066± 0.002	0.234± 0.008	0.123± 0.006	0.206± 0.010
	GNN	0.832± 0.009	0.131± 0.017	0.111± 0.022	0.128± 0.008	0.284± 0.044	0.143± 0.032	0.260± 0.051
	GIN+	0.810± 0.015	0.150± 0.005	0.108± 0.005	0.141± 0.010	0.339± 0.008	0.314± 0.009	0.361± 0.006
	GCN+	0.831± 0.013	0.101± 0.013	0.068± 0.016	0.092± 0.016	0.279± 0.005	0.262± 0.008	0.316± 0.005

in correctly predicting the target.

Results The results in Table 3 confirm that the GT consistently achieves higher prediction accuracy than MPNN baselines for seven and ten components across tasks. However, GNNPlus baselines show improved performance on small circuits, indicating a potential advantage of local message passing. The increase in RSE for larger circuits is mainly due to the reduced availability of training data from costly high-fidelity simulations, combined with the combinatorial growth of circuit space, which makes learning increasingly difficult for all models. However, performance differences among models are evident across all circuit sizes.

Table 3. Performance comparison of electronic circuits baseline models (RSE, lower is better).

	Efficiency			Voltage		
Method	**5 comp**	**7 comp**	**10 comp**	**5 comp**	**7 comp**	**10 comp**
GT	0.07± 0.03	0.16± 0.01	0.37± 0.02	0.06± 0.01	0.25± 0.02	0.40± 0.03
GIN	0.09± 0.06	0.21± 0.04	0.44± 0.13	0.14± 0.05	0.35± 0.03	0.54± 0.12
GIN+	0.04± 0.01	0.22± 0.02	0.49± 0.07	0.05± 0.01	0.33± 0.02	0.64± 0.05
GCN+	0.04± 0.01	0.24± 0.05	0.53± 0.05	0.05± 0.01	0.34± 0.03	0.60± 0.03
GatedGCN+	0.04± 0.01	0.27± 0.07	0.55± 0.01	0.05± 0.01	0.30± 0.02	0.59± 0.0

3.2.2. Chip design: Learning to generate small circuits

A central challenge in Boolean circuit synthesis is ensuring optimized circuits remain functionally equivalent to the original: regardless of transformations aimed at reducing gate count, delay, power, or area, outputs must match for all inputs. This makes the task delicate—small changes can alter behavior. Finding a minimal circuit is NP-hard, making synthesis a core computational challenge. Classical tools such as ABC (Brayton & Mishchenko, 2010) use heuristics to efficiently generate near-optimal circuits. Recently, learning-based methods have emerged as alternatives that use data-driven models to capture structural patterns and generalize across designs (Shi et al., 2023; Li et al., 2025b). The natural representation of Boolean circuits as directed acyclic graphs (DAGs) makes graph-based generative learning particularly promising. However, generating DAGs is challenging due to the need for exact graph-truth-table matching, and because current methods are not designed to use DAGs as inputs. We provide a dataset that tasks the network with constructing structurally efficient logic circuits that are equivalent to the given logic functions. Details on the dataset, arising challenges, and experimental evaluation can be found in Appendix B.2.1.

3.3. Reasoning and optimization

This GraphBench domain encompasses graph learning tasks that require complex reasoning and hard optimization, ranging from theorem proving and hardware verification to logistics and network design. Models must capture fine-grained structural dependencies and generalize across instances. We consider three problem families: (i) *SAT solver selection and performance prediction*, predicting solver behavior or choosing the best solver per instance; (ii) *combinatorial optimization*, including NP-hard problems such as maximum independent set, max-cut, and graph coloring; and (iii) *algorithmic reasoning*, where models approximate polynomial-time graph algorithms. Benchmarks assess accuracy, robustness, scalability, and out-of-distribution generalization.

3.3.1. Combinatorial optimization: Learning to predict optimal solution objectives

Combinatorial optimization (CO) lies at the intersection of optimization, operations research, discrete mathematics, and computer science (Korte & Vygen, 2012). The goal is to find a solution from a discrete set that optimizes a given objective under constraints. Formally, a *combinatorial optimization problem instance* I is described by a tuple (Ω, F, w), where $\Omega(I)$ is a *finite set*, $F(I) \subseteq 2^{\Omega(I)}$ is the set of *feasible solutions*, and $w\colon \Omega(I) \to \mathbb{R}$ assigns *weights* to elements of $\Omega(I)$. The *cost* or *objective value* of a solution $S \in F(I)$ is given by $c(S) = \sum_{\omega \in S} w(\omega)$. The goal is to find an optimal solution $S^*_I \in F(I)$ that minimizes c over the feasible set (maximization problems can be converted to minimization

problems by switching the sign).

CO is central to applications such as vehicle routing, scheduling, and resource allocation (Paschos, 2014). Graphs are a natural representation for CO because many problems are inherently discrete, network-based, or easily reformulated as graphs. Their computational hardness (Karp, 1972) makes CO well-suited for learning-based methods, where MPNNs and GNNs can serve as efficient surrogates for large instances while requiring expressive architectures to solve them effectively (Chen et al., 2023; 2024).

Description of the learning tasks We include tasks for both supervised and unsupervised learning settings. For the supervised setting, the objective is to train a graph learning model to predict the optimal objective value of given CO instances. Given a set of CO problem instances $\mathcal{I}$ with corresponding optimal solutions S_I^*, and a model $m\colon \mathcal{I} \to \mathbb{R}$, the learning task is to minimize the MAE between the predicted and true objective values, $1/|\mathcal{I}| \sum_{I\in\mathcal{I}} |m(I) - c(S_I^*)|$. While this setup focuses on predicting the objective value rather than producing explicit solutions, it offers a practical advantage: it avoids the challenge of defining a meaningful loss function when multiple distinct optimal solutions exist. This simplification enables consistent, well-defined evaluation and learning. The supervised setting relies on solver-generated solutions, which can be computationally infeasible for large instances. This motivates the unsupervised setting, which is valuable when ground-truth solutions are unavailable or expensive to obtain, and it directly targets the CO problem. We provide a differentiable surrogate loss function $\mathcal{L}\colon \mathbb{R}^{|\Omega(I)|} \to \mathbb{R}$ as well as decoders $d\colon \mathbb{R}^{|\Omega(I)|} \to F(I)$ for each CO problem. The model $m\colon \mathcal{I} \to \mathbb{R}^{|\Omega(I)|}$ is trained in an unsupervised fashion to predict a score for each variable that indicates whether it belongs to the solution set, minimizing $1/|\mathcal{I}| \sum_{I\in\mathcal{I}} \mathcal{L}(m(I))$. At test time, the model's output scores are converted into a feasible solution to the CO problem using the decoder. The model's performance is measured based on the objective value of the decoded solution, $c(d(m(I)))$. This setup is widely used in graph learning for CO literature (Karalias & Loukas, 2020; Min et al., 2022; Wenkel et al., 2024). We adapt our surrogate loss functions and decoders from this existing literature.

Details on the datasets We consider three representative hard CO problems on graphs, the *maximum independent set* (MIS), the *max-cut*, and the *graph coloring problem*. They span a broad spectrum of structural properties and computational challenges. We formally define these problems in Appendix B.3.2. In this section, we use MIS as an example to find the largest subset of non-adjacent nodes. The generality of the (Ω, F, w) framework makes it applicable to a wide range of CO problems, beyond those proposed in our benchmark. We use three well-established random graph models to generate problem graphs: RB graphs (Xu et al., 2007), Erdős-Rényi (ER) graphs (Erdős & Rényi, 1960), and Barabási-Albert (BA) graphs (Albert & Barabási, 2002). These models are widely used to train CO models. We generate 50 000 instances per dataset and provide both small- and large-scale variants. Appendix B.3.2 provides a summary of the parameters used for graph generation, which are taken from existing literature. The generated graphs include no node or edge features.

Challenges Since most CO problems are challenging to solve with existing solvers such as KaMIS or Gurobi, a major challenge lies in providing an optimal solution via a graph based method. Furthermore, solving the respective CO problems is shown to require more than 1-WL expressivity (Sato et al., 2019), indicating the necessity for expressive architectures. In addition, the proposed unsupervised target provides additional challenges regarding training of MPNN methods, since most of them are usually trained without a surrogate loss.

Results We show supervised learning results on MIS in Table 4 and unsupervised results, including all three CO problems, in Appendix B.3.2. Extended unsupervised results, including the other two problems, can be found in subsubsection B.3.2. Among the baselines, GIN achieves the best performance across most datasets. We attribute this to MPNNs' strong inductive bias, which aligns well with the structure of graph-based CO problems. DeepSet generally outperforms the MLP baseline on the supervised task, except for the ER graphs. This suggests that global information aggregation is beneficial for this task. Conversely, the GT performs poorly on most datasets. We hypothesize that this is due to the training difficulties associated with this task.

Table 4. MIS results for each CO dataset with supervised learning (MAE, lower is better).

	RB graph		ER graph		BA graph	
Method	**Small**	**Large**	**Small**	**Large**	**Small**	**Large**
GIN	0.491± 0.099	2.125± 0.484	0.234± 0.191	0.352± 0.265	0.292± 0.041	0.111± 0.016
GT	4.112± 2.353	0.915± 0.235	6.486± 8.101	9.641± 15.335	3.481± 3.446	1.829± 1.383
MLP	1.583± 0.052	1.437± 0.520	0.751± 0.767	0.914± 0.307	2.825± 0.949	3.383± 0.504
DeepSets	0.918± 0.186	1.427± 0.224	0.756± 0.711	2.244± 0.364	2.304± 0.262	3.362± 1.491
GCN+	0.024± 0.004	0.015± 0.001	0.011± 0.001	0.009± 0.0001	0.0009± 0.0002	0.004± 0.001
GIN+	0.021± 0.0002	0.016± 0.001	0.012± 0.001	0.008± 0.001	0.0006± 0.0002	0.001± 0.0003

3.3.2. SAT SOLVING: ALGORITHM SELECTION AND PERFORMANCE PREDICTION

The Boolean satisfiability problem (SAT) is a longstanding NP-complete (Cook, 1971) problem that remains theoretically significant. Beyond its role in complexity theory, it has many applications, including hardware/software verification (Biere et al., 2009), automated planning (Kautz & Selman, 1996), and operations research (Gomes et al., 2008). SAT instances can be expressed as graphs, reflecting the permutation-invariance induced by the commutativity and associativity of conjunction and disjunction. We pro-

vide three datasets that span a wide range of instance sizes and, to the best of our knowledge, constitute the largest dataset for algorithm selection and performance prediction for SAT solvers. Each is provided with two learning tasks: *performance prediction*, a regression problem whose goal is to predict the computation time of SAT solvers on unseen instances, and *algorithm selection*, a multi-class classification problem that aims to select the best performing algorithm for a given SAT instance. Dataset-specific challenges include noisy target data due to inherently stochastic solver runtimes and large-scale graphs requiring capturing both local and global formula structure. See Appendix B.3.1 for details on the data, baselines, and experimental results.

3.3.3. Algorithmic reasoning: Learning to simulate algorithms

In addition to the SAT, many real-world tasks depend on efficient graph algorithms. Given the active exploration of the intersection of algorithms and neural networks across domains (Estermann et al., 2024; Fan et al., 2024a; Kaiser & Sutskever, 2016; Zaremba & Sutskever, 2014), our goal is to provide dedicated datasets for graph algorithmic reasoning. Prior work (Velickovic et al., 2020b; Xu et al., 2020; Bounsi et al., 2024) and benchmarks such as CLRS (Velickovic et al., 2022) have demonstrated that GNN architectures can achieve strong performance on graph problems when provided with hints and graph invariants—tasks that require substantial theoretical expressiveness from graph learning models (Arvind et al., 2020). To broaden this research area, we contribute 21 large-scale datasets that cover seven classic graph algorithms, each at three difficulty levels, thereby expanding the landscape of neural algorithmic reasoning benchmarks. To further investigate the algorithm's learning capabilities, all datasets include a shift in the data distribution of generated graphs, as well as larger graphs in the test sets. Throughout experiments, we observe strong baseline performance of GNN and GT models, while extrapolation regarding size and data distribution is not perfectly achieved. For details, as well as baselines and experimental results, see Appendix B.3.3.

3.4. Earth systems

The earth systems domain in GraphBench covers graph learning tasks from geospatial, environmental, and climate data. Nodes represent locations, sensors, or regions, while edges encode spatial or physical relationships. These tasks support applications such as weather forecasting, climate impact assessment, and resource management, testing whether graph-based models can integrate heterogeneous features, capture long-range dependencies, and generalize across time and space.

3.4.1. Weather forecasting: Medium-range atmospheric state prediction

Weather forecasting is critical for agriculture, energy, and public safety (Diehl et al., 2013; Ramar & Mirnalinee, 2014). Traditional *numerical weather prediction* (NWP) is accurate but computationally expensive due to physics-based simulations (Bauer et al., 2015). Recent machine learning methods provide faster, scalable alternatives that can outperform NWP for medium-range forecasts and extreme events, while improving uncertainty quantification (Price et al., 2023). Because earth system data are sparse, irregular, and interconnected (Reichstein et al., 2019), with interactions spanning local to planetary scales (Bauer et al., 2015), graph-based models offer a natural way to capture multi-scale, non-Euclidean atmospheric dependencies.

Description of the learning task The objective of the task is to model medium-range weather evolution by predicting the residual change in the atmospheric state over a fixed 12-hour time horizon. Specifically, given an initial snapshot of the current atmospheric state, the model forecasts the twelve-hour future change in meteorological variables. The training objective is a spatially and variable-weighted *mean-squared error* (MSE) for the twelve-hour-ahead prediction, inspired by the training objective used in GraphCast (Lam et al., 2023). For each verification time $d \in D$, the model uses x_{d-2}, x_{d-3} to predict a residual change Δx, yielding $\hat{x}^d = x_{d-2} + \Delta x$. The loss compares $\hat{x}^d$ with x^d. The loss function thus corresponds to

$$\mathcal{L}(x^d, \hat{x}^d) := \frac{1}{|D|\,|G| \sum_{j \in J} |L_j|} \sum_{d \in D} \sum_{i \in G} \sum_{j \in J} \sum_{\ell \in L_j} a_i \, w_j \, s_{j,\ell} \left(\hat{x}^d_{i,j,\ell} - x^d_{i,j,\ell}\right)^2,$$

with level weights $s_{j,\ell}$ where D is the set of forecast datetimes, G the grid cells, J the variables, L_j the pressure levels for variable j, $a_i = \frac{\cos(\mathrm{lat}_i)}{\frac{1}{|G|}\sum_{k \in G} \cos(\mathrm{lat}_k)}$ are mean-normalized latitude weights, w_j are variable weights, and P_ℓ are the pressure levels. During evaluation, an unweighted MSE is reported for each variable. Predicting a residual change Δx rather than an absolute value improves stability and generalization (He et al., 2020).

Details on the dataset We utilize reanalysis data from the ERA5 dataset, which has been preprocessed via the WeatherBench2 pipeline (Rasp et al., 2024). Multiple resolutions of this dataset are available. We use a downsampled version, containing a 64×32 equiangular grid, employing a conservative area-preserving interpolation. ERA5 data has a temporal resolution of 6 hours, with time steps at 0h, 6h, 12h, and 18h. Each weather state contains 62 physical and derived variables: 15 are defined across the 13 pressure levels, and 47 are defined at the surface. Of those, we use five

variables for the surface and six for the atmospheric levels each, as outlined by Price et al. (2023). The pressure-level variables include temperature, humidity, and wind speed. The surface-level variables also include information about the location, such as a land-sea mask, sea level pressure, and precipitation. The dataset includes the original grid data as a graph, the icosahedron as a mesh graph, and the edge mapping between the two. Following (Price et al., 2023), the mapping is learned during training. Each grid node contains the current weather state x_{d-2} and the previous state x_{d-3}, as well as static location information. For variables used in evaluation, see Table 5, and for an extended description of the weather forecasting datasets, see Appendix D.4.

Table 5. MSE for each weather variable for selected pressure levels and surface variables. Lower is better. The selected pressure levels align with the evaluated levels of WeatherBench2 (Rasp et al., 2024). Persistence is a basic weather forecasting model that provides a forecast by assuming that variable values remain constant, as in the input values. GraphCast is referred to in Lam et al. (2023). Number formatting: k indicates a multiplicative factor of 10^3 while n indicates 10^{-9}.

Pressure Level	Variable	GT	Persistence	GraphCast
Surface	2-m temperature (2T)	0.841±0.013	7.123	0.068
	10-m u wind component (10U)	1.162±0.016	2.166	0.012
	10-m v wind component (10V)	1.422±0.023	3.266	0.013
	Mean sea level pressure (MSL)	13.035k±0.294k	60.056k	240.832
	Total precipitation (TP)	422.697n±9.143n	714.517n	52.377n
500	Temperature (T)	0.529±0.004	1.120	0.007
	U component of wind (U)	4.398±0.082	6.658	0.048
	V component of wind (V)	5.843±0.081	12.988	0.053
	Geopotential (Z)	958.756±17.791	48.637k	155.057
	Specific humidity (Q)	55.49n±1.362n	66.211n	1.090n
	Vertical wind speed (W)	0.0051±0.00009	6.242	0.050
700	Temperature (T)	0.539±0.007	1.051	0.006
	U component of wind (U)	2.608±0.072	3.951	0.025
	V component of wind (V)	3.312±0.082	6.754	0.027
	Geopotential (Z)	634.775±15.533	32.773k	141.296
	Specific humidity (Q)	191.388n±5.037n	250.556n	3.304n
	Vertical wind speed (W)	0.0046±0.00013	3.352	0.027
850	Temperature (T)	0.735±0.018	1.351	0.009
	U component of wind (U)	2.263±0.071	4.015	0.022
	V component of wind (V)	2.839±0.065	6.565	0.023
	Geopotential (Z)	630.858±15.533	32.453k	139.716
	Specific humidity (Q)	276.236n±4.399n	354.058n	4.881n
	Vertical wind speed (W)	0.0031±0.00008	3.517	0.024

Challenges Weather data exhibit a strong temporal distribution shift: climate trends and global weather phenomena change atmospheric statistics over time, requiring models to be robust to more than interpolation. Although naturally graph-structured, the spatial grid is highly regular, with most nodes sharing similar local neighborhoods. This homogeneity makes connectivity alone weakly informative, increasing the importance of positional encodings and global context. The task also requires modeling long-range interactions, which challenge local message passing and can exacerbate oversmoothing. Finally, node features are high-dimensional and heterogeneous, spanning variables with different scales, units, and pressure levels, requiring models to balance competing signals during training.

Results Table 5 reports results for each weather variable at three pressure levels, following WeatherBench2 (Rasp et al., 2024). Alongside GraphCast (Lam et al., 2023) and the Persistence baseline, we evaluate an additional model based on the GDT implementation of Stoll et al. (2025). We use a 12-hour forecast horizon across all variables. Our baseline provides a transparent and reproducible lower bound for medium-range weather forecasting. Although it underperforms GraphCast, this is expected given its simpler architecture, shorter training time, and lower-resolution training data. Nevertheless, it improves over Persistence for every variable, indicating effective learning. Implementation details are provided in Appendix B.4.1.

4. Format, licensing, and long-term access

Data are released under open licenses: Apache 2.0 for most datasets, including WeatherBench2; MIT for SAT instances; and GPL for AClib. For social networks, the original BlueSky data are hosted at `https://zenodo.org/records/11082879` under the Creative Commons Attribution 4.0 International License. These records may be updated to respect users' "Right to Erasure", and we are implementing a system to keep our dataset consistent with such updates. All GRAPHBENCH datasets are released under the Creative Commons Attribution-NonCommercial 4.0 license to ensure compliance with source licenses, except weather forecasting data, which follows WeatherBench2 and is released under Apache 2.0. GRAPHBENCH is continuously updated with new datasets and hosted on institutional servers and public repositories for persistent access.

5. Conclusion and outlook

We introduce GRAPHBENCH, a next-generation benchmark suite that addresses the fragmentation of graph learning evaluation. By unifying various domains within a standardized framework, GRAPHBENCH enables fair and reproducible comparison across diverse graph tasks. The suite covers node-, edge-, and graph-level prediction as well as generative tasks, using realistic splits, task-relevant metrics, and explicit out-of-distribution evaluation. Strong baselines with modern MPNNs and graph transformers provide reference points for future work. Our results highlight persistent challenges, including temporal distribution shift, scalability to large graphs, and modeling domain-specific structure. Through principled datasets, consistent evaluation, and robust baselines, GRAPHBENCH aims to advance generalizable and practically relevant graph learning.

Vision for GRAPHBENCH *We envision* GRAPHBENCH *as a living benchmark that continually expands to new domains, tasks, and evaluation paradigms, supporting progress in both fundamental graph machine learning research and real-world applications.* We further aim to extend GRAPHBENCH to support training next-generation multimodal graph foundation models.

Acknowledgements

TS and CM are partially funded by a DFG Emmy Noether grant (468502433) and RWTH Junior Principal Investigator Fellowship under Germany's Excellence Strategy.

References

Alamgir, N., Nejati, S., and Bright, C. SHA-256 collision attack with programmatic SAT. In *Workshop on Practical Aspects of Automated Reasoning (PAAR) and Satisfiability Checking and Symbolic Computation Workshop (SC-Square), adjunct to (IJCAR 2024)*, 2024.

Albert, R. and Barabási, A.-L. Statistical mechanics of complex networks. *Reviews of modern physics*, (1), 2002.

Alet, F., Price, I., El-Kadi, A., Masters, D., Markou, S., Andersson, T. R., Stott, J., Lam, R., Willson, M., Sanchez-Gonzalez, A., and Battaglia, P. Skillful joint probabilistic weather forecasting from marginals. *ArXiv preprint*, 2025.

Ansótegui, C., Giráldez-Cru, J., and Levy, J. The community structure of SAT formulas. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2012.

Arvind, V., Fuhlbrück, F., Köbler, J., and Verbitsky, O. On weisfeiler-leman invariance: Subgraph counts and related graph properties. *Journal of Computer and System Sciences*, 2020.

Balyo, T., Heule, M. J. H., and Järvisalo, M. SAT competition 2016: Recent developments. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

Bansal, A., Stoll, D., Janowski, M., Zela, A., and Hutter, F. Jahs-bench-201: A foundation for research on joint architecture and hyperparameter search. In *Proccedings of Advances in Neural Information Processing Systems NeurIPS*, 2022.

Barabási, A.-L. and Albert, R. Emergence of scaling in random networks. *science*, (5439), 1999.

Bauer, P., Thorpe, A., and Brunet, G. The quiet revolution of numerical weather prediction. *Nature*, (7567), 2015.

Bazhenov, G., Platonov, O., and Prokhorenkova, L. Graphland: Evaluating graph machine learning models on diverse industrial data. *arXiv preprint*, 2025.

Bechler-Speicher, M., Finkelshtein, B., Frasca, F., Müller, L., Tönshoff, J., Siraudin, A., Zaverkin, V., Bronstein, M. M., Niepert, M., Perozzi, B., Galkin, M., and Morris, C. Position: Graph learning will lose relevance due to poor benchmarks. *ArXiv preprint*, 2025.

Bechler-Speicher, M., Gottlieb, Y., Isakov, A., Abensur, D., Tavory, A., Haimovich, D., Guy, I., and Weinsberg, U. Billion-scale graph foundation models. *arXiv preprint arXiv:2602.04768*, 2026.

Bevilacqua, B., Nikiforou, K., Ibarz, B., Bica, I., Paganini, M., Blundell, C., Mitrovic, J., and Velickovic, P. Neural algorithmic reasoning with causal regularisation. In *Proceedings of the International Conference on Machine Learning ICML*, Proceedings of Machine Learning Research, 2023.

Bi, K., Xie, L., Zhang, H., Chen, X., Gu, X., and Tian, Q. Pangu-weather: A 3d high-resolution model for fast and accurate global weather forecast. *ArXiv preprint*, 2022.

Biere, A., Heule, M., van Maaren, H., and Walsh, T. *Handbook of Satisfiability*. Frontiers in Artificial Intelligence and Applications. 2009.

Biere, A., Heljanko, K., and Wieringa, S. Aiger 1.9 and beyond, 2011.

Biere, A., Heule, M., Maaren, H. v., and Walsh, T. *Handbook of Satisfiability - Second Edition*. 2021.

Biere, A., Fleury, M., and Pollitt, F. Cadical_vivinst, isasat, gimsatul, kissat, and tabularasat entering the sat competition 2023. In *Proceedings of SAT Competition 2023*, 2023.

Biere, A., Faller, T., Fazekas, K., Fleury, M., Froleyks, N., and Pollitt, F. CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In *Proceedings of SAT Competition 2024*, 2024.

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H. H., Hutter, F., Leyton-Brown, K., Tierney, K., and Vanschoren, J. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 2016.

Bodnar, C., Bruinsma, W. P., Lucic, A., Stanley, M., Brandstetter, J., Garvan, P., Riechert, M., Weyn, J. A., Dong, H., Vaughan, A., Gupta, J. K., Thambiratnam, K., Archibald, A. T., Heider, E., Welling, M., Turner, R. E., and Perdikaris, P. Aurora: A foundation model of the atmosphere. 2024.

Bogaerts, B., Nordström, J., Oertel, A., and Yıldırımoglu, C. U. Breakid-kissat in sat competition 2023 (system description). In *Proceedings of SAT Competition 2023*, 2023.

Bogaerts, B., Nordström, J., Oertel, A., and Vandesande, D. BreakID-kissat in SAT Competition 2024 (System Description). In *Proceedings of SAT Competition 2024*, 2024.

Böther, M., Kißig, O., Taraz, M., Cohen, S., Seidel, K., and Friedrich, T. What's wrong with deep learning in tree search for combinatorial optimization. In *Proceedings of the International Conference on Learning Representations ICLR*, 2022.

Bounsi, W., Ibarz, B., Dudzik, A., Hamrick, J. B., Markeeva, L., Vitvitskyi, A., Pascanu, R., and Velickovic, P. Transformers meet neural algorithmic reasoners. *arXiv preprint*, 2024.

Brayton, R. and Mishchenko, A. Abc: An academic industrial-strength verification tool. In *Proceedings of the International Conference on Computer Aided Verification CAV*. Springer, 2010.

Breiman, L. Random forests. *Machine Learning*, (1), 2001.

Brummayer, R., Lonsing, F., and Biere, A. Automated testing and debugging of SAT and QBF solvers. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing SAT*, Lecture Notes in Computer Science, 2010.

Cantürk, S., Manolache, A., Mielke, A., Qian, C., Siraudin, A., Morris, C., Niepert, M., and Wolf, G. Graip: A benchmarking framework for neural graph inverse problems. *arXiv preprint*, 2026.

Cappart, Q., Chételat, D., Khalil, E. B., Lodi, A., Morris, C., and Velickovic, P. Combinatorial optimization and reasoning with graph neural networks. In *Proceedings of the International Joint Conference on Artificial Intelligence IJCAI*, 2021.

Caramia, M. and Dell'Olmo, P. k-insertions graphs: a proposal of a new class of benchmarks. Manuscript, 2002a.

Caramia, M. and Dell'Olmo, P. k-fullins graphs: a proposal of a new class of benchmarks. Manuscript, 2002b.

Carballo-Castro, A., Madeira, M., Qin, Y., Thanou, D., and Frossard, P. Generating directed graphs with dual attention and asymmetric encoding. *ArXiv preprint*, 2025.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

Chen, Z., Liu, J., Wang, X., and Yin, W. On representing mixed-integer linear programs by graph neural networks. In *Proceedings of the International Conference on Learning Representations ICLR*, 2023.

Chen, Z., Liu, J., Chen, X., Wang, X., and Yin, W. Rethinking the capacity of graph neural networks for branching strategy. In *Proceedings of Advances in Neural Information Processing Systems NeurIPS*, 2024.

Chvátal, V. Coloring the queen graphs, 2004. URL http://www.cs.concordia.ca/~chvatal/queengraphs.html. Web repository.

Cook, S. A. The complexity of theorem-proving procedures. *ACM Symposium on Theory of Computing (STOC)*, 1971.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms, 3rd Edition*. 2009. ISBN 978-0-262-03384-8.

Coupette, C., Wayland, J., Simons, E., and Rieck, B. No metric to rule them all: Toward principled evaluations of graph-learning datasets. In *Proceedings of the International Conference on Machine Learning ICML*, 2025.

Diao, C. and Loynd, R. Relational attention: Generalizing transformers for graph-structured tasks. In *Proceedings of the International Conference on Learning Representations ICLR*, 2023.

Diehl, A., Bruckner, S., Gröller, M. E., Delrieux, C., and Saulo, C. Visual trend analysis in weather forecast. In *Proceedings of the International Conference on Information Visualisation*, 2013.

Dwivedi, V. P., Joshi, C. K., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *arXiv preprint*, 2020.

Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. Graph neural networks with learnable structural and positional representations. In *Proceedings of the International Conference on Learning Representations ICLR*, 2022a.

Dwivedi, V. P., Rampásek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. Long range graph benchmark. In *Proceedings of the Advances in Neural Information Processing Systems*, 2022b.

Eggensperger, K., Hutter, F., Hoos, H. H., and Leyton-Brown, K. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015.

Eggensperger, K., Lindauer, M., and Hutter, F. Pitfalls and best practices in algorithm configuration. *Journal of Artificial Intelligence Research*, 2019.

Erdős, P. and Rényi, A. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 1960.

Estermann, B., Lanzendörfer, L. A., Niedermayr, Y., and Wattenhofer, R. PUZZLES: A benchmark for neural algorithmic reasoning. In *Proceedings of the Advances in Neural Information Processing Systems NeurIPS*, 2024.

Failla, A. and Rossetti, G. "i'm in the bluesky tonight": Insights from a year worth of social data. *PLoS ONE*, (11), 2024.

Falkner, S., Klein, A., and Hutter, F. BOHB: robust and efficient hyperparameter optimization at scale. In *Proceedings of the International Conference on Machine Learning ICML*, Proceedings of Machine Learning Research, 2018.

Fan, L., Hua, W., Li, L., Ling, H., and Zhang, Y. Nphard-eval: Dynamic benchmark on reasoning ability of large language models via complexity classes. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024a.

Fan, S., Cao, N., Zhang, S., Li, J., Guo, X., and Zhang, X. From specification to topology: Automatic power converter design via reinforcement learning. *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021.

Fan, S., Lu, H., Zhang, S., Cao, N., Zhang, X., and Li, J. Graph-transformer-based surrogate model for accelerated converter circuit topology design. In *Proceedings of the ACM/IEEE Design Automation Conference*, 2024b.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint*, 2019.

Fey, M., Sunil, J., Nitta, A., Puri, R., Shah, M., Stojanovič, B., Bendias, R., Barghi, A., Kocijan, V., Zhang, Z., He, X., Lenssen, J. E., and Leskovec, J. Pyg 2.0: Scalable learning on real world graphs. In *Proceedings of the Temporal Graph Learning Workshop adjunct to KDD*, 2025.

Frasca, F., Bar-Shalom, G., Ziser, Y., and Maron, H. Neural message-passing on attention graphs for hallucination detection. In *International Conference on Learning Representations*, 2026.

Froleyks, N., Heule, M., Iser, A., Järvisalo, M., and Suda, M. SAT competition 2020. *Artificial Intelligence*, 2021.

Gansner, E., Koutsofios, E., and North, S. Drawing graphs with dot, 2006.

Gao, Y. Kissat MAB prop in SAT Competition 2023. In *Proceedings of SAT Competition 2023*, 2023.

Gao, Y., Wu, H., Shu, R., Dong, H., Xu, F., Chen, R. R., Yan, Y., Wen, Q., Hu, X., Wang, K., Wu, J., Li, Q., Xiong, H., and Huang, X. Oneforecast: a universal framework for global and regional weather forecasting. In *Proceedings of the International Conference on Machine Learning ICML*, 2025.

Giráldez-Cru, J. and Levy, J. Generating SAT instances with community structure. *Artificial Intelligence*, 2016.

Goldberg, A. V. and Tarjan, R. E. A new approach to the maximum flow problem. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, 1986. ISBN 0897911938.

Goldie, A., Mirhoseini, A., and Dean, J. That chip has sailed: A critique of unfounded skepticism around ai for chip design. *arXiv preprint*, 2024.

Gomes, C. and Shmoys, D. Completing quasigroups or latin squares: A structured graph coloring problem. In *Proceedings of the Computational Symposium on Graph Coloring and Generalizations*, 2002.

Gomes, C. P., Kautz, H., Sabharwal, A., and Selman, B. Satisfiability solvers. *Handbook of Knowledge Representation*, 2008.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024.

Haberlandt, A. and Green, H. Sbva-cadical and sbva-kissat: Structured bounded variable addition. In *Proceedings of SAT Competition 2023*, 2023.

Hagberg, A. and Conway, D. Networkx: Network analysis with python. *URL: https://networkx. github. io*, 2020.

Hamilton, W. L., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Proceedings of Advances in Neural Information Processing Systems NeurIPS*, 2017.

He, F., Liu, T., and Tao, D. Why resnet works? residuals generalize. *IEEE transactions on neural networks and learning systems*, (12), 2020.

Hendrycks, D. and Gimpel, K. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *arXiv preprint*, 2016.

Heule, M. J. H., Järvisalo, M., and Suda, M. SAT competition 2018. *Journal on Satisfiability, Boolean Modeling and Computation*, (1), 2019.

Holland, P. W., Laskey, K. B., and Leinhardt, S. Stochastic blockmodels: First steps. *Social networks*, (2), 1983.

Holme, P. and Kim, B. J. Growing scale-free networks with tunable clustering. *Physical review E*, (2), 2002.

Hopcroft, J. E. and Karp, R. M. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, (4), 1973.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. In *Proceedings of Advances in Neural Information Processing Systems NeurIPS*, 2020a.

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V. S., and Leskovec, J. Strategies for pre-training graph neural networks. In *8th International Conference on Learning Representations, ICLR*, 2020b.

Hu, W., Fey, M., Ren, H., Nakata, M., Dong, Y., and Leskovec, J. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint*, 2021.

Hutter, F., Kotthoff, L., Hoos, H. H., and Leyton-Brown, K. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 2014a.

Hutter, F., López-Ibáñez, M., Fawcett, C., Lindauer, M., Hoos, H. H., Leyton-Brown, K., and Stützle, T. Aclib: A benchmark library for algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization, Revised Selected Papers*, Lecture Notes in Computer Science, 2014b.

Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 2014c.

Ibarz, B., Kurin, V., Papamakarios, G., Nikiforou, K., Bennani, M., Csordás, R., Dudzik, A. J., Bosnjak, M., Vitvitskyi, A., Rubanova, Y., Deac, A., Bevilacqua, B., Ganin, Y., Blundell, C., and Velickovic, P. A generalist neural algorithmic learner. In *Proceedings of the Learning on Graphs Conference*, Proceedings of Machine Learning Research, 2022.

Ilango, R., Loff, B., and Oliveira, I. C. Np-hardness of circuit minimization for multi-output functions. CCC '20, Dagstuhl, DEU, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 9783959771566. doi: 10.4230/LIPIcs.CCC.2020.22. URL https://doi.org/10.4230/LIPIcs.CCC.2020.22.

Iser, A. and Jabs, C. Global benchmark database. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing SAT*, LIPIcs, 2024.

Jabrayilov, A. and Mutzel, P. New integer linear programming models for the vertex coloring problem. In *Proceedings of LATIN: Theoretical Informatics: Latin American Symposium*, 2018.

Kahn, A. B. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, 1962. doi: 10.1145/368996.369025. URL https://doi.org/10.1145/368996.369025.

Kaiser, L. and Sutskever, I. Neural gpus learn algorithms. In *Proceedings of the International Conference on Learning Representations ICLR*, 2016.

Karalias, N. and Loukas, A. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *Proceedings of Advances in Neural Information Processing Systems NeurIPS*, 2020.

Karp, R. M. Reducibility among combinatorial problems. In *Complexity of Computer Computations*. 1972.

Kautz, H. and Selman, B. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1996.

Keisler, R. Forecasting global weather with graph neural networks. *arXiv preprint*, 2022.

Kerschke, P., Kotthoff, L., Bossek, J., Hoos, H. H., and Trautmann, H. Leveraging TSP solver complementarity through machine learning. *Evolutionary Computation*, (4), 2018.

Korte, B. and Vygen, J. *Combinatorial Optimization: Theory and Algorithms*. 5th edition, 2012.

Kotthoff, L. LLAMA: leveraging learning to automatically manage algorithms. *arXiv preprint*, 2013.

Kou, L., Markowsky, G., and Berman, L. A fast algorithm for steiner trees. *Acta informatica*, (2), 1981.

Kruskal, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, (1), 1956.

Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., Ravuri, S., Ewalds, T., Eaton-Rosen, Z., Hu, W., Merose, A., Hoyer, S., Holland, G., Vinyals, O., Stott, J., Pritzel, A., Mohamed, S., and Battaglia, P. Learning skillful medium-range global weather forecasting. *Science*, (6677), 2023.

Lamm, S., Sanders, P., Schulz, C., Strash, D., and Werneck, R. F. Finding near-optimal independent sets at scale. In *Proceedings of the Workshop on Algorithm Engineering and Experiments, ALENEX*, 2016.

Leskovec, J. and Sosic, R. Snap: A general purpose network analysis and graph mining library. *arXiv preprint*, 2016.

Lewis, R. and Palmer, G. Gcol: A high-performance python library for graph colouring. *The Journal of Open Source Software*, (108), 2025.

Li, H., Peng, C., Xu, C., and Yang, Z. Open-book neural algorithmic reasoning. In *Proceedings of Advances in Neural Information Processing Systems NeurIPS*, 2024a.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18:185:1–185:52, 2017. URL https://jmlr.org/papers/v18/16-558.html.

Li, M., Shitole, V., Chien, E., Man, C., Wang, Z., Zhang, Y., Krishna, T., and Li, P. Layerdag: A layerwise autoregressive diffusion model for directed acyclic graph generation. In *Proceedings of the Will Synthetic Data Finally Solve the Data Access Problem? workshop, adjunct to ICLR*, 2025a.

Li, S., Li, C.-M., Luo, M., Coll, J., Cherif, M. S., Habet, D., and Manya, F. Esa solvers, kissat mab binary and amsat in sat competition 2024. In *Proceedings of SAT Competition 2024*, 2024b.

Li, W., Li, R., Ma, Y., Chan, S. O., Pan, D., and Yu, B. Rethinking graph neural networks for the graph coloring problem. *arXiv preprint*, 2022.

Li, X., Li, X., Chen, L., Zhang, X., Yuan, M., and Wang, J. Circuit transformer: A transformer that preserves logical equivalence. In *Proceedings of the International Conference on Learning Representations ICLR*, 2025b.

Li, Z., Guo, J., and Si, X. G4satbench: Benchmarking and advancing SAT solving with graph neural networks. *Transactions on Machine Learning Research*, 2024c.

Lindauer, M., van Rijn, J. N., and Kotthoff, L. The algorithm selection competitions 2015 and 2017. *Artificial Intelligence*, 2019.

Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., and Hutter, F. SMAC3: A versatile bayesian optimization package for hyperparameter optimization. *J. Mach. Learn. Res.*, 2022.

Liu, J., Zhang, J., and Sun, Y. Kissat_mab-dc in sat competition 2024. In *Proceedings of SAT Competition 2024*, 2024.

Lu, Y.-C., Pentapati, S. S. K., Zhu, L., Samadi, K., and Lim, S. K. Tp-gnn: A graph neural network framework for tier partitioning in monolithic 3d ics. *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 2020.

Luo, Y., Shi, L., and Wu, X.-M. Can classic GNNs be strong baselines for graph-level tasks? simple architectures meet excellence. In *Forty-second International Conference on Machine Learning*, 2025.

Mahdavi, S., Swersky, K., Kipf, T., Hashemi, M., Thrampoulidis, V., and Liao, R. Towards better out-of-distribution generalization of neural algorithmic reasoning tasks. *Transactions on Machine Learning Research*, 2023.

Mallappa, U., Mostafa, H., Galkin, M., Phielipp, M., and Majumdar, S. Floorset-a vlsi floorplanning dataset with design constraints of real-world socs. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design ICCAD*, 2024.

McConaghy, T., Palmers, P., Steyaert, M., and Gielen, G. G. E. Trustworthy genetic programming-based synthesis of analog circuit topologies using hierarchical domain-specific building blocks. *IEEE Transactions on Evolutionary Computation*, 2011.

Min, Y., Wenkel, F., Perlmutter, M., and Wolf, G. Can hybrid geometric scattering networks help solve the maximum clique problem? In *Proceedings of Advances in Neural Information Processing Systems NeurIPS*, 2022.

Minder, J., Grötschla, F., Mathys, J., and Wattenhofer, R. SALSA-CLRS: A sparse and scalable benchmark for algorithmic reasoning. In *The Second Learning on Graphs Conference*, 2023.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E. M., Wang, S., Lee, Y., Johnson, E., Pathak, O., Nazi, A., Pak, J., Tong, A., Srinivasa, K., Hang, W., Tuncer, E., Le, Q. V., Laudon, J., Ho, R., Carpenter, R., and Dean, J. A graph placement methodology for fast chip design. *Nature*, (7862), 2021.

Mishchenko, A. and Miyasaka, Y. Problems and results of iwls 2023 programming contest, 2023.

Mishchenko, A., Chatterjee, S., Jiang, R., and Brayton, R. K. Fraigs: A unifying representation for logic synthesis and verification. Technical report, ERL Technical Report, 2005.

Mizuno, K. and Nishihara, S. Toward systematic generation of 3col instances based on minimal unsolvable structures. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, (12), 2007.

Mizuno, K. and Nishihara, S. Constructive generation of very hard 3-colorability instances. *Discrete Applied Mathematics*, (2), 2008.

Močkus, J. *Bayesian approach to global optimization: theory and applications*, volume 37. 1989.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. TUDataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint*, 2020.

Moshiri, N. The dual-barab\'asi-albert model. *arXiv preprint*, 2018.

Müller, L. and Morris, C. Aligning transformers with weisfeiler-leman. In *Proceedings of the International Conference on Machine Learning ICML*, 2024.

Müller, L., Kusuma, D., Bonet, B., and Morris, C. Towards principled graph transformers. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Nath, A. and Kuhnle, A. A benchmark for maximum cut: Towards standardization of the evaluation of learned heuristics for combinatorial optimization. *Transactions on Machine Learning Research*, 2024.

Nejati, S., Liang, J. H., Gebotys, C. H., Czarnecki, K., and Ganesh, V. Adaptive restart and cegar-based solver for inverting cryptographic hash functions. In *Proceedings of the International Conference on Verified Software. Theories, Tools, and Experiments VSTTE, Revised Selected Papers*, Lecture Notes in Computer Science, 2017.

Nenzi, P. and Vogt, H. *Ngspice User's Manual, Version 23*, 2011. URL https://pkgs.fedoraproject.org/repo/extras/ngspice/ngspice23-manual.pdf/eb0d68eb463a41a0571757a00a5b9f9d/ngspice23-manual.pdf. Accessed: 2025-09-23.

Newman, M. E. J. *Networks: An Introduction*. 2010. ISBN 9780199206650.

Newman, M. E. J. and Watts, D. J. Renormalization group analysis of the small-world network model. *Physics Letters A*, (4-6), 1999.

Nudelman, E., Leyton-Brown, K., Hoos, H. H., Devkar, A., and Shoham, Y. Understanding Random SAT: Beyond the Clauses-to-Variables Ratio. In *Proceedings of International Conference on Principles and Practice of Constraint Programming CP*, 2004.

Oskarsson, J., Landelius, T., Deisenroth, M. P., and Lindsten, F. Probabilistic weather forecasting with hierarchical graph neural networks. In *Proceedings of Advances in Neural Information Processing Systems NeurIPS*, 2024.

Ou, S., Yang, Q., and Liu, J. The global production pattern of the semiconductor industry: an empirical research based on trade network. *Humanities and Social Sciences Communications*, 11(1):750, 2024. ISSN 2662-9992. doi: 10.1057/s41599-024-03253-5. URL https://doi.org/10.1057/s41599-024-03253-5.

Paschos, V. T. *Applications of combinatorial optimization*. 2014.

Price, I., Sanchez-Gonzalez, A., Alet, F., Andersson, T. R., El-Kadi, A., Masters, D., Ewalds, T., Stott, J., Mohamed, S., Battaglia, P., Lam, R., and Willson, M. Gencast: Diffusion-based ensemble forecasting for medium-range weather. *arXiv preprint*, 2023.

Qiu, R., Sun, Z., and Yang, Y. DIMES: A differentiable meta solver for combinatorial optimization problems. In *Proceedings of Advances in Neural Information Processing Systems NeurIPS*, 2022.

Rai, S., Neto, W. L., Miyasaka, Y., Zhang, X., Yu, M., Yi, Q., Fujita, M., Manske, G. B., Pontes, M. F., Da Rosa, L. S., et al. Logic synthesis meets machine learning: Trading exactness for generalization. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1026–1031. IEEE, 2021.

Ramar, K. and Mirnalinee, T. A semantic web for weather forecasting systems. In *Proceedings of the International Conference on Recent Trends in Information Technology*. IEEE, 2014.

Rasp, S., Dueben, P. D., Scher, S., Weyn, J. A., Mouatadid, S., and Thuerey, N. Weatherbench: a benchmark data set for data-driven weather forecasting. *Journal of Advances in Modeling Earth Systems*, (11), 2020.

Rasp, S., Hoyer, S., Merose, A., Langmore, I., Battaglia, P., Russell, T., Sanchez-Gonzalez, A., Yang, V., Carver, R., Agrawal, S., Chantry, M., Bouallegue, Z. B., Dueben, P., Bromberg, C., Sisk, J., Barrington, L., Bell, A., and Sha, F. Weatherbench 2: A benchmark for the next generation of data-driven global weather models. *Journal of Advances in Modeling Earth Systems*, (6), 2024.

Reeves, J. E. and Bryant, R. E. Preprocessors prelearn and reencode entering the sat competition 2023. In *Proceedings of SAT Competition 2023*, 2023.

Reichstein, M., Camps-Valls, G., Stevens, B., Jung, M., Denzler, J., Carvalhais, N., and Prabhat, F. Deep learning and process understanding for data-driven earth system science. *Nature*, (7743), 2019.

Rice, J. R. The algorithm selection problem. *Advances in Computers*, 1976.

Rodionov, G. and Prokhorenkova, L. Neural algorithmic reasoning without intermediate supervision. In *Proceedings of Advances in Neural Information Processing Systems NeurIPS*, 2023.

Rossi, R. A. and Ahmed, N. K. The network data repository with interactive graph analytics and visualization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015.

Roussel, O. Controlling a solver execution with the runsolver tool. *Journal on Satisfiability, Boolean Modeling and Computation*, (4), 2011.

Rozemberczki, B., Allen, C., and Sarkar, R. Multi-scale attributed node embedding. *ArXiv preprint*, 2021.

Sanokowski, S., Hochreiter, S., and Lehner, S. A diffusion model framework for unsupervised neural combinatorial optimization. In *Proceedings of the International Conference on Machine Learning ICML*, 2024.

Sato, R., Yamada, M., and Kashima, H. Approximation ratios of graph neural networks for combinatorial problems. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Satopaa, V., Albrecht, J. R., Irwin, D. E., and Raghavan, B. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *Proceedings of the IEEE International Conference on Distributed Computing Systems ICDCS Workshops*, 2011.

Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., and Dill, D. L. Learning a SAT solver from single-bit supervision. In *Proceedings of the International Conference on Learning Representations ICLR*, 2019.

Shavit, H. Algorithm selection for SAT using graph neural networks. Master's thesis, Leiden University, 2023.

Shavit, H. and Hoos, H. H. Revisiting SATzilla Features in 2024. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing SAT*, 2024.

Shi, Z., Pan, H., Khan, S., Li, M., Liu, Y., Huang, J., Zhen, H.-L., Yuan, M., Chu, Z., and Xu, Q. Deepgate2: Functionality-aware circuit representation learning. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2023.

Stoll, T., Müller, L., and Morris, C. Generalizable insights for graph transformers in theory and practice. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Sun, H., Guha, E. K., and Dai, H. Annealed training for combinatorial optimization on graphs. *arXiv preprint*, 2022.

Sun, Z. and Yang, Y. DIFUSCO: graph-based diffusion solvers for combinatorial optimization. In *Proceedings of Advances in Neural Information Processing Systems NeurIPS*, 2023.

Tchinda, R. K. and Djamegni, C. T. hkis, ukissatinc, pakisinc and pahkis in the sat competition 2023. In *Proceedings of SAT Competition 2023*, 2023.

Toenshoff, J., Ritzert, M., Wolf, H., and Grohe, M. Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 2021.

Tönshoff, J. and Grohe, M. Learning from algorithm feedback: One-shot SAT solver guidance with gnns. *ArXiv preprint*, 2025.

Tönshoff, J., Kisin, B., Lindner, J., and Grohe, M. One model, any CSP: graph neural networks as fast global search heuristics for constraint satisfaction. In *Proceedings of the International Joint Conference on Artificial Intelligence IJCAI*, 2023.

Tsaras, D., Li, X., Chen, L., Xie, Z., and Yuan, M. Elf: Efficient logic synthesis by pruning redundancy in refactoring. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7. IEEE, 2025.

Tseitin, G. S. On the complexity of derivation in propositional calculus. In *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*. 1983.

Velickovic, P., Buesing, L., Overlan, M. C., Pascanu, R., Vinyals, O., and Blundell, C. Pointer graph networks. In *Proceedings of Advances in Neural Information Processing Systems NeurIPS*, 2020a.

Velickovic, P., Ying, R., Padovano, M., Hadsell, R., and Blundell, C. Neural execution of graph algorithms. In *Proceedings of the International Conference on Learning Representations ICLR*, 2020b.

Velickovic, P., Badia, A. P., Budden, D., Pascanu, R., Banino, A., Dashevskiy, M., Hadsell, R., and Blundell, C. The CLRS algorithmic reasoning benchmark. In *Proceedings of the International Conference on Machine Learning ICML*, Proceedings of Machine Learning Research, 2022.

Vignac, C., Krawczuk, I., Siraudin, A., Wang, B., Cevher, V., and Frossard, P. Digress: Discrete denoising diffusion for graph generation. In *Proceedings of the International Conference on Learning Representations ICLR*, 2023.

Wang, H., Wang, K., Yang, J., Shen, L., Sun, N., Lee, H.-S., and Han, S. Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 2020.

Wang, W., Hu, Y., Tiwari, M., Khurshid, S., McMillan, K. L., and Miikkulainen, R. Neuroback: Improving CDCL SAT solving using graph neural networks. In

Proceedings of the International Conference on Learning Representations ICLR, 2024.

Wenkel, F., Cantürk, S., Horoi, S., Perlmutter, M., and Wolf, G. Towards a general recipe for combinatorial optimization with multi-filter GNNs. In *Proceedings of the Learning on Graphs Conference*, 2024.

Weyn, J. A., Durran, D. R., and Caruana, R. Improving data-driven global weather prediction using deep convolutional neural networks on a cubed sphere. *Journal of Advances in Modeling Earth Systems*, (9), 2020.

Wille, R., Große, D., Teuber, L., Dueck, G. W., and Drechsler, R. RevLib: An online resource for reversible functions and reversible circuits. In *Proceedings of the International Symposium on Multiple-Valued Logic*, 2008. RevLib is available at http://www.revlib.org.

Wong, F., Zheng, E. J., Valeri, J. A., Donghia, N. M., Anahtar, M. N., Omori, S., Li, A., Cubillos-Ruiz, A., Krishnan, A., Jin, W., Manson, A. L., Friedrichs, J., Helbig, R., Hajian, B., Fiejtek, D. K., Wagner, F. F., Soutter, H. H., Earl, A. M., Stokes, J. M., Renner, L. D., and Collins, J. J. Discovery of a structural class of antibiotics with explainable deep learning. *Nature*, 2023.

Xia, L. and Huang, C. Anygraph: Graph foundation model in the wild. *arXiv preprint arXiv:2408.10700*, 2024.

Xu, K., Boussemart, F., Hemery, F., and Lecoutre, C. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence*, (8), 2007. ISSN 0004-3702.

Xu, K., Li, J., Zhang, M., Du, S. S., Kawarabayashi, K., and Jegelka, S. What can neural networks reason about? In *Proceedings of the International Conference on Learning Representations ICLR*, 2020.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. Satzilla: Portfolio-based Algorithm Selection for SAT. *Journal of Artificial Intelligence Research*, 2008.

Xu, L., Hutter, F., Shen, J., Hoos, H. H., and Leyton-Brown, K. Satzilla2012: Improved algorithm selection based on cost-sensitive classification models. *Proceedings of SAT Challenge*, 2012.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the International Conference on Machine Learning ICML*, JMLR Workshop and Conference Proceedings, 2016.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*, 2018.

Yu, K., Zhao, H., Huang, Y., Yi, R., Xu, K., and Zhu, C. DISCO: Efficient diffusion solver for large-scale combinatorial optimization problems. *arXiv preprint*, 2024.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Zaremba, W. and Sutskever, I. Learning to execute. *arXiv preprint*, 2014.

Zhang, D., Dai, H., Malkin, N., Courville, A. C., Bengio, Y., and Pan, L. Let the flows tell: Solving graph combinatorial problems with gflownets. In *Proceedings of Advances in Neural Information Processing Systems NeurIPS*, 2023.

Zhang, G., He, H., and Katabi, D. Circuit-gnn: Graph neural networks for distributed circuit design. In *Proceedings of the International Conference on Machine Learning ICML*, Proceedings of Machine Learning Research, 2019.

Zhang, Y., Ren, H., and Khailany, B. Grannite: Graph neural network inference for transferable power estimation. *ACM/IEEE Design Automation Conference (DAC)*, 2020.

Zhang, Z., Chételat, D., Cotnareanu, J., Ghose, A., Xiao, W., Zhen, H., Zhang, Y., Hao, J., Coates, M., and Yuan, M. Grass: Combining graph neural networks with expert knowledge for SAT solver selection. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD*, 2024.

Zhao, Z. and Zhang, L. An automated topology synthesis framework for analog integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

Zheng, Z., Huang, S., Zhong, J., Shi, Z., Dai, G., Xu, N., and Xu, Q. Deepgate4: Efficient and effective representation learning for circuit design at scale. In *Proceedings of the International Conference on Learning Representations ICLR*, 2025.

A. The GRAPHBENCH framework

In this section, we provide information on the GRAPHBENCH framework and its proposed expansion. Furthermore, we provide examples for adding new datasets, evaluation procedures, or optimization procedures, as well as guidance on using GRAPHBENCH in a custom model pipeline.

A.1. Future of GRAPHBENCH

We consider GRAPHBENCH a continuously expanded and updated benchmarking framework, with new domains and benchmarks added after the initial release. Since graph learning data spans many more domains than our initial dataset collection, we provide this underlying framework to enable the addition of additional datasets and domains. In addition, each domain provides extension capabilities, as larger datasets can be generated from the underlying generation procedures used for the provided datasets.

As shown in Figure 3, the GRAPHBENCH framework is built around dataset loading, evaluation, and optimization. Each part can be modified independently, allowing the use of only parts of GRAPHBENCH in custom models or the replacement of a proposed evaluation pipeline. Moreover, the GRAPHBENCH framework aims to be lightweight, relying only on essential packages to provide functionality.

Furthermore, GRAPHBENCH is modular by design, allowing collaborators and community members to update parts of GRAPHBENCH and expand it with additional datasets or evaluation tasks. Since each evaluator and dataset handler is self-contained and can be used with its corresponding dataset without requiring user implementation, the framework's expansion is significantly streamlined.

Beyond datasets and evaluation methods, we plan to provide additional scripts for GNN and graph transformer baselines, enabling replication and further research on the provided datasets. We present the framework structure along with a proposed expansion in Figure 3.

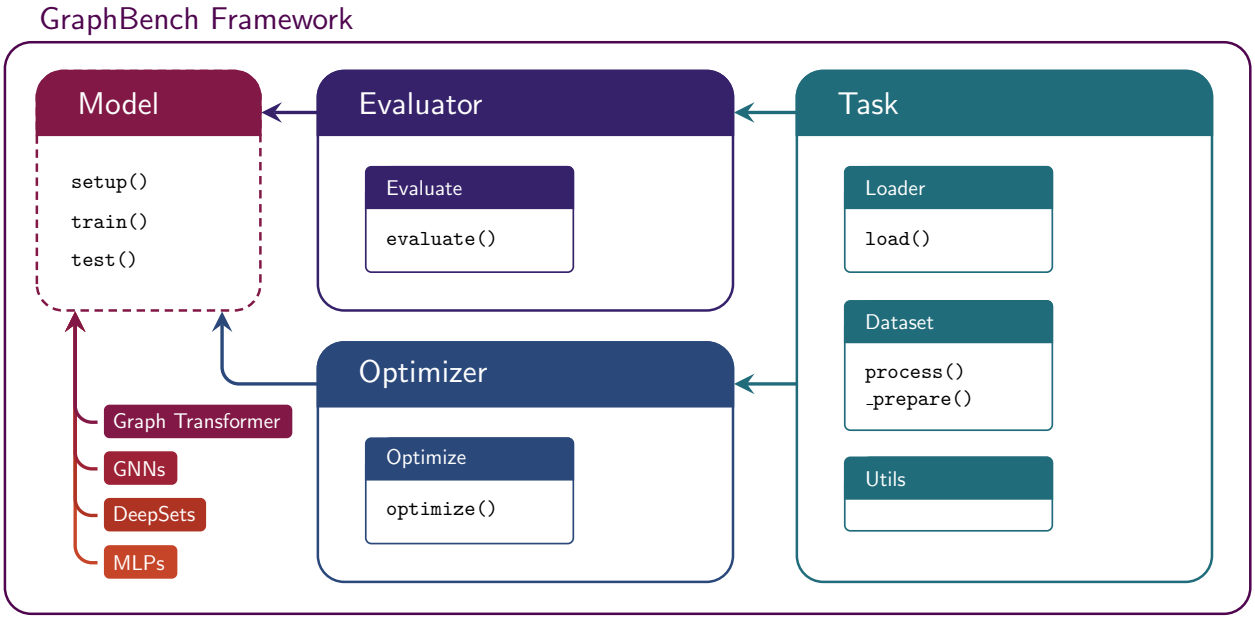

Figure 3. Modular design of GraphBench. Each top-level header includes one foundational part of GRAPHBENCH with methods and functions outlined below. *Model*, shown with a dashed line, constitutes a planned addition to the framework.

A.2. Example usage of GRAPHBENCH

We provide a unified and easy-to-use interface for GRAPHBENCH datasets. Inspired by benchmarks such as Open Graph Benchmark (OGB) (Hu et al., 2020a), we use wrapper methods to handle dataset loading. Unlike competing benchmarks,

however, GRAPHBENCH offers a single loading class (`graphbench.Loader`) for all datasets, independent of task type. Each wrapper includes the necessary precomputations and comes with predefined splits, enabling direct use in downstream tasks. Nonetheless, we offer customization options for each dataset integrated in the loader method. For seamless integration, all datasets are fully compatible with PYTORCH GEOMETRIC through its `InMemoryDataset` interface. We provide GRAPHBENCH as a Python package distributed via PyPi, enabling continuous updates to code and datasets.

Beyond loading datasets, GRAPHBENCH provides utilities for experimentation. Given a model, hyperparameter optimization can be run directly via `graphbench.Optimizer`, which implements the optimization algorithms described in Appendix E. For evaluation, `graphbench.Evaluator` loads the corresponding metrics for each selected dataset. Since specific datasets require specialized architectures (e.g., for graph generation), we also supply dataset information and usage guidelines in our code repository. Listing 1 presents pseudocode for training a predefined model using GRAPHBENCH.

```
import graphbench

model = #your torch model
dataset_name = #name of the task or list of tasks
root = #directory where datasets will be stored
pre_filter = #PyTorch Geometric filter matrix
pre_transform = #PyTorch Geometric-like transform during loading
transform = #PyTorch Geometric-like transform at computation time

#Setting up the components of graphbench
evaluator = graphbench.Evaluator(dataset_name)
optimizer = graphbench.Optimizer(optimization_args, training_method)
loader = graphbench.Loader(root, dataset_name, pre_filter, pre_transform, transform)

#load a GraphBench dataset and get splits
dataset = loader.load()

#optimize your model
opt_model = optimizer.optimize()

#use graphbench evaluator with targets y_true and predictions y_pred
results = evaluator.evaluate(y_true, y_pred)
```

Listing 1. PYTORCH-like pseudocode for GRAPHBENCH showcasing the usage of the dataset loader and application to downstream tasks.

```
#adding your own evaluator
#graphbench/evaluator.py
class Evaluator():
...
#define logic for new evaluator
    def get_new_evaluator():
        ...
        #y_true as target values, y_pred as prediction
        #calculation logic in self.new_evaluator()
        return lambda y_pred, y_true: self.new_evaluator(y_pred, y_true)

#add evaluator to list:
    def _get_metric_from_name(self, metric_name):
        metric_dict = {...
        "new_eval": self.get_new_evaluator()
        }
        ...

#add evaluator to master.csv file to enable loading from Evaluator()
#task_type denotes the type of task for logging purposes
#graphbench/master.csv
new_evaluator,task_type,"new_eval"

#adding your own dataset to the loader
#graphbench/loader.py
class Loader():
...
```

```
    def _loader(self, dataset_name):
        ...
        elif "new_dataset_name" in dataset_name:
            from graphbench.datasets.new_dataset import newDataset
            train_dataset = newDataset(...)
            valid_dataset = newDataset(...)
            test_dataset = newDataset(...)
        ...

#add a new line to the csv containing the dataset name information
#graphbench/datasets.csv
...
new_dataset_name, new_dataset

#adding logic for new datasets
#graphbench/datasets/new_dataset.py
class newDataset(InMemoryDataset):
    #define torch geometric InMemoryDataset init method (also handles downloading)
    def __init__():
        ...
    #prepare dataset and save it to disk
    def _prepare():
        ...
    #optional processing logic, depending on the prepare method
    def process():
        ...
```

Listing 2. PyTorch-like pseudocode for GraphBench on how to add additional datasets or evaluation handlers in the framework.

Moreover, GraphBench also provides access to the data download and processing handlers. As showcased in Listing 2, a new dataset can be easily added by providing download and processing methods, as well as dataset names in the datasets.csv file. Similar instructions hold for the expansion of the evaluation and optimization module. Since optimization is done via SMAC3 (), optimization modules can be expanded using the established framework.

Table 6. Summary of currently available GraphBench datasets for each subdomain.

Subdomain	Name	Node Feat.	Edge Feat.	Directed	Hetero	#Tasks	Split Scheme	Split Ratio	Task Type	Metric
Social networks	BlueSky – quotes	✓	–	✓	–	3	Temporal	(55/15)/15/15	Node regression	MAE / R^2 / Spearman corr.
	BlueSky – replies	✓	–	✓	–	3	Temporal	(55/15)/15/15	Node regression	MAE / R^2 / Spearman corr.
	BlueSky – reposts	✓	–	✓	–	3	Temporal	(55/15)/15/15	Node regression	MAE / R^2 / Spearman corr.
Chip design	AIG	✓	✓	✓	–	1	Fixed	80/10/10	Generation	Ad-hoc Score
Electronic circuits	5 Components	✓	–	–	–	1	Random	70/10/20	Regression	RSE
	7 Components	✓	–	–	–	1	Random	70/10/20	Regression	RSE
	10 Components	✓	–	–	–	1	Random	70/10/20	Regression	RSE
SAT Solving	Small	–	–	–	✓ \ –	5	Fixed	80/10/10	Classification / Regression	Closed Gap / RMSE
	Medium	–	–	–	✓ \ –	5	Fixed	80/10/10	Classification / Regression	Closed Gap / RMSE
	Large	–	–	–	✓ \ –	5	Fixed	80/10/10	Classification / Regression	Closed Gap / RMSE
Combinatorial optimization	RB – Small	–	–	–	–	6	Fixed	70/15/15	Regression / Unsupervised	MAE / CO
	RB – Large	–	–	–	–	6	Fixed	70/15/15	Regression / Unsupervised	MAE / CO
	ER – Small	–	–	–	–	6	Fixed	70/15/15	Regression / Unsupervised	MAE / CO
	ER – Large	–	–	–	–	6	Fixed	70/15/15	Regression / Unsupervised	MAE / CO
	BA – Small	–	–	–	–	6	Fixed	70/15/15	Regression / Unsupervised	MAE / CO
	BA – Large	–	–	–	–	6	Fixed	70/15/15	Regression / Unsupervised	MAE / CO
Algorithmic reasoning	Topological Sorting	✓	✓	✓	–	3	Fixed	98/1/1	Node regression	MAE
	MST	–	✓	–	–	3	Fixed	98/1/1	Edge classification	Accuracy / F1
	Bridges	–	–	–	–	3	Fixed	98/1/1	Edge classification	Accuracy / F1
	Steiner Trees	✓	✓	–	–	3	Fixed	98/1/1	Edge classification	Accuracy / F1
	Max Clique	–	–	–	–	3	Fixed	98/1/1	Node classification	Accuracy / F1
	Flow	✓	✓	✓	–	3	Fixed	98/1/1	Regression	MAE
	Max Matching	–	✓	–	–	3	Fixed	98/1/1	Edge classification	Accuracy / F1
Weather forecasting	ERA5 64x32	✓	✓	✓ \ –	–	1	Temporal	86/5/9	Node regression	MSE

B. Subdomain Details

Here, we provide additional details on the subdomains and their datasets.

Table 7. Statistics of currently-available GRAPHBENCH datasets.

Subdomain	Name	#Graphs	Avg. #Nodes	Avg. #Edges	Avg. Deg.
Social networks	BlueSky – quotes	1	289 136 / 286 425 / 311 272	3 075 967 / 3 815 996 / 4 525 872	10.64 / 13.32 / 14.54
	BlueSky – replies	1	470 816 / 464 867 / 569 424	9 287 565 / 10 769 386 / 12 318 196	19.73 / 23.17 / 21.63
	BlueSky – reposts	1	498 012 / 508 818 / 580 112	12 049 951 / 14 658 114 / 17 261 865	24.20 / 28.81 / 29.76
Chip design	AIG	1 200 000	125.9	236.3	3.7
Electronic circuits	5 Components	73 000	13.0	30.0	7.0
	7 Components	14 000	17.0	42.0	9.3
	10 Components	6 000	24.0	56.0	12.3
SAT Solving	Small – VG	69 596	1 323.96	38 791.52	33.9
	Small – VCG	69 596	8 304.22	45 402.51	5.4
	Small – LCG	69 596	9 628.12	48 050.44	4.75
Combinatorial optimization	RB – Small	50 000	241.0	9 016.1	36.9
	RB – Large	50 000	1 027.8	102 644.8	99.4
	ER – Small	50 000	249.9	9 463.5	37.3
	ER – Large	50 000	750.2	84 450.1	112.4
	BA – Small	50 000	249.9	991.6	3.9
	BA – Large	50 000	750.1	2 992.4	3.9
Algorithmic reasoning-EASY	Topological Sorting	1 010 000	16.0 / 16.0 /128.0	41.006 / 40.997 / 1121.008	2.563 / 2.562 / 8.758
	MST	1 010 000	16.0 / 16.0 /128.0	24.024 / 24.004 / 1147.972	1.5015/ 1.5003 / 8.9685
	Bridges	1 010 000	16.0 / 16.0 /128.0	20.754/ 20.794/ 277.08	1.2971 / 1.2996 / 2.1647
	Steiner Trees	1 010 000	16.0 / 16.0 /128.0	28.739 / 28.733 / 755.284	1.796 / 1.796 / 5.901
	Max Clique	1 010 000	16.0 / 16.0 /128.0	77.99 / 77.95 / 2698.5	4.875 / 4.872 / 21.081
	Flow	1 010 000	16.0 / 16.0 /64.0	75.76 / 75.84 / 530.96	4.734 / 4.740 / 8.296
	Max Matching	1 010 000	16.0 /16.0 / 128.0	38.890 / 38.897 / 746.453	2,431 / 2,431 / 5,834
Algorithmic reasoning-MEDIUM	Topological Sorting	1 000 000	16.0 / 16.0 /128.0	36.167 / 36.137 / 1097.05	2.260 / 2.259 / 8.571
	MST	1 010 000	16.0 / 16.0 /128.0	23.976 / 24.003 / 1140.312	1.498/ 1.50 / 8.909
	Bridges	1 010 000	16.0 / 16.0 /128.0	17.539/ 17.54/ 277.550	1.0962 / 1.0963 / 2.168
	Steiner Trees	1 010 000	16.0 / 16.0 /128.0	19,549 / 19.562 /742.301	1.222 / 1,223 / 5.799
	Max Clique	1 010 000	16.0 / 16.0 /128.0	108.0 / 108.0 / 2687.22	6.751 / 6.751 / 20.994
	Flow	1 010 000	16.0 / 16.0 /64.0	42.36 / 42.36 / 645.20	2.648 / 2.648 / 10.08
	Max Matching	1 010 000	16.0 / 16.0 /128.0	16.305 / 16.306 / 726,908	1,0189/ 1,020 /5,679
Algorithmic reasoning-HARD	Topological Sorting	1 010 000	16.0 / 16.0 /128.0	36.143 / 36.142 / 827.09	2.259 / 2.259 / 6.462
	MST	1 010 000	16.0 /16.0 /128.0	24.008/ 23.998/ 978.825	1,50 / 1.499 / 7.647
	Bridges	1 010 000	16.0 /16.0 /128.0	17.539/ 17.54/ 277.08	1.096 / 1.096 / 2.165
	Steiner Trees	1 010 000	16.0 /16.0 /128.0	19.583 / 19.565 / 663.547	1.224 /1,223 / 5.184
	Max Clique	1 010 000	16.0 / 16.0 /128.0	108.01 / 107.99 / 1812.04	6.751 / 6.750 / 14.157
	Flow	1 010 000	16.0 / 16.0 /64.0	42.36 / 42.36 / 645.20	2.648 / 2.648 / 10.08
	Max Matching	1 010 000	16.0 /16.0 / 128.0	16,303 / 16,304 / 746,521	1,018/ 1,019 / 5,832
Weather forecasting	ERA5 64x32	93 544	4 610	59 667	12.9

B.1. Social networks: Predicting engagements on BlueSky

Related work A wide range of social network benchmarks are available in repositories such as SNAP (Leskovec & Sosic, 2016) and TUDATASETS (Morris et al., 2020), yet these datasets diverge substantially from real-world social networks. Benchmarks such as FACEBOOK, GITHUB, TWITCH, LASTFM, DEEZER, and REDDIT (Rozemberczki et al., 2021; Hamilton et al., 2017) illustrate the main issues: (1) interactions are collapsed into static graphs, eliminating temporal dynamics; (2) prediction targets are categorical (e.g., user type, demographic attribute) rather than continuous behavioral outcomes; (3) topologies are simplified to undirected friendships or follows, omitting richer relations such as replies, reposts, or pull requests; (4) node features are weak or outdated, often sparse indicators or embeddings from GloVe/Word2Vec; and (5) preprocessing alters network structure in disruptive and biasing manners, for example by removing large communities in REDDIT. The TUDATASET collection shares similar limitations. Their REDDIT variants represent discussion threads as featureless graphs with reply edges and categorical thread-level labels, while COLLAB and IMDB construct ego-networks labeled by field or genre. These datasets are likewise static, categorical, topologically restricted, feature-poor, and further constrained by artificial graph boundaries that remove long-range dependencies.

The recent GRAPHLAND benchmark (Bazhenov et al., 2025) represents a substantial step toward more realistic social-network evaluation with the comprised POKEC-REGIONS, TWITCH-VIEWS, and the ARTNET datasets. We note that these include temporal splits and the prediction of continuous node-wise targets, such as view counts for digital content; however, they consider static and, in most cases, undirected friendship- or follow-based edge connectivity.

With respect to these previously proposed datasets, our datasets are designed to support more realistic forecasting, with temporal splits, continuous, forward-looking, engagement-based targets, time-evolving, directed, interaction-based topologies, and node features derived from user-generated post content using modern language-model embeddings. Finally, our datasets do not undergo *arbitrary* subsampling.

More on metrics We assess model performance using three metrics: the Mean Absolute Error (MAE), the coefficient of determination (R^2), and the Spearman correlation (σ). As for R^2, given a set of evaluation nodes $U \subset V$, reference engagement kind κ and prediction interval $\tau_{1,2}$, it is defined as

$$R^2_{\kappa,\tau_{1,2}} \coloneqq 1 - \frac{\sum_{u \in U} (y^{\kappa,(u)}_{\tau_{1,2}} - \hat{y}^{\kappa,(u)})^2}{\sum_{u \in U} (y^{\kappa,(u)}_{\tau_{1,2}} - \bar{y}^{\kappa}_{\tau_{1,2}})^2},$$

where $\hat{y}^{\kappa,(u)}$ is the model's prediction for engagement κ and user u and $\bar{y}^{\kappa}_{\tau_{1,2}}$ is the mean target value over U. This standard regression metric indicates the overall variance captured by the model, enabling assessment of its predictive accuracy relative to that of a trivial predictor. The Spearman correlation metric σ is defined, instead, as:

$$\sigma_{\kappa,\tau_{1,2}} \coloneqq 1 - \frac{6 \cdot \sum_{u \in U} \left(\mathrm{R}[y^{\kappa,(u)}_{\tau_{1,2}}] - \mathrm{R}[\hat{y}^{\kappa,(u)}]\right)^2}{n_U(n_U^2 - 1)},$$

where $n_U = |U|$ and expression $\left(\mathrm{R}[y^{\kappa,(u)}_{\tau_{1,2}}] - \mathrm{R}[\hat{y}^{\kappa,(u)}]\right)$ evaluates the difference in the ranking of user $u \in U$ within either targets ($y^{\kappa}_{\tau_{1,2}}$) or model's predictions ($\hat{y}^{\kappa}$). This measures how well predictions preserve the relative ordering of users by future engagement, and is not necessarily captured by accuracy metrics such as R^2.

More details on the dataset

The BLUESKY dataset curated by Failla & Rossetti (2024), is available from `https://zenodo.org/records/14669616`. More details on our benchmark construction follow:

- *Graph structures.* The dataset contains three graph structures that capture network-wide *interactions* based on quotes, replies, and reposts: a *directed* edge from node u to node v indicates that u quoted, replied to, or reposted v's posts. These interactions are originally timestamped, and we only construct edges from interactions observed in $(t_0, t_1]$. We only consider one single edge when multiple interactions are present in $\tau_{0,1}$ for the same user pair. In future iterations of this dataset, we will consider endowing edges with additional features conveying the amount and time distribution of such interactions.
- *Node features.* Each node corresponds to a user, which we describe with an aggregated representation of the content of their posts in the interval $(t_0, t_1]$; these are obtained by a pretrained language model. For each user, we employ the SENTENCE-TRANSFORMERS/ALL-MINILM-L6-V2 language model to embed the text obtained by concatenating the

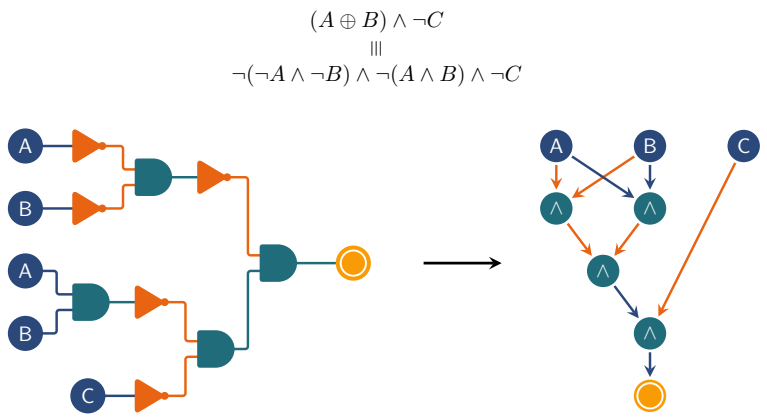

Figure 4. Transformation of a Boolean formula into an And–Inverter Graph (AIG). The left side shows the logic network with explicit gates and inversions; the right side shows the corresponding AIG representation using only AND nodes (teal) and negated edges (orange).

content of each of their posts at the monthly granularity.[3] We subsequently aggregate these (by averaging) over the time interval $(t_0, t_1]$.

- *Node targets.* These are calculated as the *median* number of engagements obtained by the user's posts ($y^{\kappa,(u)}_{\tau_{A,B}} = \text{median}(E^{\kappa,(u)}_{\tau_{A,B}})$). These are considered as separate prediction targets and are measured in the time interval $(t_1, t_2]$, i.e., posterior to observed interactions and posts. We observed that the median is a more robust statistic than, e.g., the mean, in this context. We apply a logarithmic transformation to reduce skew in these prediction targets.
- *Splitting procedure.* The splitting strategy ensures that models are trained exclusively on past information and evaluated on later interactions. The posts considered in deriving node features and targets span the time interval from February 17th, 2023 (t_{start}) to March 18th, 2024 (t_{end}). We consider three key time points: t_{A}, t_{B}, t_{C}. These are obtained in a way that the proportion of posts in $\tau_{\text{start},A}, \tau_{A,B}, \tau_{B,C}, \tau_{C,D}$ amounts to, resp., $55\%, 15\%, 15\%, 15\%$. The dataset is then accordingly split into training, validation, and test splits as follows:
  - Training: $t_0 = t_{\text{start}}$, $t_1 = t_{\text{A}}$, $t_2 = t_{\text{B}}$;
  - Validation: $t_0 = t_{\text{start}}$, $t_1 = t_{\text{B}}$, $t_2 = t_{\text{C}}$;
  - Test: $t_0 = t_{\text{start}}$, $t_1 = t_{\text{C}}$, $t_2 = t_{\text{end}}$.

  In particular:
  - t_A: December 11th, 2023;
  - t_B: January 22nd, 2024;
  - t_C: February 20th, 2024.

  While ensuring the model is not trained on post contents and network interactions from the future, by setting $t_0 = t_{\text{start}}$ in all splits this partitioning strategy also reflects a realistic scenario where a social network "grows in time," in the sense that user representations evolve as they generate new content and their connections expand as they interact with more users.

B.2. Hardware design

B.2.1. Chip design: Learning to generate small circuits

Chip design is among the most complex engineering challenges today, with significant implications for the global economy and supply chains (Ou et al., 2024). Fabricating a competitive system-on-chip requires arranging billions of transistors under strict performance, power, and area constraints. This process is orchestrated by *electronic design automation* (EDA) tools, which automate key stages from high-level synthesis to physical layout. Yet, as design scale and complexity continue to grow, bottlenecks in design space exploration and functional verification are becoming increasingly severe.

[3] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Logic synthesis is a fundamental step within the EDA flow, translating a behavioral specification into a structural implementation of interconnected logic gates (Rai et al., 2021). The resulting Boolean circuit provides the core representation of a system's logic. Optimizing this circuit, typically by minimizing its size (area), is a well-known NP-hard problem (Ilango et al., 2020). For decades, progress has relied on carefully engineered heuristics and combinatorial optimization techniques embedded in EDA tools. However, these classical approaches are reaching diminishing returns, and their computational costs scale poorly with modern circuit complexity (Tsaras et al., 2025).

Recently, machine learning has begun to influence key stages of chip design, including macro placement (Mirhoseini et al., 2021) and floor planning (Mallappa et al., 2024). In this work, we propose focusing these efforts on logic synthesis, framing it as a conditional graph generation problem: generating a circuit that correctly implements a given Boolean function while minimizing the number of gates. The goal is thus to produce circuits that satisfy a prescribed truth table exactly while optimizing structural efficiency.

A central challenge in Boolean circuit synthesis is ensuring optimized circuits remain functionally equivalent to the original. Regardless of transformations that reduce gate count, delay, power, or area, the outputs must match for all inputs. This makes the task delicate—small changes can alter behavior. Finding a minimal circuit is NP-hard, making synthesis a core computational challenge. Classical tools such as ABC (Brayton & Mishchenko, 2010) use heuristics to efficiently generate near-optimal circuits. Recently, learning-based methods have emerged as alternatives (Mirhoseini et al., 2021; Li et al., 2025b) that use data-driven models to capture structural patterns and generalize across designs. The natural representation of Boolean circuits as directed acyclic graphs makes graph-based generative learning particularly promising.

Formally, let $f\colon \{0,1\}^n \to \{0,1\}^m$ be a Boolean function with n inputs and m outputs. Its behavior is fully specified by a truth table $\boldsymbol{T}^f \in \{0,1\}^{m\times 2^n}$, where each column $\boldsymbol{T}^f_{\boldsymbol{x}}$ corresponds to an input $\boldsymbol{x} \in \{0,1\}^n$ and satisfies $f(\boldsymbol{x}) = \boldsymbol{T}^f_{\boldsymbol{x}}$. A logic circuit $C := (V, E)$ is represented as a *directed acyclic graph* (DAG) of in-degree at most two. The node set is partitioned into input nodes $V^{\text{in}} := \{v_1^{\text{in}}, \dots, v_n^{\text{in}}\}$, gate nodes $V^{\text{gate}} := \{v_1^{\text{gate}}, \dots, v_k^{\text{gate}}\}$, each computing a Boolean function of arity at most two (e.g., AND), and output nodes $V^{\text{out}} := \{v_1^{\text{out}}, \dots, v_m^{\text{out}}\}$, each of in-degree one. The edge set $E \subseteq V \times V \times \{0,1\}$ specifies connections, where a label of 1 denotes signal inversion (NOT) and 0 denotes direct transmission. For an input $\boldsymbol{x} \in \{0,1\}^n$, values are assigned to input nodes, propagated through gate nodes (applying inversion where specified), and collected at output nodes to yield the circuit output $C(\boldsymbol{x}) \in \{0,1\}^m$. Because the combination of AND and NOT is functionally complete, any Boolean function can be represented using only these two operations. We therefore adopt the standard representation of *and-inverter graphs* (AIGs), where circuits are DAGs of AND gates connected by edges that may invert their signals (Mishchenko et al., 2005). AIGs are widely used in industrial logic synthesis and provide a compact, canonical representation of circuits. Figure 4 provides an overview of how the AIG is constructed from a Boolean formula and how it compares to traditional logic gates.

Related work AlphaChip (Mirhoseini et al., 2021) kickstarted efforts to bring the latest machine learning advancements into a relatively conservative area of chip design, applying reinforcement learning to the macro placement task and outperforming human engineers on specific metrics, as well as processing designs for newer generations of TPUs (Goldie et al., 2024). In logic synthesis, considerable work has focused on discriminative objectives (Zheng et al., 2025) rather than generative ones. One of the main benchmarks in the field is the contest organized by the *International Workshop on Logic & Synthesis* (IWLS) (Mishchenko & Miyasaka, 2023), in which participants are asked to optimize 100 preselected circuits. Historically, combinatorial optimization methods prevailed due to the lack of training splits and the limited amount of available data. However, recent machine learning methods have shown promising results by training on large amounts of synthetic data simulated with standard EDA tools.

Description of the learning task Given a set of N Boolean functions, $\{f_i\}_{i=1}^N$, $N \in \mathbb{N}$, $f_i\colon \{0,1\}^{n_i} \to \{0,1\}^{m_i}$, the goal of Boolean circuit synthesis is to construct, for each function, a logic circuit $C_i = (V_i, E_i)$ with n_i input and m_i output nodes that satisfies the following conditions.

1. **Logical correctness (constraint)** For all $\boldsymbol{x} \in \{0,1\}^{n_i}$, the output of the circuit must match the output of f_i, i.e., $C_i(\boldsymbol{x}) = f_i(\boldsymbol{x})$, where $C_i(\boldsymbol{x})$ denotes the vector of output values produced by the circuit on input $\boldsymbol{x}$. In that case, C_i is said to be *logically equivalent* to f_i, which we denote $C_i \equiv f_i$.
2. **Structural efficiency (objective)** For each f_i, among all circuits $C = (V, E)$ satisfying logical correctness, select a circuit that optimizes a predefined circuit score function $\mathrm{s}(C)$, where $\mathrm{s}\colon \mathcal{C}_{f_i} \to \mathbb{R}_{\geq 0}$, which measures structural properties of the circuit, such as the number of gate nodes,

$$\arg\max_{C \in \mathcal{C}_{f_i}} \mathrm{s}(C),$$

where $\mathcal{C}_{f_i} = \{C \mid C \equiv f_i\}$ is the set of all circuits implementing f_i. The score function should favor AIGs with fewer internal nodes while penalizing incorrect solutions. Therefore, we define score over the set of functions $\{f_i\}_{i=1}^N$ as

$$\text{score}(\tilde{C}) := \frac{100}{N} \sum_{i=1}^{N} \frac{\#C_i^{\text{ref}}}{\#\tilde{C}_i} \cdot \mathbb{1}_{\tilde{C}_i \equiv f_i}, \tag{1}$$

where $\tilde{C}$ is the set of generated AIGs, $\#\tilde{C}_i$ and $\#C_i^{\text{ref}}$ are the number of internal nodes of the i-th generated circuit and of the corresponding baseline AIG generated by ABC, $\mathbb{1}_{\tilde{C}_i \equiv f_i} = 1$ if $\tilde{C}_i$ is logically equivalent to f_i, and 0 otherwise.

With this benchmark, we introduce a new generative task paired with a challenging optimization objective, i.e., models must generate valid tree-like DAGs that not only satisfy a target truth table but also minimize circuit size. This setting provides a rigorous testbed for evaluating the capabilities of generative graph methods under realistic and computationally challenging constraints.

Details on the dataset For this benchmark, we require paired datasets of truth tables and their corresponding AIGs to train generative models. To construct such a dataset, we first generated 1 200 000 random truth tables in the PLA format (Wille et al., 2008), each with 6-8 inputs and 1 or 2 outputs. These truth tables were then compiled into optimized AIGs in the AIGER format (Biere et al., 2011) using ABC. During this step, we applied a sequence of standard optimization commands (`strash`, `resyn2`, `resyn2`, `dc2`, `resyn2rs`, `resyn2`). We subsequently converted the AIGs into DOT format (Gansner et al., 2006) via the AIGER library, from which they can be conveniently transformed into PyTorch Geometric graphs. Alongside these graph representations, we store the truth tables explicitly as matrices of size $m \times 2^n$, where n is the number of inputs and m the number of outputs. We split the dataset into training, validation, and test sets at 80/10/10. Details concerning the dataset are presented in Table 8.

Table 8. Statistics of the synthetic AIG dataset.

#Graphs	#Inputs	#Outputs	Mean #Nodes	Median #Nodes	Max. #Nodes
1.2M	6-8	1-2	125.9	104.0	335.0

Experimental evaluation and baselines We report results using four standard ABC variants.

- Strash converts the circuit into an AIG representation using structural hashing. This serves as the basis for subsequent optimizations.
- Resyn is a lightweight synthesis script that alternates between balancing and rewriting.
- Compress2 applies balancing, rewriting, and refactoring, together with zero cost; neither method produces a structure that enables further simplifications.
- Resyn2rs is a heavyweight optimization script that performs multiple rounds of balancing, rewriting, refactoring, and resubstitution, progressively increasing the cut sizes ($K = 6$ to 12).

As you may notice, the methods are ordered by complexity, with Strash being the simplest and lightest, and Resyn2rs the most complex and expensive. Notably, we do not include any learning-based baselines for this dataset. While numerous graph-generative approaches exist (e.g., (Vignac et al., 2023)), most are designed for undirected graphs and thus unsuitable for DAGs. Moreover, our experiments with two DAG-specific generative methods, LayerDAG (Li et al., 2025a) and Directo (Carballo-Castro et al., 2025), indicate that neither method produces circuits functionally equivalent to the target truth table. This suggests a new and unexplored direction for the graph generation community, i.e., developing conditional DAG generative models that enforce strict constraints, such as functional equivalence with a given truth table.

Dataset challenges Aside from general graph-generation difficulties, this dataset presents specific challenges related to DAG generation and comparison with truth tables. As providing an exact match to the target truth table is very hard to achieve in general, even with sophisticated solvers as seen below, learning methods struggle to achieve good performance. Furthermore, small changes in generation can lead to substantially worse outcomes, requiring exact methods which most graph generation models are not. In addition to valid DAG generation, the target chip design score also takes circuit size into account, pushing models to learn smaller circuits.

Results Our results using four different ABC variants are reported in Table 9. We report the score defined in Equation (1).

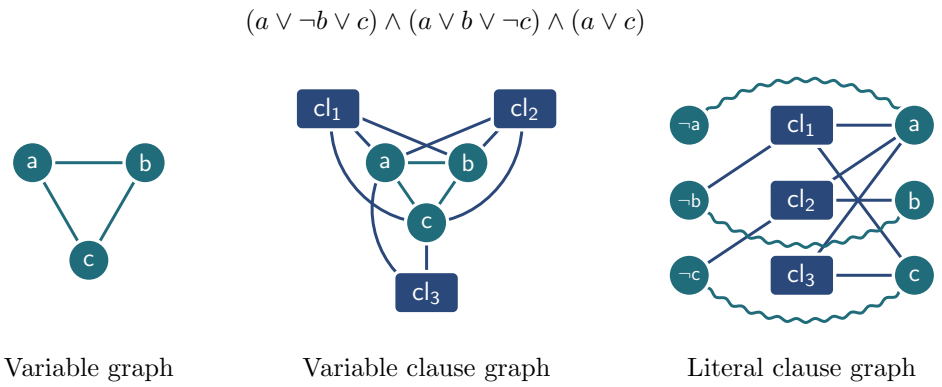

Figure 5. Overview of the three graph types generated from a given SAT formula.

The methods rank consistently with their complexity across all settings, with the simple STRASH yielding the worst results and the advanced RESYN2RS achieving the best performance.

Table 9. **Chip Design.** Results of the different ABC methods on the AIG dataset. Higher is better.

	Number of (inputs, outputs)						
Method	(6,1)	(6,2)	(7,1)	(7,2)	(8,1)	(8,2)	Overall
STRASH	73.90	74.23	76.55	75.71	77.63	76.52	75.76
RESYN	86.79	88.16	89.28	89.05	90.45	89.78	88.92
COMPRESS2	92.53	93.57	93.18	92.79	93.28	92.49	92.97
RESYN2RS	93.30	95.24	95.15	95.64	96.24	96.11	95.28

B.2.2. ELECTRONIC CIRCUITS: PREDICTING VOLTAGE CONVERSION RATIO

Related work Learning for circuit design has been explored for physical implementation (Lu et al., 2020) and parameter optimization (Wang et al., 2020), typically under the assumption of fixed topologies. Topology exploration, in contrast, has relied on heuristic search, including evolutionary and genetic algorithms (McConaghy et al., 2011), as well as tree-based strategies (Fan et al., 2021; Zhao & Zhang, 2020). Our benchmark differs in that it treats performance prediction as a supervised graph learning task, using standardized splits and metrics to enable systematic comparison across architectures.

B.3. Reasoning and Optimization

B.3.1. SAT SOLVING: ALGORITHM SELECTION AND PERFORMANCE PREDICTION

The *Boolean satisfiability problem* (SAT) is a central and longstanding NP-complete problem (Karp, 1972; Biere et al., 2021). It asks whether there exists a truth assignment that satisfies a Boolean formula φ over variables $v_1, \ldots, v_r$, typically expressed in *conjunctive normal form* (CNF):

$$\varphi = \bigwedge_{i=1}^{n} c_i, \quad \text{where each } c_i = \bigvee_{j=1}^{m_i} l_{i,j}.$$

Here, each $l_{i,j}$ is a literal, i.e., either a variable v_p or its negation $\neg v_p$. The task is to decide whether there exists an assignment a such that $\varphi(a) =$ true. SAT is of significant theoretical importance as one of the first problems proven NP-complete (Cook, 1971), and it underpins numerous practical applications in hardware and software verification (Biere et al., 2009), automated planning (Kautz & Selman, 1996), and operations research (Gomes et al., 2008). Its significance has led to the development of a wide range of solvers, systematically benchmarked in annual SAT competitions (e.g., (Froleyks et al., 2021; Balyo et al., 2017; Heule et al., 2019)). A key property of modern SAT solvers is *performance complementarity*—no single solver dominates across all instances. This motivates the study of *algorithm selection* (Rice, 1976; Xu et al., 2008), where the aim is

to choose the most effective solver for a given instance. *Empirical performance prediction* is a closely related problem, which seeks to forecast the computation time (or another performance metric) of a solver on a given instance. Such predictions can support algorithm selection, configuration, scheduling, and explainability (Hutter et al., 2014a). State-of-the-art methods for algorithm selection and performance prediction rely on *hand-crafted features* extracted from SAT instances. The most widely adopted feature set is that of the SATZILLA family of algorithm selectors (Nudelman et al., 2004; Hutter et al., 2014a; Shavit & Hoos, 2024). These features include basic structural properties (e.g., number of clauses and variables), graph-based descriptors derived from instance structure, and probing features obtained by briefly running a solver to collect computation-time statistics. Machine learning models—most often tabular models—are then trained on these features for prediction or selection tasks.

SAT instances can be naturally represented as graphs, reflecting their permutation-invariant structure that arises from the commutativity and associativity of logical conjunction and disjunction. Indeed, many of the hand-crafted SATZILLA features are derived from such graph representations, including the

- *Variable-clause graph*: a bipartite, undirected graph with a node for each variable v and each clause c, where an edge connects a variable to a clause if and only if the variable appears in that clause.
- *Variable graph*: an undirected graph with one node per variable, where two variables v_i and v_j are connected if they co-occur in at least one clause.
- *Clause graph*: an undirected graph with one node per clause, where two clauses c_i and c_j are connected if they share at least one negated literal.

SATZILLA extracts statistical properties of these graphs—such as mean degree, coefficient of variation, clustering coefficient, and graph diameter—and uses them as features for downstream learning tasks.

Several studies have explored the use of GNNs or MPNNs to predict the satisfiability of a formula, e.g., Selsam et al. (2019). Some of this work employs an additional representation, the *literal-clause graph*, a bipartite graph with one node per literal (a variable or its negation) and one node per clause, with edges linking literals to the clauses they appear in. However, many such approaches are *unsound*—they can produce incorrect results—or cannot guarantee valid satisfying assignments or proofs of unsatisfiability, limiting their practical usefulness (Selsam et al., 2019; Li et al., 2024c). Other research has integrated MPNNs into existing SAT solvers to replace or guide solver heuristics (Wang et al., 2024; Tönshoff & Grohe, 2025). These approaches retain the soundness and completeness guarantees of classical solvers while potentially accelerating the solving process. Nonetheless, benchmarking novel MPNN architectures in this setting is challenging because it requires extensive pretraining and large-scale evaluation. A complementary line of work applies MPNNs to algorithm selection in SAT (Zhang et al., 2024; Shavit, 2023), predicting the most effective solver for a given instance directly from graph-based representations.

We add node features to provide the GNN with additional information about the formula's structure. Our node embeddings are inspired by the hand-crafted features used in SATZILLA (Nudelman et al., 2004) and the node-level encodings proposed in GRASS (Zhang et al., 2024). These features capture local syntactic properties (e.g., polarity counts, clause membership statistics), normalized and relational quantities (e.g., positive-to-negative ratio, normalized frequency), and sinusoidal positional encodings for clause nodes. Tables 10 to 12 summarize the node features for each graph type. All features are computed during graph construction and stored as floating-point node attributes.

In GRAPHBENCH, we introduce, to the best of our knowledge, the largest algorithm-selection benchmark for SAT solvers, covering eleven solvers and more than 100 000 instances. Our benchmark supports multiple graph representations of SAT formulae and also provides the SATZILLA 2024 feature set (Shavit & Hoos, 2024), enabling their combination with MPNNs. Unlike existing GNN benchmarks for SAT (Li et al., 2024c), which primarily focus on satisfiability prediction or assignment generation, our benchmark is designed for practical downstream tasks in SAT solving—specifically, algorithm selection and performance prediction. Figure 5 provides an overview of how the three graph types are constructed from a given Boolean formula.

Related work As the use of MPNNs for algorithm selection is a relatively new research direction, no benchmarks exist for this task. Previously, Shavit (2023) used a dataset of 3 000 synthetically generated SAT instances, while Zhang et al. (2024) used a proprietary dataset as well as the 2022 Anniversary track of the SAT Competition, while removing large instances. Outside of the graph domain, the ASLIB benchmark suite is used to benchmark algorithm selection methods (Bischl et al., 2016). ASLIB contains multiple algorithm-selection datasets across various domains, including SAT and TSP. Each dataset contains features extracted from instances (such as SATZILLA) as well as the computation times of several algorithms on

Table 10. Node features for the variable-clause graph (VCG). Variable nodes carry 12 features, and clause nodes carry 22 features. Edges between variables and clauses are attributed with the literal polarity ($+1/-1$); reverse edges are included.

Node type	Dim.	Feature description
Variable (12 features)	1–2	Constant placeholders (reserved, set to zero)
	3	Number of Horn clauses in which this variable appears
	4	Number of occurrences as a positive literal
	5	Number of occurrences as a negative literal
	6	Total number of clause appearances (degree)
	7	Ratio of positive to negative occurrences
	8	Number of appearances in binary clauses (length two)
	9	Number of appearances in unit clauses (length one)
	10	Number of appearances in ternary clauses (length three)
	11	Normalised frequency (degree divided by the total number of clauses)
	12	Mean length of clauses containing this variable
Clause (22 features)	1–10	Sinusoidal positional encoding (10 dimensions)
	11	Number of positive literals
	12	Number of negative literals
	13	Ratio of positive to negative literals
	14	Clause length (number of literals)
	15	Indicator: unit clause (length one)
	16	Indicator: binary clause (length two)
	17	Indicator: ternary clause (length three)
	18	Indicator: Horn clause (at most one positive literal)
	19	Normalized clause length (divided by the maximum clause length)
	20	Normalized clause position (index divided by the total number of clauses)
	21	Indicator: definite Horn clause (exactly one positive literal)
	22	Indicator: negative clause (all literals are negative)

Table 11. Node features for the literal-clause graph (LCG). Literal nodes carry 12 features, and clause nodes carry 22 features. Literal-to-clause edges carry a polarity attribute ($+1/-1$); reverse and complement edges (linking each literal to its negation) are also included.

Node type	Dim.	Feature description
Literal (12 features)	1	Indicator: positive literal
	2	Indicator: negative literal
	3	Number of Horn clauses in which this literal appears
	4	Total number of clause occurrences
	5	Number of clause occurrences of the complementary literal
	6	Degree (equal to the occurrence count)
	7	Ratio of this literal's occurrences to those of its complement
	8	Number of appearances in binary clauses (length two)
	9	Number of appearances in unit clauses (length one)
	10	Number of appearances in ternary clauses (length three)
	11	Normalized frequency (occurrences divided by the total number of clauses)
	12	Mean length of clauses containing this literal
Clause (22 features)	1–10	Sinusoidal positional encoding (10 dimensions)
	11	Number of positive literals
	12	Number of negative literals
	13	Ratio of positive to negative literals
	14	Clause length (number of literals)
	15	Indicator: unit clause (length one)
	16	Indicator: binary clause (length two)
	17	Indicator: ternary clause (length three)
	18	Indicator: Horn clause (at most one positive literal)
	19	Normalized clause length (divided by the maximum clause length)
	20	Normalized clause position (index divided by the total number of clauses)
	21	Indicator: definite Horn clause (exactly one positive literal)
	22	Indicator: negative clause (all literals are negative)

Table 12. Node features for the variable graph (VG). Each variable node carries 12 features. Edges connect co-occurring variables and carry no attributes.

Node type	Dim.	Feature description
Variable (12 features)	1	Number of Horn clauses in which this variable appears
	2	Number of occurrences as a positive literal
	3	Number of occurrences as a negative literal
	4	Total co-occurrence count (degree in the variable graph)
	5	Ratio of positive to negative occurrences
	6	Number of appearances in binary clauses (length two)
	7	Number of appearances in unit clauses (length one)
	8	Number of appearances in ternary clauses (length three)
	9	Normalized frequency (clause appearances divided by the total number of clauses)
	10	Mean length of clauses containing this variable
	11	Total number of distinct clauses containing this variable
	12	Mean co-occurrence degree (co-occurrence count divided by the number of containing clauses)

those instances. We note that ASLIB provides SAT scenarios based on a subset of SAT competitions, including instances and solvers from specific years. In contrast, our dataset comprises all available SAT competition instances, as well as numerous instances from other sources.

Description of the learning task We introduce two learning tasks. *Performance prediction* is a regression problem that aims to predict the computation time of SAT solvers on unseen instances, arising in applications such as algorithm selection (Rice, 1976) and algorithm configuration (e.g. Lindauer et al., 2022). *Algorithm selection* is a multi-class classification problem that aims to select the best algorithm for a given instance. While algorithm selection can be reduced to performance prediction by choosing the solver with the lowest predicted computation time, we treat them as distinct problems. For algorithm selection, we adopt the loss function proposed by (Zhang et al., 2024), which penalizes the predicted probabilities in proportion to the solver's computation time. Intuitively, a solver that only slightly underperforms on an instance incurs a small penalty, while poor selections are penalized more heavily, i.e.,

$$\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{k=1}^{K} p_i^k \cdot t_i^k - t_i^*\right)^2,$$

where p_i^k is the predicted probability of selecting solver k for instance i, t_i^k is the computation time of solver k on instance i, and $t_i^* = \min_k t_i^k$ is the computation time of the *virtual best solver* (VBS). Each instance is provided with three graph-based representations VG, VCG, and LCG, described above. In all cases, SATZILLA features can be integrated as node attributes to enrich the graph representation.

The performance of a solver s on an instance set $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ is measured using the widely adopted *penalized average computation time* (PAR_k) metric, where k denotes the penalty factor for unsolved instances under a cutoff time c. The $\text{PAR}_k(s, \Pi)$ score is defined as the mean computation time of s across Π, where each unsolved instance contributes a computation time of $k \cdot c$. Lower PAR_k values indicate better performance. For algorithm selectors, both feature extraction and model inference time are included in the total solving time, implying that MPNN inference must be efficient to be practically helpful.

For the performance prediction task, the model output is the base-10 logarithm of the PAR_{10} score of solver s,

$$\log_{10}(\text{PAR}_{10}(s, \Pi)),$$

a transformation shown to improve predictive accuracy in prior work (Hutter et al., 2014a). Model performance is evaluated using the *root mean-squared error* (RMSE) between predicted and actual log-scaled computation times.

For algorithm selection, we use the *closed gap* (CG) metric, which quantifies how close a selector comes to the performance of the VBS relative to the single best solver (SBS). Formally,

$$\mathrm{CG}(\Pi) = \frac{\mathrm{PAR}_{10}(\mathrm{SBS}, \Pi) - \mathrm{PAR}_{10}(s, \Pi)}{\mathrm{PAR}_{10}(\mathrm{SBS}, \Pi) - \mathrm{PAR}_{10}(\mathrm{VBS}, \Pi)}.$$

A higher CG score indicates stronger selection performance, with $\mathrm{CG} = 1$ corresponding to VBS performance and $\mathrm{CG} = 0$ to SBS performance.

Details on the datasets We created the largest algorithm-selection scenario for SAT solvers to date. We collected all SAT Competition instances via GBD (Iser & Jabs, 2024) and added all instances from the AClib benchmark for algorithm configuration (Hutter et al., 2014b). Together, these sources include SAT formulae from diverse real-world applications and synthetically generated instances. To further enrich the dataset, we generated additional formulae using three well-established SAT instance generators, i.e.,

- *FuzzSAT* (Brummayer et al., 2010) produces CNF fuzzing instances originally designed to identify issues in digital circuits. It generates random Boolean circuits as DAGs and converts them into CNF via the Tseitin transformation (Tseitin, 1983). To construct the DAG, we used $v \in [50,150]$ input variables; operands were randomly paired and connected by randomly chosen operator nodes until each input was referenced at least $t \in [1,5]$ times. Finally, random clauses of length $s \in [3,8]$ were added to increase complexity.
- *Cryptography* (Nejati et al., 2017; Alamgir et al., 2024) generates SAT instances encoding preimage attacks on MD4, SHA-1, and SHA-256. Each encoding applies multiple *rounds* of operations; higher numbers of rounds yield harder instances. We set the number of rounds to $d \in [16,30]$, which yields formulae that are challenging for modern solvers but not unsolvable. Additionally, we fixed $i \in [0,384]$ of the 512 input bits; fixing more bits results in easier instances.
- *Community attachment* (Giráldez-Cru & Levy, 2016) produces pseudo-random SAT formulae with community structure, a property often observed in real-world SAT encodings (e.g., hardware and software verification tasks (Ansótegui et al., 2012)). The generator can create relatively small but difficult instances. We used $n \in [1\,000, 3\,000]$ variables, $c \in [5,100]$ communities, a clause-to-variable ratio of $[3.7, 4.5]$, and modularity factor $q \in [0.3,0.9]$. Higher modularity values enforce stronger community structure.

For each generator, parameter ranges were chosen to yield challenging but still solvable instances. To ensure this, we excluded ranges of values yielding no instances solvable by KISSAT, the winner of the 2024 SAT Competition, between 1 and 5 000 seconds. After generation, we applied the SATELLITE preprocessor to remove redundant clauses and variables. A summary of all generated instances is provided in Table 13.

We pre-selected solvers for the benchmark by constructing a portfolio of m complementary state-of-the-art SAT solvers, based on the results of the 2024 SAT Competition. Portfolios were built using beam search to identify the set of m solvers achieving the VBS performance. To determine m, we evaluated portfolios of all possible sizes and plotted portfolio size against achieved VBS. We then identified the inflection point (the "knee") of this curve using the Kneedle algorithm (Satopaa et al., 2011) and adopted the corresponding portfolio. The ASF library (**?**) was used to perform the selection procedure. The same procedure was repeated on the SAT Competition 2023.

In total, our dataset includes the following solvers. From the 2024 SAT Competition: Kissat (Biere et al., 2024), BreadIdKissat (Bogaerts et al., 2024), AMSAT (Li et al., 2024b), and KissatMABDC (Liu et al., 2024). From the 2023 SAT Competition: SBVA CadiCal (Haberlandt & Green, 2023), Reencode Kissat (Reeves & Bryant, 2023), Minisat XOR, KissatMAB Prop PR (Gao, 2023), hKissatInc (Tchinda & Djamegni, 2023), CadiCal vivinst (Biere et al., 2023), and BreakID Kissat (Bogaerts et al., 2023). Solver computation times were measured in CPU time using `runsolver` (Roussel, 2011) and the generic wrapper tools (Eggensperger et al., 2019), ensuring accurate and consistent performance measurements.

We provide three datasets, SMALL, which includes only small formulae with up to 3 000 variables and 15 000 clauses, MEDIUM, which consists of the small formulae with additional medium size formulae with size of up to 20 000 variables and 80 000 clauses and LARGE, which includes all formulae. This way, we provide datasets with varying levels of hardness, as formula size can pose hurdles for training time and GPU memory. While the LARGE and MEDIUM datasets are unlikely to be accessible on current hardware, we provide them to keep the benchmark relevant in the long term.

Table 13. Overview of the instances for the SAT solving benchmark.

Source	**# of Instances**	**# of Variables**				**# of Clauses**			
		min	max	avg	median	min	max	avg	median
SAT Competition	31 024	3	25 870 369	69 190.77	13 168.50	7	871 935 536	1 022 025.50	118 526.00
Community Attachment	29 994	922	2 956	1 931.05	1 928.00	3 566	13 413	8 088.71	8 041.00
AClib	33 955	6	248 738	3 319.65	1 270.00	24	103 670 100	136 471.03	10 000.00
FuzzSAT	8 020	2	944 685	10 217.75	1 820.00	1	4 094 516	45 638.30	8 074.50
Cryptography	4 873	6	12 415	4 264.89	3 225.00	24	185 006	55 309.63	45 316.00
Total	107 866	2	25 870 369	22 434.71	1 879.00	1	871 935 536	345 051.72	10 177.50

Dataset challenges: Algorithm selection and performance prediction are particularly challenging in this setting for several reasons. First, solver performance is inherently stochastic: repeated runs of the same solver on the same instance can result in substantially different running times. This induces noisy target values that the learning algorithm must account for. Second, the targets are right-censored due to the use of solver cutoffs: when a solver does not terminate before the cutoff, its true running time is only known to exceed the cutoff. Third, the SAT instances can be very large, with corresponding graphs containing thousands to millions of variable nodes, making scalable graph processing a major challenge. Finally, accurate prediction requires the GNN to capture both local structural patterns and the formula's global properties.

Experimental evaluation and baselines All SAT solving tasks in the benchmark are graph-level. For the graph transformer baselines, we employ an encoder-processor-decoder pipeline, as described in Appendix D. We used the following architectures: GIN, GCN, GIN+, GCN+, Gated GCN+. While we considered GT for the tasks, we encountered high computational costs in calculating the positional embeddings (more than a CPU day) and therefore excluded them from the evaluation. To ensure a fair comparison with hand-crafted features and to evaluate the capacity of GNNs to extract relevant information purely from the formula's topology, we did not provide the models with any initial node-level features, forcing them to learn solely from the graph structure. Furthermore, when training on medium- and large-sized graphs, we encountered out-of-memory errors even on an H100. Consequently, we only consider small formulae for MPNNs. Similarly, we observed out-of-memory issues with the clause graphs and therefore excluded them from the evaluation. We optimized the hyperparameters of these models using SMAC3 (Lindauer et al., 2022) with a multi-fidelity facade, running 60 trials. Full details of the hyperparameter configuration space and the tuning procedure can be found in Appendix E.

We note that traditionally, algorithm selection datasets are evaluated using cross-validation, primarily due to limited data availability. In contrast, our new dataset contains tens of thousands of formulas. Since it is also intended for deep learning approaches, which are computationally expensive and thus impractical to evaluate using 10-fold cross-validation, we instead provide a fixed train–validation–test split and report results across 3 random seeds.

For the performance prediction task, we compare the MPNNs against traditional baselines, which use the SATZILLA features and a (tabular) machine learning model. We use random forest (Breiman, 2001) and XGBoost (Chen & Guestrin, 2016), which are commonly used as empirical performance models (EPM) (see, e.g., Bansal et al. (2022); Eggensperger et al. (2015); Hutter et al. (2014c)). We use the SATZILLA 2024 features as input and predict the log10-scaled computation time. These baselines are implemented using the ASF framework (**?**). For algorithm selection, we provide several well-known baselines. First, we use pairwise regression (Kerschke et al., 2018) (PR; predicts performance differences between each pair of solvers, then picks the solver with the best sum), pairwise classification (Xu et al., 2012) (PC; predicts the better solver in each pair, then uses majority vote, winner of the ICON challenge on algorithm selection (Lindauer et al., 2019)) as well as a simple multi-class classification (Kotthoff, 2013) (MC; directly predicts the best solver). Similar to the EPM baselines, these baselines use the SATZILLA 2024 features with a random forest. We tune these baselines using SMAC3 (Lindauer et al., 2022) for 50 trials.

Results We start by presenting the results for the performance prediction task. We present the RMSE of the log-scaled computation time for the solvers from SAT Competition 2024 (KISSAT, BREAKIDKISSAT, KISSATMABDC, AMSAT) for the small formulae in table Table 14. Note that missing values (–) for Gated GCN Plus with the VG representation indicate that the model encountered out-of-memory (OOM) errors on the GPU. The table shows that the best performance for KISSAT, BREAKIDKISSAT, and AMSAT is achieved by GCN Plus with the VCG graph, as this model outperforms the SATzilla features. For KISSATMABDC, a simple GIN with a variable graph performs best. We also note that the SAT-specific architecture, GraSS, performs surprisingly poorly across the board compared to general-purpose architectures. To the best of our knowledge, this is the first time the SATzilla features have been outperformed on the performance

Table 14. **SAT Solving.** RMSE ($\log_{10}$ time) for performance prediction on SMALL SAT instances. We report results on three different graph representations (VG, VCG, LCG) and using the hand-crafted SATzilla features. Lower is better.

Solver	Method	VG	VCG	LCG	SATZILLA
KISSAT	GIN	0.66± 0.02	0.64± 0.07	0.70± 0.01	
	GCN	0.98± 0.00	0.70± 0.01	0.82± 0.07	
	GCN+	0.69± 0.03	0.51± 0.02	0.67± 0.09	
	GIN+	0.78± 0.02	0.87± 0.05	0.83± 0.07	
	Gated GCN+	–	1.05± 0.04	1.04± 0.04	
	RF				0.65± 0.00
	XGB				0.64± 0.00
BREAKIDKISSAT	GIN	0.65± 0.01	0.63± 0.04	0.68± 0.01	
	GCN	0.96± 0.00	0.70± 0.01	0.80± 0.09	
	GCN+	0.69± 0.01	0.58± 0.03	0.62± 0.03	
	GIN+	0.82± 0.10	0.89± 0.03	0.95± 0.08	
	Gated GCN+	–	0.94± 0.02	0.94± 0.03	
	RF				0.68± 0.00
	XGB				0.67± 0.00
KISSATMABDC	GIN	0.64± 0.03	0.72± 0.01	0.70± 0.02	
	GCN	0.99± 0.00	0.75± 0.02	0.74± 0.01	
	GCN+	0.74± 0.07	0.86± 0.01	0.78± 0.07	
	GIN+	0.83± 0.09	1.05± 0.06	0.94± 0.01	
	Gated GCN+	–	0.91± 0.15	1.05± 0.18	
	RF				0.70± 0.00
	XGB				0.69± 0.00
AMSAT	GIN	0.71± 0.02	0.70± 0.01	0.71± 0.01	
	GCN	1.01± 0.00	0.82± 0.10	0.75± 0.01	
	GCN+	0.85± 0.09	0.61± 0.02	0.74± 0.06	
	GIN+	0.92± 0.07	1.05± 0.04	1.04± 0.06	
	Gated GCN+	–	1.02± 0.01	1.03± 0.07	
	RF				0.74± 0.00
	XGB				0.74± 0.00

Table 15. **SAT Solving.** RMSE ($\log_{10}$ time) for performance prediction on MEDIUM and LARGE SAT instances. Lower is better.

(a) Medium-size SAT instances

Solver	Method	SATZILLA
KISSAT	RF	0.63± 0.00
	XGB	0.62± 0.00
BREAKIDKISSAT	RF	0.66± 0.00
	XGB	0.65± 0.00
KISSATMABDC	RF	0.68± 0.00
	XGB	0.67± 0.00
AMSAT	RF	0.70± 0.00
	XGB	0.69± 0.00

(b) Large SAT instances

Solver	Method	SATZILLA
KISSAT	RF	0.64± 0.00
	XGB	0.64± 0.00
BREAKIDKISSAT	RF	0.66± 0.00
	XGB	0.65± 0.00
KISSATMABDC	RF	0.67± 0.00
	XGB	0.66± 0.00
AMSAT	RF	0.69± 0.00
	XGB	0.69± 0.00

prediction task. We furthermore present the results of random forest and XGBoost on medium-size and large formulae in Tables 15a and 15b, such that they can be compared against graph machine learning solutions in the future. Overall, we observe similar performance between XGBoost and random forest, with a slight advantage to XGBoost.

For the algorithm selection task, we report the closed gap metric of different selectors in Table 16. For the small instances, we observe that gated GCN plus performs best, followed by GCN plus and GIN, all of which are trained on literal-clause graphs. These graph-based approaches outperform the SATzilla feature-based baselines, with only pairwise regression achieving a positive closed gap. Comparing the different graph representations across tasks, we observe clear trade-offs: while the VCG representation yields the strongest performance for the prediction task, the LCG representation enables the highest gap closure for algorithm selection. This suggests that the explicit literal polarity and complementary edges in LCG might be more beneficial for relative performance ranking than for absolute runtime prediction. Among the three baselines based on hand-crafted features, we observe that multi-class classification underperforms both pairwise classification and regression, leaving a negative gap on the small and medium-sized datasets. This highlights that algorithm selection is not a simple multi-class classification problem. In that formulation, the model loses valuable information about the performance of algorithms other than the best one. Finally, we find more gaps closed for the large dataset across all three baselines, as it includes harder instances. As time increases, the benefit of algorithm selection grows because feature computation accounts for a smaller fraction of the total computation time. Moreover, for harder instances, multiple solvers frequently time out, so choosing a solver that successfully solves the instance becomes especially valuable, given the additional penalty imposed by the PAR score.

Table 16. **SAT Solving.** Algorithm selection results (gap closed; higher is better).

Instances	Method	VG	VCG	LCG	SATZILLA
SMALL	GIN	0.45± 0.00	0.45± 0.00	0.45± 0.01	
	GCN	0.00± 0.00	0.40± 0.00	0.38± 0.03	
	GCN+	0.31± 0.00	0.41± 0.01	0.47± 0.02	
	GIN+	0.28± 0.05	0.02± 0.01	0.16± 0.08	
	Gated GCN+	–	0.40± 0.04	0.48± 0.03	
	PR				0.38± 0.02
	MC				-0.44± 0.01
	PC				-0.26± 0.00
MEDIUM	PR				0.35± 0.01
	MC				-0.43± 0.04
	PC				0.01± 0.01
LARGE	PR				0.51± 0.01
	MC				-0.05± 0.01
	PC				0.21± 0.01

B.3.2. COMBINATORIAL OPTIMIZATION: LEARNING TO PREDICT OPTIMAL SOLUTION OBJECTIVES

Related work Many recent learning-based optimization work rely on synthetic data (Toenshoff et al., 2021; Tönshoff et al., 2023; Zhang et al., 2023; Karalias & Loukas, 2020), either RB graphs (Xu et al., 2007), Erdős-Rényi (ER) graphs (Erdős & Rényi, 1960), or Barabási-Albert (BA) graphs (Albert & Barabási, 2002). Some also utilize pre-existing datasets from earlier graph benchmarks, e.g., (Karalias & Loukas, 2020) uses TUDatasets (Morris et al., 2020), and Li et al. (2022) uses Planetoid datasets (Yang et al., 2016). Earlier efforts have also produced benchmarks for specific graph CO problems. For graph coloring, examples include hard 3-colorability datasets featuring cliques (Mizuno & Nishihara, 2007; 2008), queen graphs (Chvátal, 2004), Latin square graphs (Gomes & Shmoys, 2002), K-Insertions graphs (Caramia & Dell'Olmo, 2002a), and K-FullIns graphs (Caramia & Dell'Olmo, 2002b). The DIMACS benchmark collection (Rossi & Ahmed, 2015) provides hundreds of real-world networks and problem instances across various CO domains. Still, their heterogeneity and the limited number of instances per category make them less well-suited to modern machine learning pipelines. More recently, Nath & Kuhnle (2024) introduced a max-cut benchmark covering a variety of graph classes. Cantürk et al. (2026) also introduces datasets for CO, but frames it as an inverse problem. Our motivation differs from these prior works. We aim to design a unified set of graph datasets shared across multiple CO tasks, enabling direct comparison between methods in a consistent experimental setting. While some prior work generates random instances for individual problems, we standardize the process by fixing both the generation procedures and the training-validation splits, ensuring that all methods compare on

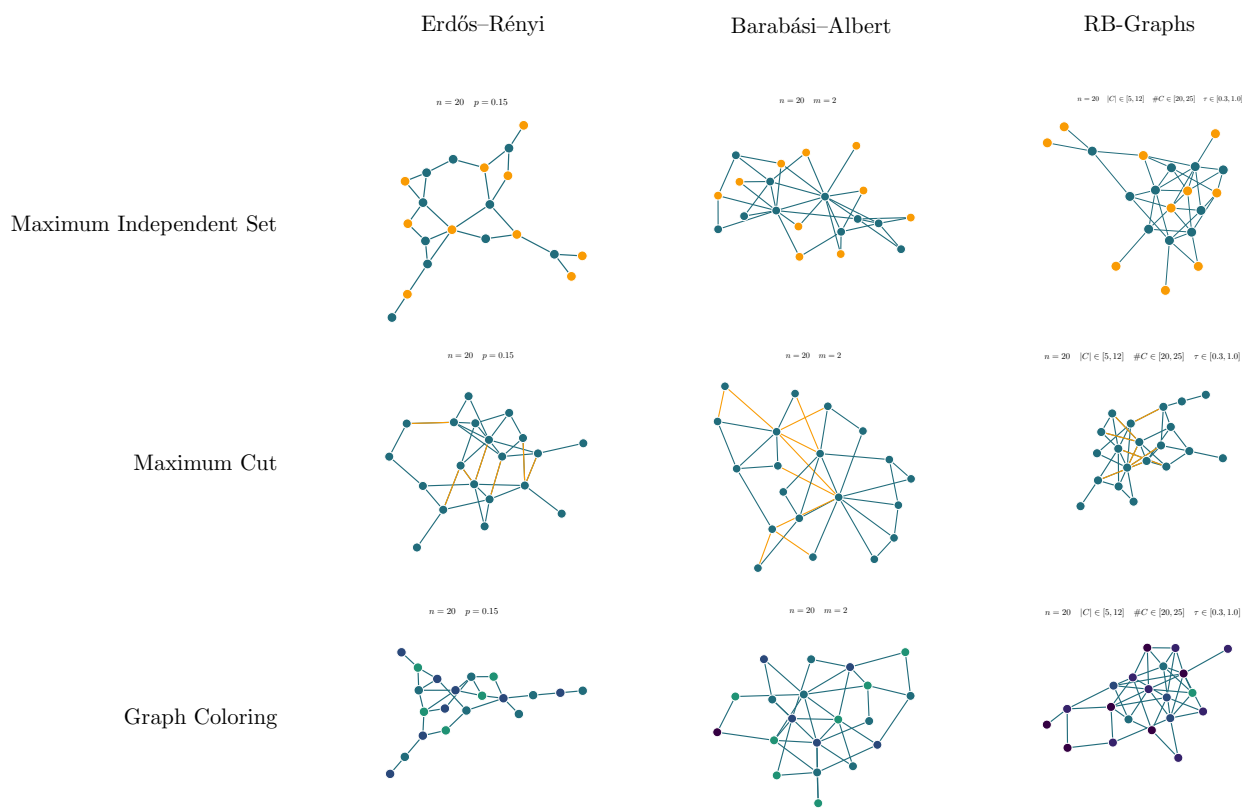

Figure 6. Down-scaled sample graphs and solutions across all problems and graph generators for the *Combinatorial Optimization* datasets with our chosen parameters.

the same instances.

Definitions of CO problems Given a graph $G = (V(G), E(G))$, the MIS problem is to find the largest subset of non-adjacent nodes, formally,

- $\Omega(I) \coloneqq V(G)$
- $F(I) \coloneqq \{S \subseteq V(G) \mid \forall\, u, v \in S,\ (u, v) \notin E(G)\}$
- $w\colon V(G) \to \mathbb{R}$, in the unweighted case $w(v) \coloneqq -1$.

The objective is to find $S_I^* \in F(I)$ minimizing $c(S) \coloneqq \sum_{v \in S} w(v)$.

The max-cut problem seeks a 2-way partition of the node set such that the total weight of edges across the cut is maximized, i.e.,

- $\Omega(I) \coloneqq E(G)$,
- $F(I) \coloneqq \{S \subseteq E(G) \mid \exists\, V'(G) \subset V(G), \forall (u, v) \in S\colon u \in V(G), v \in V(G) \setminus V'(G)\}$,
- $w\colon E(G) \to \mathbb{R}$, in the unweighted case $w(e) \coloneqq -1$.

The objective is to find $S_I^* \in F(I)$ minimizing $c(S) \coloneqq \sum_{e \in S} w(e)$.

Graph coloring is the problem of assigning at most k colors to the nodes such that adjacent nodes receive different colors. Inspired by the assignment-based model (Jabrayilov & Mutzel, 2018), it can be formulated in search of a coloring function $f\colon V(G) \to [k]$, such that

- $\Omega(I) \coloneqq [k]$,
- $F(I) \coloneqq \{S \subseteq [k] \mid \forall\, (u, v) \in E(G)\colon f(v) \neq f(u) \wedge \forall\, v \in V(G)\colon f(v) \in S\}$,

Table 17. Parameters for CO problems generation.

Generator	Size	Nodes	Nodes/Clique	Cliques	Tightness	Edge Prob.	Attached Edges
RB	small	[200, 300]	[5, 12]	[20, 25]	[0.3, 1]	-	-
	large	[800, 1200]	[20, 25]	[40, 55]	[0.3, 1]	-	-
ER	small	[200, 300]	-	-	-	0.15	-
	large	[700, 800]	-	-	-	0.15	-
BA	small	[200, 300]	-	-	-	-	2
	large	[700, 800]	-	-	-	-	2

- $w\colon \Omega(I) \to \{1\}$.

The objective is to find the best $S_I^* \in F(I)$, such that the minimum number of colors is used, $c(S) \coloneqq \sum_{s \in S} w(s)$. Figure 6 shows examples of all three CO problems across the three graph generators we use.

Graph generation parameters We consider three well-established random graph models: RB, ER, and BA graphs. These models are widely used in machine learning to solve CO problems. The diversity and difficulty of each dataset are controlled via graph generation parameters. Specifically,

- for RB graphs, we specify integers n (number of cliques), k (nodes per clique), and a float p (constraint tightness). The total number of nodes is kn, and we discard graphs whose size falls outside a predefined range.
- For BA graphs, we control the number of nodes and the parameter m, which determines how many edges are attached from each new node.
- For ER graphs, we use the number of nodes and the edge probability p, which controls the probability that an edge exists between any given pair of nodes.

We generate 50 000 instances per dataset and provide both small- and large-scale variants. Table 17 shows the parameters we used to generate the graphs in our datasets. For RB graphs, parameters are taken from (Zhang et al., 2023; Sanokowski et al., 2024; Wenkel et al., 2024; Sun et al., 2022). Parameters for large ER graphs are from (Böther et al., 2022; Qiu et al., 2022; Sun & Yang, 2023; Yu et al., 2024). Parameters for large BA graphs are from (Böther et al., 2022). The parameters for ER-small and BA-small are adjusted to match RB-small's graph size. When a parameter is specified as a range, it is sampled uniformly at random for each instance. The only exception to this is the number of nodes in RB graphs, where rejection sampling is used to ensure that the graph size falls inside a specified range, as described in Section 3.3.1. We provide datasets in both PyTorch Geometric format (Fey & Lenssen, 2019; Fey et al., 2025) and NetworkX format (Hagberg & Conway, 2020).

Heuristic solutions for supervised learning We use KaMIS (Lamm et al., 2016) to calculate heuristic MIS solutions. The solution to max-cut is obtained by formulating it as an integer programming problem and solving it with (Gurobi Optimization, LLC, 2024), with a timeout of 3600 seconds. We use GCol (Lewis & Palmer, 2025) for a heuristic graph chromatic number, with a maximum of 10^8 search iterations.

Model architecture Our proposed model uses the encoder-decoder architecture. The encoder, an MLP, processes node features derived from RWSE (Dwivedi et al., 2022a) and node degrees. The processor module is the only part that differs between baselines and performs the bulk of the calculations. It is followed by a decoder that generates node representations. For the supervised setting, we apply summation aggregation to obtain a scalar graph-level prediction as the final output. For the unsupervised setting, we leave the node-level scores as the output.

Extended results Table 18 shows the baseline results for unsupervised CO across all three CO problems. We report the average objective value of solutions obtained by running the decoder on the model's output scores for test-set problem instances. Out of the four baselines, GIN performs best across all three CO problems, with some exceptions. As expected, the graph-based models GIN and GT generally perform better than MLP and DeepSets. One exception is graph coloring on the (very sparse) BA graphs, where DeepSets outperforms the other two models across all graph sizes. Note that our baselines do not use diffusion models or other advanced architectures or techniques explicitly tailored for CO. Their performance is therefore much weaker than that of methods that do use them, e.g. Sanokowski et al. (2024); Zhang et al. (2023). Also note that most learning-based methods currently cannot compete with exact solvers and some hand-crafted heuristics in terms of

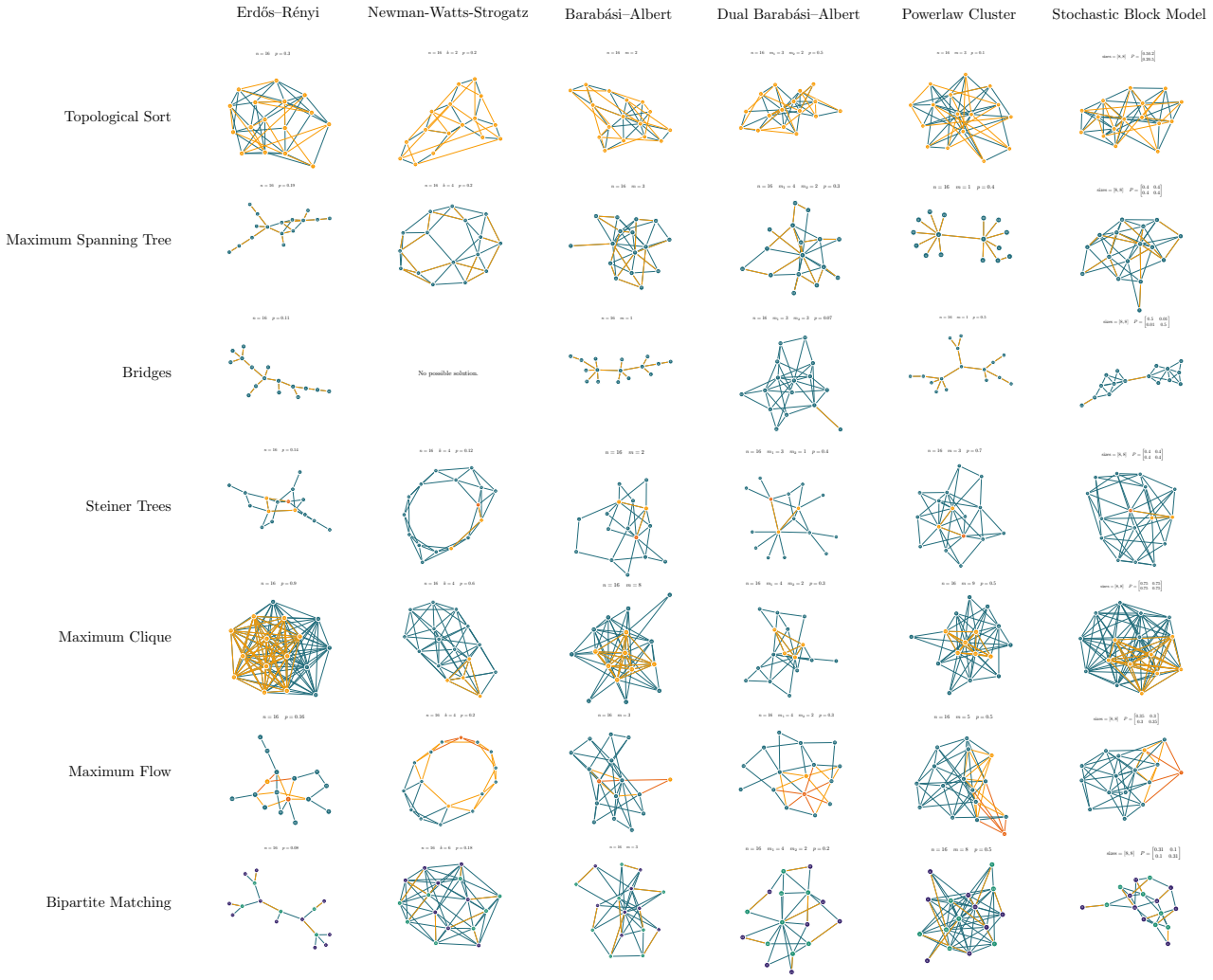

Figure 7. Example graphs and solutions across all graph generators and all performed tasks for the *Algorithmic Reasoning* datasets with our chosen parameters.

solution quality, but are often much quicker (Sun & Yang, 2023; Sanokowski et al., 2024; Zhang et al., 2023).

B.3.3. Algorithmic reasoning: Learning to simulate algorithms

In addition to SAT, many real-world tasks depend on the efficient solution of graph algorithms. Given the active exploration of the intersection of algorithms and neural networks across domains (Estermann et al., 2024; Fan et al., 2024a; Kaiser & Sutskever, 2016; Zaremba & Sutskever, 2014), our goal is to provide dedicated datasets for graph algorithmic reasoning. Prior work (Velickovic et al., 2020b; Xu et al., 2020; Bounsi et al., 2024) and benchmarks such as CLRS (Velickovic et al., 2022) and SALSA-CLRS (Minder et al., 2023) have demonstrated that GNN architectures can achieve strong performance on graph problems when provided with hints and graph invariants—tasks that require substantial theoretical expressiveness from graph learning models (Arvind et al., 2020). To broaden this research area, we contribute large-scale datasets for several common graph algorithm tasks, expanding the landscape of neural algorithmic reasoning benchmarks. Figure 7 provides an overview of our graph generators, the tasks we perform, and the parameters we chose for generation.

Related work Algorithmic reasoning has been adopted for various architectures (Li et al., 2024a; Velickovic et al., 2020a). Apart from neural networks (Diao & Loynd, 2023; Mahdavi et al., 2023; Bounsi et al., 2024; Rodionov & Prokhorenkova, 2023) and reinforcement learning approaches (Estermann et al., 2024), (Velickovic et al., 2022) propose the CLRS benchmark for a variety of algorithmic tasks, derived from the *Introduction to Algorithms* textbook (Cormen

Table 18. **Combinatorial Optimization.** Results for each CO dataset with unsupervised learning.

			Size	
Problem	**Dataset**	**Method**	Small	Large
MIS (MIS size ↑)	RB	GIN	17.294± 0.328	13.999± 0.321
		GT	16.542± 0.477	13.406± 0.140
		MLP	16.105± 0.097	13.040± 0.214
		DeepSets	16.021± 0.032	13.183± 0.035
		Solver	20.803± 1.817	42.547± 4.449
	ER	GIN	25.418± 0.407	26.276± 0.408
		GT	22.984± 0.473	24.980± 0.292
		MLP	23.183± 0.016	24.259± 0.449
		DeepSets	23.050± 0.061	24.220± 0.056
		Solver	33.604± 1.428	45.637± 0.631
	BA	GIN	100.16± 3.674	135.00± 0.720
		GT	99.579± 6.448	114.26± 0.601
		MLP	95.108± 2.042	114.49± 0.758
		DeepSets	95.076± 0.173	114.89± 0.016
		Solver	142.86± 16.54	433.77± 19.17
Maximum Cut (maximum cut size ↑)	RB	GIN	2106.7± 14.62	24748.± 87.76
		GT	1925.7± 32.75	21524.± 184.0
		MLP	1727.7± 165.1	20357.± 249.6
		DeepSets	140.02± 155.5	3575.9± 730.0
		Solver	2920.1± 97.23	33914.± 7861.
	ER	GIN	2327.9± 24.78	20878.± 107.9
		GT	2172.7± 91.75	16534.± 278.0
		MLP	1866.7± 67.64	7335.4± 57.49
		DeepSets	33.634± 20.84	27.663± 6.763
		Solver	2835.5± 607.6	23884.± 1809.
	BA	GIN	397.00± 0.605	1044.1± 0.649
		GT	363.76± 0.639	986.93± 3.128
		MLP	308.73± 0.224	929.20± 4.060
		DeepSets	1.0669± 0.800	154.31± 151.5
		Solver	460.91± 50.13	1260.4± 48.81
Graph Coloring (number of colors used ↓)	RB	GIN	25.166± 0.288	55.513± 0.526
		GT	25.146± 0.253	55.562± 0.648
		MLP	24.733± 0.667	55.558± 0.557
		DeepSets	26.723± 0.189	71.051± 0.604
		Solver	19.970± 3.465	41.480± 6.634
	ER	GIN	16.182± 0.202	34.587± 0.545
		GT	16.188± 0.201	34.385± 0.413
		MLP	17.110± 0.144	34.658± 0.394
		DeepSets	18.266± 0.018	55.345± 0.551
		Solver	10.235± 0.836	22.933± 0.772
	BA	GIN	5.1318± 0.114	6.2028± 0.283
		GT	5.0939± 0.070	6.1167± 0.086
		MLP	5.9900± 0.127	9.4215± 0.186
		DeepSets	3.2780± 0.122	3.1981± 0.239
		Solver	3.0000± 0.000	3.0000± 0.000

et al., 2009). This benchmark comprises 30 algorithmic problems, each accompanied by hints to guide the solution. However, this benchmark excludes regression tasks and limits learning tasks to specific algorithms. Furthermore, follow-up works investigated the importance of hints given in CLRS (Rodionov & Prokhorenkova, 2023) and proposed a variety of architectures for solving algorithmic tasks (Bevilacqua et al., 2023; Ibarz et al., 2022; Li et al., 2024a).

Description of learning task We consider a collection of seven algorithmic reasoning tasks, inspired by the task collection of Stoll et al. (2025) and selected from classical graph problems studied in algorithmics. These tasks include computing the topological sorting (Kahn, 1962), identifying bridges in a graph, computing the minimum spanning tree (Kruskal, 1956), determining the maximum flow (Goldberg & Tarjan, 1986), finding the maximum clique, computing Steiner trees (Kou et al., 1981), and solving bipartite matching (Hopcroft & Karp, 1973). Our task set spans different output granularities: node-level tasks (topological sorting, bipartite matching, max clique), edge-level tasks (bridges, minimum spanning tree, Steiner tree), and a graph-level task (max flow). To introduce additional modeling challenges, we provide three levels of difficulty per task, representing variations in the underlying graph distributions. Each task is formulated as either a binary classification or a regression problem. Given a set of graphs $\mathcal{G}_A$, an algorithmic reasoning task A, its ground-truth solution function S_A, and a model m, the goal is to train

$$m\colon \mathcal{G}_A \to \mathbb{R} \quad \text{or} \quad m\colon \mathcal{G}_A \to [0,1],$$

depending on the task. The objective is to minimize the MAE between the model's predictions and the ground-truth solution,

$$\frac{1}{|\mathcal{G}_A|} \sum_{G \in \mathcal{G}_A} |m(G) - S_A(G)| \,.$$

For binary classification tasks, we instead report the F1 score computed from the objective to maximize the accuracy between predictions and ground truth,

$$\frac{1}{|\mathcal{G}_A|} \sum_{G \in \mathcal{G}_A} \mathbb{1}(m(G) = S_A(G))$$

where $\mathbb{1}$ denotes the indicator function.

Details on the datasets For all algorithmic reasoning tasks, we work with sets of synthetically generated graphs. We begin by sampling graphs using a variety of graph generators, including Erdős–Rényi (ER) graphs (Erdős & Rényi, 1960), stochastic block model (SBM) graphs (Holland et al., 1983), power-law cluster (PC) graphs (Holme & Kim, 2002), Newman–Watts–Strogatz (NWS) graphs (Newman & Watts, 1999), Barabási–Albert (BA) graphs (Barabási & Albert, 1999), and dual Barabási–Albert (DBA) graphs (Moshiri, 2018). During sampling, we ensure that each resulting graph is unique; otherwise, it is resampled. For each task, we choose generator parameters shown in Table 19 to yield meaningful distributions of task-specific properties. We further introduce a parameter for component connections, which determines the number of random edges between disconnected components in a graph. This allows us to sample graphs with low edge probabilities without disconnected components. Additionally, based on the difficulty level, we adjust the probability distribution over the selected graph generators to further control the complexity of generated instances, as seen in Table 20. For minimum spanning tree computations, we also adjust the graph generator parameters, as indicated in Table 19, to require stronger generalization capabilities, as the same parameters proved too easy to generalize from. Ground-truth labels are computed using the NETWORKX library (Hagberg & Conway, 2020), which provides reference implementations of most of the required graph algorithms. For edge-level tasks, we also generate an edge-level representation to support models that require tokenized input. Each task includes one million generated training graphs and 10 000 validation and test graphs. To assess generalization beyond function approximation, we fix the number of nodes in the training graphs to 16 and set the validation and test graphs to have 128 nodes.

To further study generalization, we introduce size-generalization tasks, in which pre-trained models are evaluated on graphs with sizes ranging from 128 to 512 nodes. We exclude max-flow computation from size generalization, as the MAE scales with the number of nodes in the flow network and is therefore not indicative of size generalization. This setup enables an empirical analysis of how well models extrapolate to unseen graph sizes. Except for the topological sorting, Steiner tree, and bipartite matching tasks, no node features are given. For the minimum spanning tree, max flow, and Steiner tree tasks, corresponding edge weights are included as edge attributes.

Dataset challenges Across all algorithmic reasoning datasets, we distinguish training and test sets along two axes of data distribution: graph size and graph-generation sampling distributions. Drawing inspiration from CLRS (Velickovic et al.,

Table 19. Graph generation parameters for algorithmic reasoning tasks. The cc parameter denotes the number of connections between disconnected components in Erdős–Rényi graphs.

Generator	**Param.**	Top. Sort	MST	MST (shift)	Bridges	Steiner Trees	Max. Clique	Flow	Max. Matching
ER	p	0.3	0.19	0.17	0.11	0.14	0.9	0.16	0.08
	cc	1	1	1	1	1	1	1	1
NWS	k	2	4	2	-	4	4	4	6
	p	0.2	0.2	0.15	-	0.12	0.6	0.2	0.18
BA	m	2	3	2	1	2	8	3	3
DBA	m_1	3	4	2	3	3	4	4	4
	m_2	2	2	1	1	1	2	2	2
	p	0.5	0.3	0.05	0.07	0.4	0.3	0.3	0.2
PC	p	0.1	0.4	0.35	0.5	0.7	0.5	0.5	0.5
	m	3	1	5	1	3	9	5	8
SBM	p. mat.	$\begin{bmatrix}0.5 & 0.2\\0.2 & 0.5\end{bmatrix}$	$\begin{bmatrix}0.4 & 0.4\\0.4 & 0.4\end{bmatrix}$	$\begin{bmatrix}0.31 & 0.01\\0.01 & 0.31\end{bmatrix}$	$\begin{bmatrix}0.5 & 0.01\\0.01 & 0.5\end{bmatrix}$	$\begin{bmatrix}0.4 & 0.4\\0.4 & 0.4\end{bmatrix}$	$\begin{bmatrix}0.75 & 0.75\\0.75 & 0.75\end{bmatrix}$	$\begin{bmatrix}0.35 & 0.3\\0.3 & 0.35\end{bmatrix}$	$\begin{bmatrix}0.31 & 0.1\\0.1 & 0.31\end{bmatrix}$
	sizes	1/2, 1/2	1/2, 1/2	1/2, 1/2	1/2, 1/2	1/2, 1/2	1/2, 1/2	1/2, 1/2	1/2, 1/2

Table 20. Graph generation parameters for algorithmic reasoning tasks.

Split	**Difficulty**	ER	PC	NWS	BA	DBA	SBM
Train	EASY	1/3	0	1/3	0	1/3	0
	MEDIUM	1	0	0	0	0	0
	HARD	1	0	0	0	0	0
Valid/Test	EASY	1/6	1/6	1/6	1/6	1/6	1/6
	MEDIUM	1/6	1/6	1/6	1/6	1/6	1/6
	HARD	0	1/5	1/5	1/5	1/5	1/5

2022) and SALSA-CLRS (Minder et al., 2023), as well as the concept of scaffolding splits from molecular chemistry, we provide these two axes splits in order to test for OOD concepts. Since prior work (Minder et al., 2023) indicates potential size generalization capabilities in graph neural networks trained to simulate graph algorithms, we specifically test for these capabilities by increasing the graph size by a factor of 8 between the training and test sets. Furthermore, distributions are significantly shifted between training and test sets, as evidenced by three difficulty levels ranked by the magnitude of changes in the generation distribution.

Experimental evaluation and baselines We use the encoder-processor-decoder pipeline described in Appendix D. For node and graph-level tasks, the pipeline is applied directly with the task-specific encoder described below. Since the graph transformer baseline expects node-level tokenization, we provide an additional graph transform.

Since most of our tasks provide only node or edge features, we use learned vectors, as described in the encoder, for the remaining embeddings. Across all datasets, we employ one-layer linear embeddings for both integer and real-valued features. Depending on the node-, edge-, or graph-level tasks, the decoder applies linear layers that represent the classification or regression target. In the case of bridges, minimum spanning tree, Steiner trees, max clique and bipartite matching a binary classification target is used for node or edge level predictions. For flow and topological sorting, a one-dimensional regression target is used instead. For pretrained models used in size generalization experiments, we use the model directly as encoder, processor, and decoder. However, the input graph size is increased. In contrast to complete training examples, we use a few-shot scenario with only 1 000 graphs to enable fast inference. In addition, we do not consider the flow task here because it is a regression setting. Throughout all experiments, we use the same random seed for a pretrained model to enable comparison to the 128-node test set.

Results We report results on all datasets for each of the three selected difficulties. These are denoted using the terms EASY, MEDIUM, and HARD to reflect the additional generalization requirements introduced by out-of-distribution sampling. For classification tasks, we report F1 scores, and for regression tasks, we report MAE. All results are grouped by their respective estimated task difficulty in Table 22 and Table 23 below. Additionally, we report size generalization results using pre-trained

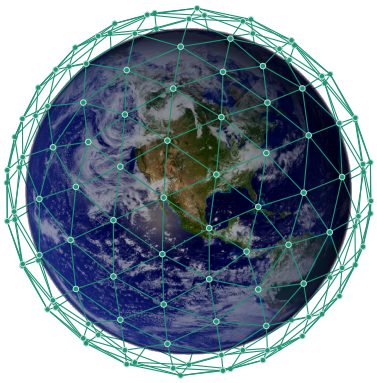

Figure 8. Down-scaled version of our node icosahedron spanning the globe. Each node contains weather variables of the corresponding location across 13 pressure levels.

baselines from the MEDIUM setting of our training procedure. For this, we evaluate each pre-trained model on graphs with 128 to 512 nodes. Results are provided in Table 24. For the minimum spanning tree, max clique, and max matching datasets, the graph transformer baseline shows improved performance. However, on the max flow, bridges, and Steiner tree datasets, the GIN baseline model performs better across the proposed difficulty levels. Furthermore, we observe a reduction in F1 scores as difficulty increases across datasets, highlighting variation in generator selection in the training data. However, the differences are not as pronounced as expected for the bridges dataset. We further observe that the GIN baseline is inconsistent across seeds for the minimum spanning tree task, whereas the graph transformer baseline does not exhibit similar issues. Additionally the GNNPlus baselines provide improved performance over the GIN baseline model in minimum spanning tree, steiner trees, maximum matching and max clique. However, we note that this advantage does not appear across all selected tasks with singular tasks such as topological sorting providing worse results than GIN or GT baselines. We further observe that there is no clear overall best performing architecture over all tasks, with results indicating that the architecture selection is depending on the algorithmic reasoning task. As an additional result, differences between difficulty levels are not as pronounced as expected from the data distribution shift. In most tasks increasing the difficulty level only reduces F1 scores marginally. Flow is the exception as a task here as MAE values increase significantly with an increasing distribution shift.

Overall, the performance of all baselines is lowest on Steiner trees and max clique tasks, indicating that these tasks are more complex for graph learning baselines. Similar to the results obtained in the CLRS benchmark (Velickovic et al., 2022) and by Müller et al. (2024) for an expressive graph transformer architecture, MST, bridges, and topological sorting provide more manageable tasks for graph learning baselines. We note that, despite previous benchmarks such as CLRS, there are currently no other baseline results on the synthetic algorithmic reasoning datasets we provide.

In the case of size generalization, we observe that both baselines are relatively robust to scaling of the test graph size. However, we observe that the generalization capability with respect to graph size is task-dependent, with the minimum spanning tree and Steiner tree tasks improving with larger graphs. Nonetheless, we note that this behavior may be observed due to shifts in average degree and node connectivity, using the same parameters as in test set generation, since each selected graph generator scales with the number of nodes. Nonetheless, performance is decreasing for topological sorting, max matching, and max clique tasks, highlighting the inability of our selected baselines to be size-generalization invariant across all tasks.

B.4. Earth systems

B.4.1. Weather forecasting: Medium-range atmospheric state prediction

Figure 8 provides an intuitive image of how the graph icosahedron spans the globe.

Related work The idea of utilizing machine learning models for weather forecasting first emerged in 2020, when (Rasp et al., 2020) and (Weyn et al., 2020) employed CNN architectures to predict global weather at resolutions of 5.625° and 1.9°, respectively. A key breakthrough occurred when (Keisler, 2022) first applied MPNNs to more naturally represent the globe, enabling 3D state predictions 6 hours ahead and facilitating multi-day forecasts. This approach has already achieved a skill

Table 21. Resulting F1 scores for algorithmic reasoning datasets, including minimum spanning tree, bridge finding and Steiner tree computation.

Task	Difficulty	Model	F1
MST	EASY	GIN	0.6906± 0.1655
		GT	0.8566± 0.0068
		GIN+	0.8225± 0.0020
		GCN+	0.8310± 0.0028
		GATEDGCN+	0.8485± 0.0220
	MEDIUM	GIN	0.7288± 0.0894
		GT	0.8504± 0.0148
		GIN+	0.7802± 0.0286
		GCN+	0.5317± 0.0318
		GATEDGCN+	0.8068± 0.0315
	HARD	GIN	0.6107± 0.2015
		GT	0.8421± 0.0115
		GIN+	0.8449± 0.0067
		GCN+	0.8554± 0.0098
		GATEDGCN+	0.8421± 0.0069
Bridges	EASY	GIN	0.9831± 0.0184
		GT	0.9269± 0.0103
		GIN+	0.9141± 0.0363
		GCN+	0.9221± 0.0151
		GATEDGCN+	0.9339± 0.0056
	MEDIUM	GIN	0.9622± 0.0077
		GT	0.8762± 0.0258
		GIN+	0.8288± 0.0363
		GCN+	0.9199± 0.0465
		GATEDGCN+	0.7007± 0.0930
	HARD	GIN	0.968± 0.0178
		GT	0.8897± 0.0304
		GIN+	0.9090± 0.0562
		GCN+	0.8893± 0.0565
		GATEDGCN+	0.6921± 0.0153
Steiner Trees	EASY	GIN	0.6691± 0.0288
		GT	0.6691± 0.0112
		GIN+	0.6656± 0.0111
		GCN+	0.7073± 0.0021
		GATEDGCN+	0.6589± 0.0130
	MEDIUM	GIN	0.5845± 0.0404
		GT	0.5672± 0.0790
		GIN+	0.4830± 0.0170
		GCN+	0.6388± 0.0188
		GATEDGCN+	0.6459± 0.0081
	HARD	GIN	0.6383± 0.0675
		GT	0.5212± 0.0219
		GIN+	0.6639± 0.0030
		GCN+	0.6933± 0.0046
		GATEDGCN+	0.6638± 0.0107

Table 22. Resulting F1 scores for algorithmic reasoning datasets max clique and max matching.

Task	**Difficulty**	**Model**	F1
Max Clique	EASY	GIN	0.4584± 0.0101
		GT	0.5407± 0.0088
		GIN+	0.4635± 0.0096
		GCN+	0.4835± 0.0065
		GATEDGCN+	0.4687± 0.0194
	MEDIUM	GIN	0.3996± 0.0852
		GT	0.4859± 0.0017
		GIN+	0.4921± 0.0056
		GCN+	0.5186± 0.0439
		GATEDGCN+	0.4948± 0.0389
	HARD	GIN	0.4102± 0.0820
		GT	0.4868± 0.0013
		GIN+	0.4807± 0.0027
		GCN+	0.5385± 0.0171
		GATEDGCN+	0.5362± 0.0245
Max Matching	EASY	GIN	0.7527± 0.0051
		GT	0.7402± 0.0172
		GIN+	0.7017± 0.0687
		GCN+	0.7988± 0.0047
		GATEDGCN+	0.7939± 0.0074
	MEDIUM	GIN	0.6399± 0.0231
		GT	0.6915± 0.009
		GIN+	0.7158± 0.0188
		GCN+	0.6702± 0.0551
		GATEDGCN+	0.7210± 0.0224
	HARD	GIN	0.6595± 0.0164
		GT	0.6743± 0.0038
		GIN+	0.6333± 0.1262
		GCN+	0.6854± 0.0398
		GATEDGCN+	0.7310± 0.0322

Table 23. Resulting MAE for topological sorting and flow.

Task	**Difficulty**	**Model**	MAE
Topological Sorting	EASY	GIN	0.1001± 0.0089
		GT	0.116± 0.0058
		GIN+	0.1351± 0.0377
		GCN+	0.3064± 0.0300
		GATEDGCN+	0.1088± 0.0064
	MEDIUM	GIN	0.1537± 0.0031
		GT	0.1305± 0.0094
		GIN+	0.1346± 0.0029
		GCN+	0.3122± 0.0035
		GATEDGCN+	0.1257± 0.0024
	HARD	GIN	0.1301± 0.0046
		GT	0.1532± 0.0214
		GIN+	0.1157± 0.0024
		GCN+	0.3004± 0.0391
		GATEDGCN+	0.1100± 0.0024
Flow	EASY	GIN	3.4387± 0.0631
		GT	4.2737± 0.0646
		GIN+	5.0902± 0.2041
		GCN+	4.4852± 0.0551
		GATEDGCN+	3.0029± 0.0205
	MEDIUM	GIN	9.5960± 0.1707
		GT	6.3786± 0.4262
		GIN+	9.4751± 0.4324
		GCN+	8.5583± 0.9136
		GATEDGCN+	9.8846± 0.3982
	HARD	GIN	9.5061± 0.1265
		GT	6.4833± 0.0869
		GIN+	10.3892± 0.3140
		GCN+	8.9073± 1.2009
		GATEDGCN+	8.6309± 1.4895

Table 24. Size generalization results on algorithmic reasoning datasets from Section 3.3.3. Each column represents an evaluation on 1000 graphs with the given number of nodes (128–512). For each experiment, the same generation parameters were used as in the results presented in Table 22 and Table 23. OOT denotes the case where the underlying size generalization data was not computed within 24 hours of computation on a single cluster node. All generated graphs are based on the MEDIUM difficulty setting. We use the same single seed for each pretrained model in each task to obtain inference results across different sizes.

Dataset (Score)	**Model**	128	192	256	384	512
Topological Sorting (MAE)	GT	0.1346	0.1730	0.1827	0.2018	0.2071
	GIN	0.1492	0.181	0.1981	0.2015	0.2179
MST (F1)	GT	0.8720	0.8773	0.8891	0.8874	0.8817
	GIN	0.6103	0.8132	0.8605	0.8765	0.8823
Bridges (F1)	GT	0.8799	0.8842	0.8959	0.9049	0.9118
	GIN	0.9579	0.9213	0.9190	0.9203	0.9212
Steiner Trees (F1)	GT	0.5160	0.5322	0.5221	0.5762	0.5578
	GIN	0.5499	0.5739	0.6338	0.6651	0.6502
Max Clique (F1)	GT	0.4877	0.4849	0.4890	0.4915	0.4931
	GIN	0.3496	0.3112	0.3148	0.2926	0.2673
Max Matching (F1)	GT	0.7010	0.6552	0.6206	0.5770	OOT
	GIN	0.6271	0.6326	0.6372	0.6382	OOT

comparable to that of 1° global NWP (GFS/ECMWF) on specific metrics, outperforming prior data-driven models. Building on this idea, (Lam et al., 2023) introduced *GraphCast*, a multi-scale MPNN-based system that delivers today's forecasts at 0.25° resolution. GraphCast achieved unprecedented accuracy, outperforming even ECMWF's *high-resolution deterministic model* (HRES) out to ten days. The success of these deterministic forecasts raised the question of probabilistic prediction. (Price et al., 2023) introduced *GenCast*, a generative weather model that produces an ensemble of forecasts rather than a single deterministic run. For this, they used a diffusion-based approach built on an MPNN backbone, outperforming ECMWF's *ensemble system* (ENS), the world's top operational probabilistic forecast on over 97% of evaluated metrics. Several other advances can be noted. (Oskarsson et al., 2024) have proposed Graph-EFM, a model featuring a hierarchical GNN that seamlessly handles global and regional forecasts. With *OneForecast*, (Gao et al., 2025) introduced an MPNN framework that uses nested multi-scale graphs and adaptive message passing to improve local extreme-event predictions within a global context. These works indicate a trend toward hybrid solutions that densify graphs in target regions for high-resolution detail while maintaining global consistency. Current state-of-the-art models, such as Pangu-Weather (Bi et al., 2022), Aurora (Bodnar et al., 2024), and FGN (Alet et al., 2025), all present excellent results that, in many areas, far exceed those of conventional NWP methods. It is important to note, however, that while machine learning-based approaches exceed NWP skill on tested time scales, they do not explicitly enforce conservation laws. Over long integration horizons, this can lead to subtle physical drifts that may limit specific climate-related uses. This becomes particularly relevant under regime shifts, for example, caused by climate change, that lie outside the distributions the models were trained on. Furthermore, compared with NWP models, ML-based approaches offer very limited interpretability and function mainly as black boxes.

B.5. Limitations

In the following we discuss limitations of each dataset and the proposed baseline methods. We highlight additional dataset specific limitations and results in the respective sections above. For bluesky the biggest limitations are graph sizes across tasks in the dataset, as the graph size prevents running graph transformer baselines and larger GNNs on them.

CO and algorithmic reasoning datasets are purely synthetically generated as the collection of real world instances for each task constitutes a major challenge. However, both datasets and their evaluation framework can be expanded to include real world data as well in training or testing.

For weather forecasting and SAT solving graph sizes prevent many computationally intensive methods to be run on them. In addition weather forecasting data is known to require substantial preprocessing as outlined by Lam et al. (2023). Aside from

small instances SAT instances are currently too large to run available solvers on them to generate node features and targets.

Lastly, the chip design task is currently very challenging for learning based models resulting in no currently available baselines for this task. However, we include non learning baselines to showcase the potential performance a learning based method should achieve.

Due to the amount of tasks evaluated as part of the benchmark we restricted ourselves to common GNN baselines from Luo et al. (2025) as they achieve good performance across currently available benchmarks. This also limits the application of HPO (see Appendix E) and the size of hyperparameter tuning budgets. Furthermore, we provide a simple graph transformer baseline and. Since there are to our knowledge no widely available graph foundation models which work across tasks and evaluation targets, including node, edge and graph level tasks we decided to not include foundation models so far.

C. Hardware

C.1. SAT solving

We ran the SAT solvers on a cluster equipped with two AMD EPYC 7 543 32-core processors, each with 512MB of L3 cache and 1 TB of RAM, running Rocky Linux 9. We allowed each solver to use up to 64GB of memory, so up to 8 solvers could run on the same node in parallel. Before conducting the experiments, we performed a preliminary experiment, running Kissat on 100 random SAT instances 10 times, ensuring consistent results (in terms of computation time).

For training the GNNs, we used a cluster with 4 GPU nodes, each containing 4 H100 NVL GPUs with 94GB of vRAM. Each task for GIN and GRASS ran for two to three hours and ten hours for the GNNPlus models. Furthermore, the HPO sweep outlined in Appendix E was conducted over 60 configurations.

C.2. Algorithmic reasoning

Each algorithmic reasoning task was evaluated on a cluster node equipped with one NVIDIA L40 GPU, featuring 48GB of vRAM and 512GB of RAM. Throughout all experiments, the same cluster configuration was used to ensure comparable wall-clock times. For each single task a single NVIDIA L40 GPU with 120GB of RAM was used. Due to computational time constraints, positional and structural encodings (such as RWSE and LPE) and edge transformations for the graph transformer baseline were provided beforehand. Runtimes across tasks were very similar, with GNNPlus baselines running for 20 minutes each after dataset preprocessing. The GIN baseline ran for 29 minutes on each task.

The edge level task preprocessing as outlined in subsection D.4 took between 1 hour and 40 minutes for the maximum matching task and 4 hours for the minimum spanning tree task. For preprocessing, the complete cluster node was used.

C.3. Weather forecasting

For weather forecasting, a cluster node with four NVIDIA L40 GPUs was used during evaluation. Additionally, a cluster node with four NVIDIA A100 GPUs was utilized during training to accommodate larger batch sizes. Both cluster nodes use 512GB of RAM. Training took 1 day 18 hours and 35 minutes for a single seed.

C.4. Combinatorial Optimization

Experiments for unsupervised CO were performed on an NVIDIA H100 GPU with 80GB of vRAM, and for supervised learning were on an NVIDIA L40S GPU with 48 GB vRAM. Each run on the GNNPlus models took 14 hours on average for the small datasets and for the large instances 55 hours.

C.5. Electronic Circuits

We conducted all experiments on a single NVIDIA L40 GPU with 120GB of RAM. Furthermore, RWSE was computed during data loading. The GT and GIN baseline evaluation took 15 minutes for 5 components and 13 minutes for 7 components, and 12 minutes for 10 components, respectively. GNNPlus baselines took 13 minutes per run for 5 and 7 components, and 11 minutes for 10 components.

C.6. Social networks

We conducted all experiments on a single NVIDIA A100 GPU with 120GB of RAM on the same cluster node. Training and testing of GNNPlus methods took on average 1 hour, with models on the Bluesky quotes task running longer at 2 hours.

D. Experimental setup

Similar to (Bechler-Speicher et al., 2025; Stoll et al., 2025), we provide an encoder-processor-decoder architecture for all tasks, which we detail in the following. While the task-specific components of our architecture vary, we provide a general architecture to measure the performance of selected baselines on our tasks. For the chip design and weather forecast dataset, we adapt this architecture to provide the necessary task-dependent encoder and decoder functionality.

Encoder For each task, we provide a task-specific encoder that derives encodings of node- and edge-level features with a common dimension $d \in \mathbb{R}^+$. These features are then passed to the processor as node and/or edge features. For models that require tokenized input, we provide node-level tokens. For missing node or edge features, a learnable vector of dimension d is used in their place. In case of the graph transformer baseline, we add a `[cls]` token for graph-level representations, following (Bechler-Speicher et al., 2025; Stoll et al., 2025).

Processor As our processor, we select one of the baseline models we evaluate. These consist of two dummy baselines: an MLP and a DeepSet architecture; a GNN baseline using a GIN architecture; and a Graph Transformer baseline. Independent of the selected baseline, we compute updated node and graph representations at each layer as follows, given a graph G with node representations $\boldsymbol{X}$,

$$\begin{aligned} \boldsymbol{X} &= \phi(\mathsf{LayerNorm}(\boldsymbol{X}), G) \\ \boldsymbol{X} &= \boldsymbol{X} + \mathsf{MLP}(\mathsf{LayerNorm}(\boldsymbol{X})) \end{aligned} \tag{2}$$

with the following MLP layer:

$$\mathsf{MLP}(\boldsymbol{x}) \coloneqq \sigma(\mathsf{LayerNorm}(\boldsymbol{x})\boldsymbol{W}_1)\boldsymbol{W}_2,$$

where σ denotes the GeLU nonlinearity and $\boldsymbol{W}_1, \boldsymbol{W}_2$ are weight matrices. Further, ϕ is the selected baseline, and MLP denotes a two-layer multilayer perceptron with GELU non-linearity (Hendrycks & Gimpel, 2016). For the graph transformer baseline, we employed biased attention as described by (Bechler-Speicher et al., 2025; Stoll et al., 2025). Furthermore, we incorporate RWSE (Dwivedi et al., 2022a) and LPE (Müller & Morris, 2024) as node positional encodings for our GNN and graph transformer baselines.

Decoder Since each task requires task-specific predictions, we provide a decoder for each task. Across all functions, we use the same decoder layout, differing only in the predictions they make. We propose a decoder two-layer multilayer perceptron for our baseline models consisting of two linear layers $\boldsymbol{W}_1 \in \mathbb{R}^{d \times d}, \boldsymbol{W}_2 \in \mathbb{R}^{d \times o}$ as well as GELU non-linearity, with o denoting the task-specific output dimension, i.e.,

$$\boldsymbol{W}_2(\mathsf{LayerNorm}(\mathsf{GELU}(\boldsymbol{W}_1 x)).$$

Optionally, a bias term can be added to the decoder.

Pretraining In addition to training models using our encoder-processor-decoder architecture, we provide support for evaluating pretrained models on specific tasks. These are further specified in each subsection concerning additional evaluation benchmarks.

Evaluation protocol Using the aforementioned hyperparameter tuning process, we evaluate all baseline models across three random seeds for each task and report the mean performance and standard deviation. For out-of-distribution generalization tasks, we utilize pretrained models from these baseline evaluations.

D.1. Hyperparameter selection

Here, we provide the hyperparameters used in training our baseline models for each task. We note that in most cases, no extensive hyperparameter search was conducted for the baseline models. We highlight the hyperparameters included in the sweep for each dataset below:

For blueksy datasets we sweeped over learning rates $\{1e^{-3}, 2e^{-3}, 5e^{-4}\}$ for all models and for the number of layers $l \in \{2,3,4,6,8\}$ and hidden dimension $\{128,192,384\}$. We further included residual connections and feed-forward MLP layers in all GNNPlus architectures.

Regarding electronic circuits, we provide results obtained from a hyperparameter sweep over learning rates $\{1e^{-3},2e^{-3},5e^{-4}\}$, RWSE encoding dimension $\{4,8,16\}$, and number of layers $\{3,6\}$.

For CO datasets we sweep over the following learning rates $\{1e^{-3},2e^{-3},5e^{-4}\}$ and use the six layers across all models.

Across all algorithmic reasoning tasks, we conducted hyperparameter tuning with the following selection of hyperparameters. For GIN and GT over learning rates $\{1e^{-4},2e^{-4},3e^{-4}\}$ with the exception of steiner tree tasks, where we also considered $\{1e^{-3},2e^{-3}\}$. Furthermore, layer choices are given by the following $\{6,16\}$ and PE encoding dimension and type: $\{4,8,16,32\}$ and $\{RWSE, LPE\}$. In contrast to this, the GNNPlus methods are tuned with the following hyperparameter budget: learning rates $\{1e^{-3},2e^{-3},5e^{-4}\}$, PE encoding dimension $\{4,8,16\}$ and layer choices $\{3,4\}$

For SAT solving, we used automatic hyperparameter selection via SMAC3. The process is outlined in Appendix E.

For weather forecasting, we only swept over the following learning rates: $\{1e-4,2e-4,5e-5\}$.

Table 25. List of hyperparameters used for baseline models in BlueSky datasets.

Hyperparam.	**Variable**	$\text{GNN}_{\text{quotes}}$	$\text{MLP}_{\text{quotes}}$	$\text{Deep Sets}_{\text{quotes}}$	$\text{GNN}_{\text{replies}}$	$\text{MLP}_{\text{replies}}$	$\text{Deep Sets}_{\text{replies}}$	$\text{GNN}_{\text{reposts}}$	$\text{MLP}_{\text{reposts}}$	$\text{Deep Sets}_{\text{reposts}}$
General	Learning rate	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3
	Weight decay	0.	0.	0.	0.	0.	0.	0.	0.	0.
	Dropout	0.	0.2	0.	0.	0.	0.	0.	0.2	0.2
	Epochs	1 000	1 000	1 000	1 000	1 000	1 000	1 000	1 000	1 000
Architecture	Layers	4	2	4	4	4	4	4	2	2
	Hidden dim.	128	128	128	128	128	128	128	128	128

Table 26. Incumbent hyperparameters (AS:LCG). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	1.17e-05	0.00183706	4.59e-04	4.76e-04	2.23e-05	2.29e-05
	Batch size	16	16	16	16	16	16
	Weight decay	1.86e-08	0.0515994	3.23e-05	1.35e-08	1.43e-07	3.28e-06
	Dropout	0.0591372	0.0716766	0.144238	0.0812465	0.240447	0.237349
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	13134	9622	2134	11146	9467	11215
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	256	512	256	384	512	256
	Layers	5	6	7	7	7	9
	Activation	relu	relu	relu	relu	relu	relu

D.2. Baseline Architectures

In the following, we provide implementation details on the baselines used for the datasets in GRAPHBENCH. For dataset-specific design choices, we provide detailed information in Appendix D.3.

Graph transformer architecture As described in Appendix D we use an encoder-processor-decoder baseline across tasks. In this case, we consider a graph transformer as the processor architecture following the implementation outlined in Bechler-Speicher et al. (2025) and Stoll et al. (2025). For most tasks, we use node-level tokenization, where each graph node is treated as a single token input to the graph transformer. However, for edge-level tasks, we use the transformation outlined for algorithmic reasoning tasks, allowing edge-level tokens to be used without changes to the processor architecture. Additionally, absolute PEs, such as RWSE or LPE, are added to the node embeddings before the first graph transformer layer. Then, the graph transformer layer computes full multi-head scaled-dot-product attention, adding an attention bias $\boldsymbol{B}$ to the unnormalized attention matrix and applying softmax to it. Let $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{L\times d}$ and $\boldsymbol{B} \in \mathbb{R}^{L\times L}$ with L denoting the

Table 27. Hyperparameters for baseline models on Bluesky social graph tasks: Quotes (Q), Replies (Re), and Reposts (Rp). Configurations are derived from GNN+ architectural settings for GCN, GIN, and GatedGCN.

Hyperparam.	**Variable**	GCN_Q	GIN_Q	GatedGCN_Q	GCN_{Re}	GIN_{Re}	GCN_{Rp}	GIN_{Rp}
General	Learning rate	5e-4	2e-3	5e-4	2e-3	2e-3	2e-3	5e-4
	Batch size	1	1	1	1	1	1	1
	Weight decay	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	Dropout	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	Steps	1,000	1,000	1,000	1,000	1,000	1,000	1,000
Optimizing	Warmup steps	50	50	50	50	50	50	50
	Scheduler	CosineAnnealingLR						
	Optimizer	AdamW						
	Param.	2.51M	1.44M	1.44M	0.72M	0.78M	0.78M	0.78M
PE	Type	None	None	None	None	None	None	None
	Encoding dim.	384	384	384	384	384	384	384
	Hidden dim.	384	384	384	384	384	384	384
Architecture	Hidden dim.	384	192	192	192	128	192	192
	Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
	Residual	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Layer Norm	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Num. layers	3	6	3	8	3	3	3

Table 28. Incumbent hyperparameters (AS:VCG). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	4.59e-04	0.00183706	0.00325039	4.76e-04	0.00325039	6.34e-05
	Batch size	16	16	16	16	16	16
	Weight decay	3.23e-05	0.0515994	2.48e-08	1.35e-08	2.48e-08	0.0467242
	Dropout	0.144238	0.0716766	0.352207	0.0812465	0.352207	0.308829
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	2134	9622	14173	11146	14173	7011
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	256	512	256	384	256	384
	Layers	7	6	9	7	9	7
	Activation	relu	relu	relu	relu	relu	relu

Table 29. Incumbent hyperparameters (AS:VG). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	0.036691	2.62e-04	0.00126235	0.00133172	9.74e-05	9.04e-04
	Batch size	16	16	16	16	16	16
	Weight decay	0.00279961	0.0123051	1.70e-05	0.0273136	3.95e-08	0.00392744
	Dropout	0.274407	0.357595	0.211827	0.150787	0.464041	0.272442
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	7036	3163	8877	14310	5652	5851
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	384	512	256	256	256	512
	Layers	9	5	5	6	5	4
	Activation	relu	relu	relu	relu	relu	relu

Table 30. Incumbent hyperparameters (EPM:LCG AMSAT). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	0.00379986	0.00153328	0.002836	4.76e-04	0.0157381	3.34e-04
	Batch size	16	16	16	16	16	16
	Weight decay	2.51e-07	5.03e-08	6.49e-04	1.35e-08	2.60e-05	1.76e-04
	Dropout	0.434063	0.0095966	0.271249	0.0812465	0.0159195	0.448019
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	12042	6846	9352	11146	14634	2913
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	256	512	256	384	256	512
	Layers	6	7	7	7	8	9
	Activation	relu	relu	relu	relu	relu	relu

Table 31. Incumbent hyperparameters (EPM:LCG BREAKIDKISSAT). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	0.010885	0.00183706	0.00379986	4.76e-04	3.24e-05	2.29e-05
	Batch size	16	16	16	16	16	16
	Weight decay	0.0719626	0.0515994	2.51e-07	1.35e-08	1.44e-05	3.28e-06
	Dropout	0.207184	0.0716766	0.434063	0.0812465	0.317529	0.237349
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	1973	9622	12042	11146	19819	11215
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	384	512	256	384	512	256
	Layers	4	6	6	7	6	9
	Activation	relu	relu	relu	relu	relu	relu

Table 32. Incumbent hyperparameters (EPM:LCG KISSAT). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	0.002836	0.00153328	0.0160222	4.76e-04	0.00126654	0.00598895
	Batch size	16	16	16	16	16	16
	Weight decay	6.49e-04	5.03e-08	3.69e-07	1.35e-08	1.89e-06	0.0146903
	Dropout	0.271249	0.0095966	0.06191	0.0812465	0.390265	0.0823471
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	9352	6846	7177	11146	15627	12605
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	256	512	384	384	512	256
	Layers	7	7	9	7	8	9
	Activation	relu	relu	relu	relu	relu	relu

Table 33. Incumbent hyperparameters (EPM:LCG KISSATMABDC). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	2.69e-05	0.00153328	0.00258292	4.76e-04	0.00379986	0.0157381
	Batch size	16	16	16	16	16	16
	Weight decay	3.22e-04	5.03e-08	0.0981588	1.35e-08	2.51e-07	2.60e-05
	Dropout	0.378053	0.0095966	0.145039	0.0812465	0.434063	0.0159195
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	10680	6846	3775	11146	12042	14634
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	256	512	512	384	256	256
	Layers	4	7	10	7	6	8
	Activation	relu	relu	relu	relu	relu	relu

Table 34. Incumbent hyperparameters (EPM:VCG AMSAT). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	0.010885	0.00153328	0.00153328	4.76e-04	1.51e-04	2.62e-04
	Batch size	16	16	16	16	16	16
	Weight decay	0.0719626	5.03e-08	5.03e-08	1.35e-08	9.71e-05	0.0123051
	Dropout	0.207184	0.0095966	0.0095966	0.0812465	0.459118	0.357595
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	1973	6846	6846	11146	14683	3163
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	384	512	512	384	512	512
	Layers	4	7	7	7	10	5
	Activation	relu	relu	relu	relu	relu	relu

Table 35. Incumbent hyperparameters (EPM:VCG BREAKIDKISSAT). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	0.0416586	0.00153328	0.00183706	4.59e-04	0.00258292	2.29e-05
	Batch size	16	16	16	16	16	16
	Weight decay	0.0492703	5.03e-08	0.0515994	3.23e-05	0.0981588	3.28e-06
	Dropout	0.216644	0.0095966	0.0716766	0.144238	0.145039	0.237349
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	17409	6846	9622	2134	3775	11215
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	256	512	512	256	512	256
	Layers	4	7	6	7	10	9
	Activation	relu	relu	relu	relu	relu	relu

Table 36. Incumbent hyperparameters (EPM:VCG KISSAT). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	0.002836	0.00153328	0.00153328	4.76e-04	0.00258292	0.00598895
	Batch size	16	16	16	16	16	16
	Weight decay	6.49e-04	5.03e-08	5.03e-08	1.35e-08	0.0981588	0.0146903
	Dropout	0.271249	0.0095966	0.0095966	0.0812465	0.145039	0.0823471
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	9352	6846	6846	11146	3775	12605
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	256	512	512	384	512	256
	Layers	7	7	7	7	10	9
	Activation	relu	relu	relu	relu	relu	relu

Table 37. Incumbent hyperparameters (EPM:VCG KISSATMABDC). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	4.59e-04	4.76e-04	0.00258292	0.00126235	1.51e-04	0.0369504
	Batch size	16	16	16	16	16	16
	Weight decay	3.23e-05	1.35e-08	0.0981588	1.70e-05	9.71e-05	0.0683645
	Dropout	0.144238	0.0812465	0.145039	0.211827	0.459118	0.310739
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	2134	11146	3775	8877	14683	1423
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	256	384	512	256	512	384
	Layers	7	7	10	5	10	6
	Activation	relu	relu	relu	relu	relu	relu

Table 38. Incumbent hyperparameters (EPM:VG AMSAT). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	0.036691	0.00133172	0.0029426	0.0029426	0.0369504	0.010885
	Batch size	16	16	16	16	16	16
	Weight decay	0.00279961	0.0273136	8.44e-04	8.44e-04	0.0683645	0.0719626
	Dropout	0.274407	0.150787	0.108411	0.108411	0.310739	0.207184
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	7036	14310	18592	18592	1423	1973
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	384	256	512	512	384	384
	Layers	9	6	4	4	6	4
	Activation	relu	relu	relu	relu	relu	relu

Table 39. Incumbent hyperparameters (EPM:VG BREAKIDKISSAT). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	0.036691	0.00379986	0.00260718	4.76e-04	0.002836	0.010885
	Batch size	16	16	16	16	16	16
	Weight decay	0.00279961	2.51e-07	9.99e-04	1.35e-08	6.49e-04	0.0719626
	Dropout	0.274407	0.434063	0.330087	0.0812465	0.271249	0.207184
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	7036	12042	16037	11146	9352	1973
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	384	256	512	384	256	384
	Layers	9	6	8	7	7	4
	Activation	relu	relu	relu	relu	relu	relu

Table 40. Incumbent hyperparameters (EPM:VG KISSAT). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	0.036691	0.00260718	4.76e-04	0.0029426	0.0369504	0.010885
	Batch size	16	16	16	16	16	16
	Weight decay	0.00279961	9.99e-04	1.35e-08	8.44e-04	0.0683645	0.0719626
	Dropout	0.274407	0.330087	0.0812465	0.108411	0.310739	0.207184
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	7036	16037	11146	18592	1423	1973
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	384	512	384	512	384	384
	Layers	9	8	7	4	6	4
	Activation	relu	relu	relu	relu	relu	relu

Table 41. Incumbent hyperparameters (EPM:VG KISSATMABDC). List of hyperparameters used for baseline models in SAT solving tasks. These include parameters for positional encoding (PE), the architecture (GIN), and general learning parameters.

Hyperparam.	Variable	Gated GCN+	GCN	GCN+	GIN	GIN+	GRASS
	Learning rate	0.036691	4.59e-04	0.00260718	0.00183706	0.0369504	0.00379986
	Batch size	16	16	16	16	16	16
	Weight decay	0.00279961	3.23e-05	9.99e-04	0.0515994	0.0683645	2.51e-07
	Dropout	0.274407	0.144238	0.330087	0.0716766	0.310739	0.434063
	Steps	100000	100000	100000	100000	100000	100000
	Warmup steps	7036	2134	16037	9622	1423	12042
	Scheduler	cosine	cosine	cosine	cosine	cosine	cosine
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	PE Type	-	-	-	-	-	-
	Encoding dim.	-	-	-	-	-	-
	Hidden dim.	384	256	512	512	384	256
	Layers	9	7	8	6	6	6
	Activation	relu	relu	relu	relu	relu	relu

Table 42. List of hyperparameters used for baseline models in algorithmic reasoning tasks. These include parameters for positional encoding (PE), architecture (GT/GIN), and general learning. The respective abbreviations denote the following tasks: topological sorting (TO), minimum spanning tree (MST), bridges (BR), and Steiner tree (ST).

Hyperparam.	**Variable**	GIN_{TO}	GT_{TO}	GIN_{MST}	GT_{MST}	GIN_{BR}	GT_{BR}	GIN_{ST}	GT_{ST}
General	Learning rate	1e-4	1e-4	3e-4	3e-4	2e-4	2e-4	2e-3	1e-4
	Batch size	256	256	256	256	256	256	256	256
	Weight decay	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	Dropout	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	Steps	11 880	11 880	11 880	11 880	11 880	11 880	11 880	11 880
Optimizing	Warmup steps	1 000	1 000	1 000	1 000	1 000	1 000	1 000	1 000
	Scheduler	CosineAnnealingLR							
	Optimizer	AdamW, betas=(0.9,0.999)							
	Param.	2.55M	14.93M	2.68M	15.07M	2.55M	14.93M	1.66M	14.93M
PE	Type	RWSE	RWSE	LPE	LPE	RWSE	RWSE	RWSE	RWSE
	Num. steps/Eigvals	16	16	32	32	16	16	16	16
	Encoding dim.	384	384	384	384	384	384	384	384
	Hidden dim.	768	768	384	384	768	768	768	768
	Num. enc. layers	2	2	2/2	2/2	2	2	2	2
Architecture	Layers	6	16	6	16	6	16	3	16
	Hidden dim.	384	384	384	384	384	384	384	384
	Attn. heads	0	16	0	16	0	16	0	16
	Activation	GELU	GELU	GELU	GELU	GELU	GELU	GELU	GELU

Table 43. Hyperparameters for GNNPlus baseline models across algorithmic reasoning tasks: topological sorting (TO) and Steiner tree (ST). The model variants are defined by their GNNPlus layer type.

Hyperparam.	**Variable**	GIN_{TO}	GatedGCN_{TO}	GCN_{TO}	GIN_{ST}	GatedGCN_{ST}	GCN_{ST}
General	Learning rate	2e-3	2e-3	2e-3	1e-3	2e-3	2e-3
	Batch size	256	256	256	256	256	256
	Weight decay	0.1	0.1	0.1	0.1	0.1	0.1
	Dropout	0.1	0.1	0.1	0.1	0.1	0.1
	Steps	11 880	11 880	11 880	11 880	11 880	11 880
Optimizing	Warmup steps	1 000	1 000	1 000	1 000	1 000	1 000
	Scheduler	CosineAnnealingLR					
	Optimizer	AdamW, betas=(0.9, 0.999)					
	Param.	3.11M	3.11M	3.11M	3.11M	3.11M	3.11M
PE	Type	RWSE	RWSE	RWSE	RWSE	RWSE	RWSE
	RWSE steps	16	16	16	16	16	16
	Hidden dim.	384	384	384	384	384	384
	Num. layers	2	2	2	2	2	2
Architecture	Hidden dim.	384	384	384	384	384	384
	Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
	FFN type	MLP	MLP	MLP	MLP	MLP	MLP
	Residual	Yes	Yes	Yes	Yes	Yes	Yes
	Layer Norm	Yes	Yes	Yes	Yes	Yes	Yes
	Num. layers	3	3	3	3	3	3

Table 44. Hyperparameters for GNNPlus baseline models across algorithmic reasoning tasks: bridges (BR) and minimum spanning tree (MST). The model variants are defined by their GNNPlus layer type.

Hyperparam.	**Variable**	GIN_{BR}	GatedGCN_{BR}	GCN_{BR}	GIN_{MST}	GatedGCN_{MST}	GCN_{MST}
General	Learning rate	1e-3	1e-3	1e-3	2e-3	2e-3	2e-3
	Batch size	256	256	256	256	256	256
	Weight decay	0.1	0.1	0.1	0.1	0.1	0.1
	Dropout	0.1	0.1	0.1	0.1	0.1	0.1
	Steps	11 880	11 880	11 880	11 880	11 880	11 880
Optimizing	Warmup steps	1 000	1 000	1 000	1 000	1 000	1 000
	Scheduler	CosineAnnealingLR					
	Optimizer	AdamW, betas=(0.9, 0.999)					
	Param.	3.11M	3.11M	3.11M	3.11M	3.11M	3.11M
PE	Type	RWSE	RWSE	RWSE	RWSE	RWSE	RWSE
	RWSE steps	8	16	16			
	Hidden dim.	384	384	384	384	384	384
	Num. layers	2	2	2	2	2	2
Architecture	Hidden dim.	384	384	384	384	384	384
	Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
	FFN type	MLP	MLP	MLP	MLP	MLP	MLP
	Residual	Yes	Yes	Yes	Yes	Yes	Yes
	Layer Norm	Yes	Yes	Yes	Yes	Yes	Yes
	Num. layers	3	3	3	3	3	3

Table 45. Hyperparameters for GNNPlus baseline models across algorithmic reasoning tasks: flow (FL), maximum matching (MM) and max clique (MC). The model variants are defined by their GNNPlus layer type.

Hyperparam.	**Variable**	GIN_{FL}	GatedGCN_{FL}	GCN_{FL}	GIN_{MM}	GatedGCN_{MM}	GCN_{MM}	GIN_{MC}	GatedGCN_{MC}	GCN_{MC}
General	Learning rate	2e-3	2e-3	2e-3	1e-3	2e-3	1e-3	1e-3	1e-3	1e-3
	Batch size	256	256	256	256	256	256	256	256	256
	Weight decay	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	Dropout	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	Steps	11 880	11 880	11 880	11 880	11 880	11 880	11 880	11 880	11 880
Optimizing	Warmup steps	1 000	1 000	1 000	1 000	1 000	1 000	1 000	1 000	1 000
	Scheduler	CosineAnnealingLR								
	Optimizer	AdamW, betas=(0.9, 0.999)								
	Param.	3.11M	3.11M	3.11M	3.11M	3.11M	3.11M	3.11M	3.11M	3.11M
PE	Type	RWSE	RWSE	RWSE	RWSE	RWSE	RWSE	RWSE	RWSE	RWSE
	RWSE steps	16	16	16	16	16	16	16	16	16
	Hidden dim.	384	384	384	384	384	384	384	384	384
	Num. layers	2	2	2	2	2	2	2	2	2
Architecture	Hidden dim.	384	384	384	384	384	384	384	384	384
	Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
	FFN type	MLP	MLP	MLP	MLP	MLP	MLP	MLP	MLP	MLP
	Residual	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Layer Norm	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Num. layers	3	3	3	3	3	3	3	3	3

Table 46. List of hyperparameters used for baseline models in algorithmic reasoning and weather forecasting tasks. These include parameters for positional encoding (PE), architecture (GT/GIN), and general learning. The following tasks are denoted by the respective abbreviations: max clique (MC), flow (FL), maximum matching (MM), and weather forecasting (WE).

Hyperparam.	**Variable**	GIN_{MC}	GT_{MC}	GIN_{FL}	GT_{FL}	GIN_{MM}	GT_{MM}	GT_{WE}
General	Learning rate	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4
	Batch size	256	256	256	256	256	256	8
	Weight decay	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	Dropout	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	Steps	11 880	11 880	11 880	11 880	11 880	11 880	400 000
Optimizing	Warmup steps	1 000	1 000	1 000	1 000	1 000	1 000	20000
	Scheduler	CosineAnnealingLR						
	Optimizer	AdamW betas=(0.9,0.999)						
	Param.	2.68M	15.07M	2.55M	14.93M	2.55M	14.93M	17.15M
PE	Type	LPE	LPE	RWSE	RWSE	RWSE	RWSE	None
	Num. steps/Eigvals	16	16	32	32	16	16	0
	Encoding dim.	384	384	384	384	384	384	384
	Hidden dim.	384	384	768	768	768	768	768
	Num. enc. layers	2/2	2/2	2	2	2	2	2
Architecture	Layers	6	16	6	16	6	16	16
	Hidden dim.	384	384	384	384	384	384	384
	Attn. heads	0	16	0	16	0	16	16
	Activation	GELU	GELU	GELU	GELU	GELU	GELU	GELU

Table 47. List of hyperparameters used for baseline models in CO and electronic circuits (EC) tasks. These include parameters for positional encoding (PE), architecture (GT/GIN), and general learning.

Hyperparam.	**Variable**	GIN_{CO}	GT_{CO}	MLP	DeepSet	GT_{EC}	GIN_{EC}
General	Learning rate	1e-3	1e-3	1e-3	1e-3	5e-4	5e-4
	Batch size	256	256	256	256	128	128
	Weight decay	0.	0.	0.	0.	0.1	0.1
	Dropout	0.	0.	0.	0.	0.1	0.1
	Epochs	1 000	1 000	1 000	1 000	-	-
	Steps	-	-	-	-	20 000	20 000
	Patience	100	100	100	100	-	-
Optimizing	Warmup steps	0	0	0	0	100	100
	Optimizer	Adam	Adam	Adam	Adam	Adam	Adam
	Param.	2.1M	5.6M	1.2M	2.1M	6.08M	2.1M
PE	Type	RWSE	RWSE	RWSE	RWSE	RWSE	RWSE
	Num. steps/Eigvals	16	16	16	16	16	16
	Encoding dim.	384	384	384	384	384	384
	Hidden dim.	384	384	384	384	384	384
	Num. enc. layers	2	2	2	2	2	2
Architecture	Layers	6	6	6	6	6	6
	Hidden dim.	384	384	384	384	384	384
	Attn. heads	0	4	0	0	16	0
	Activation	GELU	GELU	GELU	GELU	GELU	GELU

Table 48. Hyperparameters for GNNPlus baseline models (GINE and GCN) for CO datasets.

Hyperparam.	**Variable**	GIN+	GCN+
General	Learning rate	1e-3	1e-3
	Batch size	32	32
	Weight decay	1e-5	1e-5
	Dropout	0.0	0.05
	Max Epochs	1 000	1 000
Optimizing	Warmup epochs	50	50
	Scheduler	Cosine with Warmup	
	Optimizer	AdamW	
	Param.	477k	413k
PE	Type	RWSE	RWSE
	RWSE steps	20	20
	Hidden dim.	28	28
	Layers	2	2
Architecture	Hidden dim.	80	70
	Activation	ReLU	ReLU
	FFN type	MLP	MLP
	Residual	Yes	Yes
	Layer Norm	Yes	Yes
	Num. layers	12	9

Table 49. Hyperparameters for GNNPlus baseline models (GatedGCN, GCN, GIN) applied to Electronic Circuit tasks (Efficiency and Output Voltage).

		GatedGCN		**GCN**		**GIN**	
Hyperparam.	**Variable**	EFF	VOUT	EFF	VOUT	EFF	VOUT
General	Learning rate	1e-3	2e-3	1e-3	2e-3	1e-3	2e-3
	Batch size	128	128	128	128	128	128
	Weight decay	0.1	0.1	0.1	0.1	0.1	0.1
	Dropout	0.1	0.1	0.1	0.1	0.1	0.1
	Steps	20 000	20 000	20 000	20 000	20 000	20 000
Optimizing	Warmup steps	100	100	100	100	100	100
	Scheduler	CosineAnnealingLR					
	Optimizer	AdamW, betas=(0.9, 0.999)					
	Param.	3.11M	3.11M	3.11M	3.11M	3.11M	3.11M
PE	Type	RWSE	RWSE	RWSE	RWSE	RWSE	RWSE
	RWSE steps	8	16	8	16	8	16
	Hidden dim.	384	384	384	384	384	384
	Num. layers	2	2	2	2	2	2
Architecture	Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
	FFN type	MLP	MLP	MLP	MLP	MLP	MLP
	Residual	Yes	Yes	Yes	Yes	Yes	Yes
	Layer Norm	Yes	Yes	Yes	Yes	Yes	Yes
	Hidden dim.	384	384	384	384	384	384
	Num. layers	3	3	3	3	3	3

number of tokens and d the embedding dimension. Then attention and a graph transformer layer take the form

$$\mathsf{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{B}) \coloneqq \mathsf{softmax}\big(d^{-\frac{1}{2}} \cdot \boldsymbol{Q}\boldsymbol{K}^T + \boldsymbol{B}\big)\boldsymbol{V},$$

$$\boldsymbol{X}^{t+1} \coloneqq \mathsf{MLP}\big(\mathsf{Attention}(\boldsymbol{X}^t\boldsymbol{W}_Q, \boldsymbol{X}^t\boldsymbol{W}_K, \boldsymbol{X}^t\boldsymbol{W}_V, \boldsymbol{B})\big),$$

where $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{d\times d}$ are learnable weight matrices and MLP denotes a two layer-MLP. In practice, we compute attention over multiple heads, allowing for different attention biases to be added to the attention matrix. With ϕ given by the multihead attention computation, a processor layer is provided by Equation (2). In the processor, multiple layers are stacked, enabling the pipeline shown in Appendix D.

GINE architecture Throughout this work, we use a GINE-based GNN baseline as the processor within the framework outlined in Appendix D. Following, Hu et al. (2020b) the GINE layer updates the node representations $h_v^{(t)}$ at iteration t as follows:

$$\boldsymbol{h}_v^{(t+1)} = \mathsf{N}\Big((1+\epsilon)\boldsymbol{h}_v^{(t)} + \sum_{u\in\mathcal{N}(v)} \mathsf{ReLU}(\boldsymbol{h}_v^{(t)} + e_{uv})\Big)$$

where $\mathcal{N}(v)$ denotes the neighborhood of a node v and N a neural network such as an MLP.

We apply the GINE processor layer in a similar way to the graph transformer baseline design outlined previously by replacing ϕ with a GINE layer where N is given by a dropout layer. First, the node embeddings, optionally with added PEs, are passed to the GINE layer, where layer normalization is applied. Then, the output of the GINE message-passing layer is forwarded to a two-layer MLP. We use the same residual connection as seen in the GIN implementation from Bechler-Speicher et al. (2025). We then stack multiple layers to form the processor component of our baseline architecture. Unless otherwise specified, we use mean pooling for graph-level tasks at the end of the processor step.

GNN+ Baseline methods In addition to the graph transformer and GINE baseline methods, we use current state-of-the-art GNN methods originally proposed by as an extension to existing GNN baselines. Following the approach from their paper, we include the GIN+, GatedGCN+, and GCN+ baselines throughout our experiments. Specifically, layer updates are given by the following, with node representations $h_v^{(l)}$ updated.
GIN+:

$$\boldsymbol{h}_v^l = \text{FFN}(\text{Dropout}(\sigma(\text{BN}(\text{MLP}^l((1+\epsilon)\cdot \boldsymbol{h}_v^{l-1} + \sum_{u\in\mathcal{N}(v)} \boldsymbol{h}_u^{l-1}) + \boldsymbol{e}_{uv}\boldsymbol{W}_e^l)))) + \boldsymbol{h}_v^{l-1})$$

GatedGCN+:

$$\boldsymbol{h}_v^l = \text{FFN}(\text{Dropout}(\sigma(\text{BN}(\sum_{u\in\mathcal{N}(v)\cup\{v\}} \frac{1}{\sqrt{\hat{d}_u\hat{d}_v}}\boldsymbol{h}_u^{l-1}\boldsymbol{W}^l + \boldsymbol{e}_{uv}\boldsymbol{W}_e^l)))) + \boldsymbol{h}_v^{l-1})$$

GCN+

$$\boldsymbol{h}_v^l = \text{FFN}(\text{Dropout}(\sigma(\text{BN}(\boldsymbol{h}_v^{l-1}\boldsymbol{W}_1^l + \sum_{u\in\mathcal{N}(v)} \eta_{v,u}\odot \boldsymbol{h}_u^{l-1}\boldsymbol{W}_2^l + \boldsymbol{e}_{uv}\boldsymbol{W}_e^l)))) + \boldsymbol{h}_v^{l-1})$$

where σ denotes an activation function, FFN and MLP denote a fully connected linear layer and an l layer MLP, respectively. Furthermore, $\eta = \sigma(\boldsymbol{h}_u^{l-1}\boldsymbol{W}_3 + \boldsymbol{h}_v^{l-1}\boldsymbol{W}_4)$, $\hat{d}_u$ denotes the degree of node u and $\boldsymbol{W}$ denote weight matrices. Edge features are included via e_{uv}

D.3. Architecture choices for datasets

Social Networks On the proposed BlueSky datasets, we opted for a different GNN architecture. The high(er) number of nodes, along with high average node degree and variance, pushed us towards resorting to a message-passing architecture aggregating messages over neighborhoods via averaging (instead of summation as in GIN). We opted, in particular, for a variant of GraphConv with mean aggregation. Node representations $h_v^{(t)}$ are, namely, updated as follows:

$$h_v^{(t+1)} = \sigma\Big(W_1^{(t)}h_v^{(t)} + \frac{1}{\deg_{in}(v)}\sum_{u:\ u\to v} W_2^{(t)}h_u^{(t+1)}\Big), \tag{3}$$

where σ is set to ReLU, dropout is applied before its application and, finally, no normalization layers are interleaved.

Weather forecasting

For the proposed weather forecasting dataset, we provide a hybrid GNN+Graph Transformer architecture closely based on Graphcast (Lam et al., 2023). Since the data contains both grid and mesh structures, our architecture contains the following modules. A GNN-based encoder, a Graph Transformer processor, and a GNN decoder, along with a prediction head. The encoder consists of a single GNN layer and maps the grid points onto the mesh. It receives in addition to the node features (the current and previous time step, with associated variables) the spherical coordinates of the grid node and corresponding mesh nodes. Furthermore, edge features are given by the relative 3d positions of mesh and grid points in the global mapping as well as an L2 normalized distance. The encoder then computes the mesh features.

Let $\mathcal{G} = (\mathcal{V}_g, \mathcal{V}_m, \mathcal{E}_{g2m}, \mathcal{E}_m, \mathcal{E}_{m2g})$ be the heterogeneous graph, where $\mathcal{V}_g$ are grid nodes (lat-lon), $\mathcal{V}_m$ are mesh nodes, $\mathcal{E}_{g2m}$ grid-to-mesh edges, $\mathcal{E}_m$ mesh-internal edges, and $\mathcal{E}_{m2g}$ mesh-to-grid edges. Each grid node $i \in \mathcal{V}_g$ carries state features $\mathbf{u}_i^{(t)} \in \mathbb{R}^F$ at timestep t and spatial coordinates (ϕ_i, θ_i) (longitude, latitude).

The raw two-timestep grid state is stacked and normalized channel-wise:

$$\tilde{\mathbf{u}}_i = \frac{\left[\mathbf{u}_i^{(t)},\, \mathbf{u}_i^{(t-1)}\right] - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \;\in\; \mathbb{R}^{2F},$$

where $\boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^{2F}$ are pre-computed training-set statistics (tiled for both timesteps), and NaN/Inf entries are replaced by 0.

Further we compute the spherical to cartesian coordinate transformation: For a node at latitude λ and longitude ℓ (in degrees):

$$\phi = \frac{\pi}{180}\ell, \qquad \theta = \frac{\pi}{180}(90 - \lambda), \qquad \mathbf{p} = \begin{pmatrix} \cos\phi\sin\theta \\ \sin\phi\sin\theta \\ \cos\theta \end{pmatrix} \in \mathbb{R}^3.$$

Afterward, we can compute edge features used in the encoder and decoder:

$$\mathbf{s}_i^{\text{grid}}, \mathbf{s}_i^{\text{mesh}} = \left[\cos\theta_i,\; \cos\phi_i,\; \sin\phi_i\right] \;\in\; \mathbb{R}^3.$$

For an edge e_{ij} the relative displacement and L2 distance in 3D are:

$$\Delta\mathbf{p}_{ij} = \mathbf{p}_j - \mathbf{p}_i, \qquad d_{ij} = \|\Delta\mathbf{p}_{ij}\|_2,$$

normalized by the maximum distance $d_{\max}$ of same type edges:

$$\mathbf{s}_{ij}^{\text{edge}} = \left[\frac{d_{ij}}{d_{\max}},\; \frac{\Delta\mathbf{p}_{ij}}{d_{\max}}\right] \;\in\; \mathbb{R}^4.$$

We now compute the initial grid and mesh features:

$$\mathbf{h}_i^g = W_g\left[\tilde{\mathbf{u}}_i \,\|\, \mathbf{s}_i^{\text{grid}}\right] \;\in\; \mathbb{R}^D, \qquad i \in \mathcal{V}_g,$$

$$\mathbf{h}_j^m = W_m\, \mathbf{s}_j^{\text{mesh}} \;\in\; \mathbb{R}^D, \qquad j \in \mathcal{V}_m,$$

where $W_g \in \mathbb{R}^{D\times(2F+3)}$ and $W_m \in \mathbb{R}^{D\times 3}$ are linear embeddings.

Each grid-to-mesh edge $(i,j) \in \mathcal{E}_{g2m}$ is embedded via a MLP layer and then processed using a neighborhood aggregating GNN:

$$\mathbf{e}_{ij}^{g2m} = W_{g2m}\, \mathbf{s}_{ij}^{\text{edge}} \;\in\; \mathbb{R}^D,$$

$$\tilde{\mathbf{e}}_{ij}^{g2m} = \text{MLP}_{g2m}\Big(\mathbf{e}_{ij}^{g2m} \,\|\, \mathbf{h}_i^g \,\|\, \mathbf{h}_j^m\Big) \;\in\; \mathbb{R}^D,$$

with residual: $\hat{\mathbf{e}}_{ij}^{g2m} = \mathbf{e}_{ij}^{g2m} + \tilde{\mathbf{e}}_{ij}^{g2m}$.

$$\mathbf{a}_j^m = \sum_{i:\,(i,j)\in\mathcal{E}_{g2m}} \tilde{\mathbf{e}}_{ij}^{g2m},$$

$$\tilde{\mathbf{h}}_j^m = \mathrm{MLP}_m\big(\mathbf{h}_j^m \parallel \mathbf{a}_j^m\big)\,,$$
$$\hat{\mathbf{h}}_j^m = \mathbf{h}_j^m + \tilde{\mathbf{h}}_j^m.$$

All encoder/decoder MLPs use the SiLU activation:

$$\mathrm{MLP}(\mathbf{x}) = W_2\,\mathrm{SiLU}(W_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2, \quad \mathrm{SiLU}(z) = z \cdot \sigma(z).$$

In a second step the processor then computes the node updates for mesh nodes only. For this the same graph transformer from Stoll et al. (2025) is used as in previous tasks.

Mesh-internal edge features are embedded and projected to per-head attention biases:

$$\mathbf{b}_{jk}^{\text{mesh}} = W_{\text{enc}}\Big(W_{\text{mesh}}\,\mathbf{s}_{jk}^{\text{mesh-edge}}\Big) \;\in\; \mathbb{R}^H, \quad (j,k) \in \mathcal{E}_m,$$

where H is the number of attention heads. These biases are added to the attention logits. The processor runs L standard Transformer encoder layers:

$$\mathbf{Z}^{(\ell)} = \mathrm{TransformerLayer}_\ell\Big(\mathbf{Z}^{(\ell-1)},\, A^{\text{mesh}}\Big)\,, \quad \ell = 1, \ldots, L,$$

where $A^{\text{mesh}} \in \mathbb{R}^{(N_m+1)\times(N_m+1)\times H}$ contains the edge-derived attention biases ($-\infty$ for non-edges). The output mesh embeddings are $\bar{\mathbf{h}}_j^m = \mathbf{Z}_{j+1}^{(L)}$.

As a final step, the decoder maps mesh features back to the underlying grid. Afterward, computed predictions are denormalized, and the residual between time steps t and $t+2$ is computed as the result using an MLP prediction head.

$$\tilde{\mathbf{e}}_{ji}^{m2g} = \mathrm{MLP}_{m2g}\Big(\mathbf{e}_{ji}^{m2g} \parallel \bar{\mathbf{h}}_j^m \parallel \mathbf{h}_i^g\Big) \;\in\; \mathbb{R}^D,$$

where $\mathbf{e}_{ji}^{m2g} = W_{m2g}\,\mathbf{s}_{ji}^{\text{edge}}$.

$$\mathbf{a}_i^g = \sum_{j\colon (j,i)\in\mathcal{E}_{m2g}} \tilde{\mathbf{e}}_{ji}^{m2g}.$$

$$\hat{\boldsymbol{\delta}}_i = \mathrm{MLP}_{\text{out}}(\mathbf{h}_i^g \parallel \mathbf{a}_i^g) \;\in\; \mathbb{R}^F.$$

$$\boldsymbol{\delta}_i = \hat{\boldsymbol{\delta}}_i \odot \boldsymbol{\sigma}_{\Delta},$$
$$\hat{\mathbf{u}}_i^{(t+2)} = \mathbf{u}_i^{(t)} + \boldsymbol{\delta}_i,$$

where $\boldsymbol{\sigma}_{\Delta} \in \mathbb{R}^F$ is the pre-computed standard deviation of the per-variable one-step differences (training set) and $\odot$ denotes the elementwise product.

D.4. Dataset preprocessing

To align the proposed datasets with baseline methods, we applied the following preprocessing steps to the original datasets.

Algorithmic reasoning For edge-level tasks requiring edge classification on a graph, we transform the problem statement as follows to transform the task into a node-level task: We leverage the following edge transform which allows us to provide node level tokenization/targets on a modified graph G' obtained from the original graph G, with $V(G') := \{(v,v) \mid v \in V(G)\} \cup E(G)$ and $E(G') := \{((u,v),(w,z)) \mid u = w \vee u = z \vee v = w \vee v = z\}$. We use this transform for all baselines evaluated for algorithmic reasoning datasets.

Furthermore, since networkX outputs only directed edges for each of the edge level tasks we further transform the target tensor y as follows: We provide the same label for the reverse edge to each outgoing edge. This is done by comparing existing edges and the target tensor. Therefore, we transform the directed target into an undirected one.

Weather forecasting

Since features are provided as unnormalized values, we follow the preprocessing pipeline from Lam et al. (2023) and Price et al. (2023). We outline the process of preprocessing for weather forecasting information in the following:

First, the raw ERA5 data, provided as a .zarr file, is loaded, and the empty mesh is created once. Mesh creation depends on the number of refinement levels selected; for our case, we set it to four. Furthermore, grid points are connected to corresponding nearby mesh points, thereby creating node and edge indices for the graphs outlined in subsection D.3. Since these are the same across all time steps, they are saved as static components to reduce storage requirements.

Afterward, the node features are attached to the grid nodes, and variable count information is saved for later use in the loss function. This constitutes the raw data included in our dataset.

In order to reduce local preprocessing time, we also provide the dataset used in our evaluation. For this, the following additional steps have been applied.

Each timestep receives the previous timestep as additional node features, including all weather variables but not location information. Furthermore, static information is saved separately, and mesh node features are initialized with zero vectors.

At training time, the features are additionally normalized and enhanced with distance information as outlined in subsection D.3. All additional information is saved as metadata in the dataset or provided as additional files.

Detailed descriptions of each feature and the preprocessing pipeline can be found in (Lam et al., 2023).

E. Automated hyperparameter optimization

GRAPHBENCH also integrates automated hyperparameter optimization (HPO), which aims to make it easier to tune new models, improve performance, and enhance reproducibility. Deep graph learning architectures are notoriously sensitive to their hyperparameter configurations, ranging from learning rates and weight decay to structural choices such as the number of layers and hidden dimensions. To address this, the hyperparameter optimization in GRAPHBENCH is based on the SMAC3 (Lindauer et al., 2022) package, a versatile framework that leverages Bayesian Optimization (BO) to efficiently explore the configuration space.

At its core, standard Bayesian Optimization constructs a probabilistic surrogate model that maps a hyperparameter configuration to its observed performance. The BO loop proceeds iteratively: first, an initial set of configurations is evaluated. Then, the surrogate model is fitted to this historical data. To decide which configuration to evaluate next, BO employs an acquisition function, such as Expected Improvement (EI) (Močkus, 1989). The acquisition function evaluates the utility of candidate configurations by balancing *exploitation* (proposing configurations in regions where the surrogate model predicts high performance) and *exploration* (proposing configurations in regions where the surrogate model is highly uncertain). Once the acquisition function is maximized, the best candidate configuration is evaluated on the target task, and its result is used to update the surrogate model. In SMAC3, the default surrogate model is a random forest (RF). Random Forests are particularly well-suited for HPO because they natively handle mixed continuous, categorical, and conditional hyperparameter spaces, and they can estimate predictive variance (uncertainty) empirically from the variance across the individual regression trees.

While standard BO is vastly more efficient than random search, evaluating every proposed configuration to completion (e.g., training a GNN for 100 000 steps) remains too costly. To mitigate this, we specifically utilize the *multi-fidelity facade* provided by SMAC3. Multi-fidelity optimization accelerates the search by evaluating configurations on cheaper, approximate versions of the target task, known as *fidelities*. Fidelities can be defined by the number of training steps, the number of epochs, or the size of the dataset subset. A well-known algorithm for multi-fidelity optimization is Hyperband (Li et al., 2017). Hyperband samples configurations randomly, evaluates them on a low fidelity, and prunes unpromising configurations using a tournament-like strategy. Hyperbands repeats this process multiple times, with increasing minimum fidelities, to give slow-learning configurations a chance to show their performance.

The underlying scheduling algorithm used by the SMAC3 multi-fidelity facade is based on BOHB (Falkner et al., 2018), which combines the strengths of Hyperband for early-stage elimination of unpromising configurations with those of BO for late-stage convergence toward the global optimum. In short, BOHB samples new configurations using BO and prunes unpromising configurations using a Hyperband-style strategy.

Due to limited computational resources, we were unable to run HPO on all datasets and domains. However, we conducted HPO for all SAT datasets and for all baselines. For this purpose, we used the SMAC3 multi-fidelity facade with 60 trials per run. The chosen fidelity metric was the number of training steps: we defined the minimum budget as 1 000 steps and the maximum budget as 100 000 steps. This multi-fidelity approach allowed us to explore a vast configuration space—evaluating many architectures and hyperparameter combinations at 1 000 steps—while only training the very best candidates to the full 100 000 steps. The complete configuration space used for this process is shown in Table 50.

Table 50. Configuration space used in the hyperparameter optimization.

Hyperparameter	**Range**
Learning rate	[1e-5, 5e-2]
Weight decay	[1e-8, 1e-1]
Warmup iters	[1 000, 20 000]
Dropout	[0.0, 0.5]
Num layers	[4, 10]
Hidden dim	{256, 384, 512}