NEURAL TRACTABILITY VIA STRUCTURE: LEARNING-AUGMENTED ALGORITHMS FOR GRAPH COMBINATORIAL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural solvers provide fast solutions to graph combinatorial optimization problems, but training alone does not guarantee solution quality. Exact methods guarantee optimality but can be prohibitively expensive. We propose Neural Fixed-Parameter Tractable (N-FPT), a *neural-model-agnostic* framework that uses parameterized algorithms to improve neural solutions without retraining and guide learning. It restricts neural advice to a treewidth modulator and completes the bounded-treewidth remainder exactly. It returns and certifies the *best achievable solution* under any valid advice, improving the neural solution whenever a better compatible completion exists. We prove this guarantee and global optimality under perfect advice. Its incremental-confidence N-FPT (NIC-FPT) and randomized-deferral N-FPT (NRD-FPT) variants consolidate neural advice. Beyond inference, N-FPT *feedback* becomes available as neural decisions make exact completion tractable. From that state onward, it supplies what final solution rewards lack: the *first decision that rules out every optimal completion* and alternatives that preserve one. One exact computation supplies reusable guidance for continuations from that state, even when neural samples miss its optimum. Experiments across graph combinatorial optimization problems demonstrate solution-quality gains with different neural models and robustness to graph-size and distribution shifts. The learned completion model reaches optimal completions more often than reinforcement learning (RL) alone, without an exact solver at inference. N-FPT thus uses graph structure to improve neural solutions and guide learning toward optimal completion.

1 INTRODUCTION

Graph combinatorial optimization seeks the best discrete choices under graph constraints: for example, selecting as many mutually nonadjacent vertices as possible or partitioning vertices to maximize the number of crossing edges. Many such problems are NP-hard. Exact methods guarantee optimality but can be prohibitively expensive, while neural solvers offer fast predictions after training without comparable guarantees (Zhang et al., 2023). Parameterized algorithms offer a way to use *graph structure* to make exact computation tractable (Cygan et al., 2015; Fomin et al., 2019).

We focus on dynamic programming (DP) parameterized by treewidth, which measures how closely a graph's structure resembles a tree. A tree decomposition covers the graph with overlapping vertex sets, called *bags*, arranged in a tree (Figure 1). Every edge lies in a bag, and the bags containing any given vertex form a connected subtree (Robertson & Seymour, 1984). The decomposition's width is the *largest bag size minus one*; treewidth is the minimum width over all decompositions. The benefit of this DP is broad: every fixed graph property expressible in monadic second-order logic can be decided in linear time on graphs of bounded treewidth (Courcelle, 1990). In the dynamic programs used here, the exponential dependence is on width, not graph size (Lokshtanov et al., 2011), so large graphs can be tractable when their width is small. Large width, however, still makes this computation expensive: the number of possible local assignments grows exponentially with bag size.

We propose N-FPT to combine fast neural predictions with exact completion, without retraining. On a fixed tree decomposition, it identifies a treewidth modulator: a vertex set whose removal from the bags reduces the width to a chosen threshold. A complete valid neural solution supplies its decisions

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

on this set as *neural advice*; DP optimizes the remaining decisions under the original constraints and objective. We build on modulator-based enhancement of graph heuristics (Li et al., 2026). While that framework repairs conflicting advice when needed, our sequential neural samplers preserve feasibility at each decision, supplying valid advice directly. We extend exact completion to prediction consolidation and neural learning.

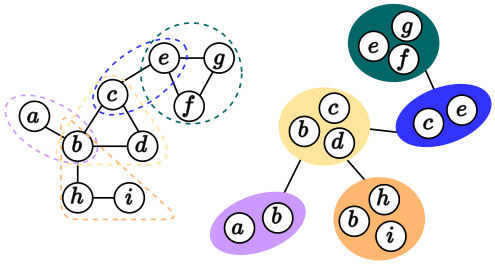


Figure 1: A graph and a width-two tree decomposition. Fixing the decisions on $\{b, e\}$ leaves at most two undecided vertices per bag, reducing the residual width to one. The disjoint bags $\{b, c, d\}$ and $\{e, f, g\}$ each require a fixed vertex, so two are necessary.

The neural solver therefore need not predict an entire optimal solution. N-FPT returns and certifies the *best achievable solution* under its advice (Proposition 3.1). It improves the neural solution whenever a better compatible completion exists; otherwise, it establishes that the solution is already optimal under the fixed advice. Under *perfect advice*—modulator decisions consistent with a global optimum—the result is globally optimal. The framework is *neural-model agnostic*: the same procedure applies to any neural solver supplying a valid solution.

However, perfect advice cannot be assumed in practice. We therefore make improving advice quality part of the learning-augmented design, rather than treating predictions as a fixed assumption. We develop NIC-FPT and NRD-FPT as two compact N-FPT variants with *consolidated prediction*. NIC-FPT commits selections supported by agreement across neural samples and conditions subsequent samples on *this consensus*. NRD-FPT randomly defers selections and resamples while retaining the others. Each variant returns the best achievable solution under its consolidated modulator advice, without retraining the neural model.

Beyond inference, N-FPT offers a different learning paradigm: learning to preserve optimal completion through exact corrections to sampled decisions. A common approach is reinforcement learning, with rewards derived from solution objective values (Bello et al., 2016; Khalil et al., 2017). This encourages high rewards on average, but a final score alone does not reveal where optimal completion was lost. As a neural completion model (NC) extends a partial solution, N-FPT supplies feedback once the remainder becomes tractable; the original starting state need not admit an affordable exact solve. From that point, its dynamic program represents *all optimal completions through shared local states*, without listing complete solutions. NC continues making its own selections, while N-FPT *feedback* identifies the *first subsequent decision that rules out every optimal completion* and teaches alternatives that preserve one. The stored states are reused across continuations from the same partial solution. Neural decisions thus enable exact computation, and exact feedback teaches NC to preserve the best value still achievable, even when its samples miss that value.

Experiments across neural models and optimization problems demonstrate solution-quality gains and robustness to graph-size and distribution shifts. With matched architectures and starting states, feedback-trained NC reaches optimal completions *more often* than RL alone without an exact solver at inference, so *these gains reflect learned completion*.

Our contributions are:

- *Best achievable solutions without neural retraining.* N-FPT accepts valid solutions from any neural model and certifies the best achievable completion of their modulator advice, with global optimality under perfect advice.

- *Consolidated prediction within* N-FPT. NIC-FPT and NRD-FPT consolidate neural predictions through sample agreement and randomized deferral, respectively. Each returns the best achievable completion of the resulting advice without updating the neural model.
- *Learning optimal completion from suboptimal samples.* As NC makes exact completion tractable, reusable N-FPT feedback identifies the first subsequent decision that loses optimal completion and teaches alternatives that preserve it, giving NC supervision beyond final rewards.

2 Problem Setting

We study maximum independent set (MIS), minimum vertex cover (MVC), and MaxCut on $G = (V, E)$, with $n = |V|$ and $m = |E|$. Let $\mathcal{F}(G) \subseteq 2^V$ be the valid solutions and $f_G(S)$ their objective, maximized for MIS and MaxCut and minimized for MVC. For a global optimum $S^\star$, write $\mathrm{OPT}(G) = f_G(S^\star)$. Neural advice is a partial assignment $a : D \to \{0, 1\}$ on $D \subseteq V$, with 1 meaning selected and 0 excluded. Its valid completions form $\mathcal{F}(G \mid a)$; we call a valid when this family is nonempty and write $V_G(a)$ for its *best achievable value*. An algorithm is *fixed-parameter tractable* (FPT) in k if it runs in $f(k) \cdot n^{\mathcal{O}(1)}$ time, with computable f and polynomial exponent independent of k (Cygan et al., 2015).

3 N-FPT: Exact Completion and Consolidated Prediction

Making Exact Completion Tractable. We construct a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ using the min-degree heuristic. Here X_t is the bag at node t of the tree T. For a target width $\eta \geq 0$, we compute a minimum modulator on this fixed decomposition (Li et al., 2026):

$$\mathcal{M}_\eta \in \arg\min_{M \subseteq V} |M| \quad \text{subject to} \quad |X_t \setminus M| \leq \eta + 1 \quad \text{for every } t \in V(T). \tag{1}$$

We solve this mixed-integer linear program exactly. The modulator is minimum for the given decomposition, not over all decompositions of the graph. Removing it from every bag gives a valid decomposition of $G \setminus \mathcal{M}_\eta$ with width at most η (Figure 1).

Given a complete valid solution $S_\theta \in \mathcal{F}(G)$ from any neural solver, we retain the advice $a_\theta(v) = \mathbf{1}\{v \in S_\theta\}$ for $v \in \mathcal{M}_\eta$ and optimize all other decisions. The neural model supplies only S_θ; neither the modulator nor the exact computation depends on its architecture or training objective. The modulator vertices are fixed, not discarded: their constraints and objective contributions remain part of the original problem. For MIS, for example, a selected modulator vertex excludes its neighbors. Advice from S_θ always admits a valid completion.

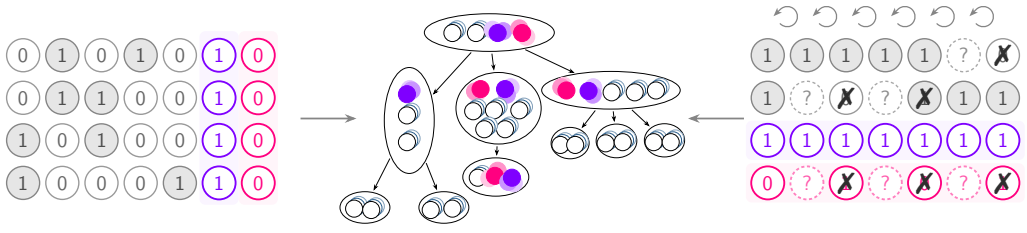


Figure 2: N-FPT (center): neural advice fixes colored vertices, and DP optimizes the remaining decisions on white vertices. NIC-FPT (left) and NRD-FPT (right) consolidate the advice; crosses indicate random deferral.

The dynamic program retains the best compatible partial solution for each of at most $2^{\eta+1}$ assignments to the undecided vertices in a bag. Tracing back an optimal root entry returns a best achievable completion (Figure 2, center). Given the decomposition and boundary contributions, completion is linear in the number of bags and exponential in η. This FPT bound excludes modulator optimization. Smaller η reduces the exact computation but can require more neural decisions; larger η permits more decisions to be optimized exactly. The relevant trade-off is thus residual width versus dependence on neural advice, not graph size or density alone.

Proposition 3.1 (Exact completion, consistency, and non-degradation)**.** *Fix a graph G, a modulator $\mathcal{M}_\eta$, and an exact problem-specific treewidth dynamic program that enforces the advice and all boundary constraints and objective contributions. Then:*

- Exact completion. *For any valid advice $a : \mathcal{M}_\eta \to \{0, 1\}$,* N-FPT *attains $V_G(a)$.*
- Consistency. *If the advice is perfect, meaning that it is consistent with some globally optimal solution, the returned solution is globally optimal.*
- Non-degradation. *For any complete valid neural solution S_θ and its restriction a_θ to $\mathcal{M}_\eta$, $V_G(a_\theta) \geq f_G(S_\theta)$ for maximization; the inequality is reversed for minimization.*

The proof is in Appendix A.1. For maximization, the neural solution's gap separates as

$$\mathrm{OPT}(G) - f_G(S_\theta) = \underbrace{\mathrm{OPT}(G) - V_G(a_\theta)}_{\text{advice regret}} + \underbrace{V_G(a_\theta) - f_G(S_\theta)}_{\text{loss in neural completion}}. \tag{2}$$

For minimization, each difference is reversed. Exact completion removes the second term; reducing the remaining advice regret requires better fixed decisions. This motivates NIC-FPT and NRD-FPT.

Consolidated Prediction. NIC-FPT and NRD-FPT consolidate neural advice within N-FPT to reduce advice regret without retraining (Figure 2). Both require a neural sampler that generates valid completions conditioned on fixed decisions. For MIS, NIC-FPT draws K valid completions per round, conditioned on the consensus from earlier rounds. It fixes an undecided vertex as selected when more than κ samples select it, with $\kappa > K/2$, and excludes its neighbors. NRD-FPT instead starts from a complete valid neural sample and independently defers each selected vertex with probability ρ at each round. Retained selections and their implied neighbor exclusions stay fixed while the model resamples a valid completion over all other vertices. Other problems use their own validity rules.

After the prescribed rounds, the final valid neural sample supplies modulator advice a. Each variant returns its best achievable completion $V_G(a)$, with the guarantees of Proposition 3.1.

Table 1: Results across neural models on MIS (Reg graphs). Gaps use the starred Gurobi reference; improvements are compared within each neural model.

Method	SS		SD		LS		LD	
	Size ↑	Gap% ↓	Size ↑	Gap% ↓	Size ↑	Gap% ↓	Size ↑	Gap% ↓
Gurobi(60s)	280.29	3.31	170.84	2.39	358.03	4.69	216.68	4.09
*Gurobi(600s)	289.90	0.00	175.03	0.00	375.64	0.00	225.93	0.00
KaMIS(600s)	299.97	-3.47	183.96	-5.10	387.11	-3.05	238.28	-5.47
GFlowNet, average								
Standalone	286.46	1.19	170.40	2.65	372.55	0.82	221.34	2.03
N-FPT	286.55	1.16	170.41	2.64	372.65	0.80	**221.35**	**2.03**
NIC-FPT	**286.86**	**1.05**	**171.71**	**1.90**	**372.88**	**0.73**	221.34	2.03
GFlowNet, best-of-20								
Standalone	290.55	-0.22	174.67	0.21	377.22	-0.42	226.23	-0.13
N-FPT	290.63	-0.25	174.68	0.20	377.35	-0.46	226.25	-0.14
NRD-FPT	**293.90**	**-1.38**	**176.96**	**-1.10**	**381.78**	**-1.63**	**228.68**	**-1.22**
GNN+RL, average								
Standalone	290.63	-0.25	174.32	0.41	374.06	0.42	225.46	0.21
N-FPT	290.66	-0.26	174.32	0.41	374.09	0.41	225.46	0.21
NIC-FPT	**290.84**	**-0.32**	**174.33**	**0.40**	**374.20**	**0.38**	**225.47**	**0.20**
GNN+RL, best-of-20								
Standalone	294.48	-1.58	177.92	-1.65	378.51	-0.76	230.02	-1.81
N-FPT	294.51	-1.59	177.92	-1.65	378.54	-0.77	230.02	-1.81
NRD-FPT	**296.70**	**-2.35**	**180.25**	**-2.98**	**381.88**	**-1.66**	**233.07**	**-3.16**

Experimental Overview. We test whether N-FPT strengthens neural solutions across *neural models*, *optimization problems*, and *test distributions*. Each trained solver is compared with its N-FPT, NIC-FPT, and NRD-FPT versions on the same test instances, without updating its weights. The objective is improvement to each existing neural solver; Gurobi and KaMIS provide reference values. Tables 1 and 2 report random regular (Reg) graphs in four settings combining small/large graphs (S/L) with sparse/dense settings (S/D). Full protocols and additional graph families are provided in the appendix.

Table 2: Results across optimization problems with GFlowNet on Reg graphs. Mean objective values.

METHOD	MaxCut ↑ SS	SD	LS	LD	METHOD	MVC ↓ SS	SD	LS	LD
Average					*Average*				
STANDALONE	1944.50	4458.06	2582.00	5767.26	STANDALONE	561.59	680.14	729.77	881.69
N-FPT (ours)	1967.72	4474.58	2587.64	5792.78	N-FPT (ours)	561.50	680.13	729.67	**881.68**
NIC-FPT (ours)	**1996.30**	**4490.85**	**2608.50**	**5824.44**	NIC-FPT (ours)	**561.19**	**680.07**	**729.44**	881.69
Best-of-20					*Best-of-20*				
STANDALONE	1963.73	4492.65	2601.64	5804.41	STANDALONE	557.50	675.87	725.10	876.80
N-FPT (ours)	1986.51	4509.91	2606.80	5830.97	N-FPT (ours)	557.42	675.86	724.97	876.78
NRD-FPT (ours)	**2007.57**	**4544.66**	**2618.95**	**5894.62**	NRD-FPT (ours)	**554.15**	**673.58**	**720.54**	**874.35**

Results across Neural Models. We first compare two established learning approaches on MIS (Table 1). GFlowNet learns a reward-weighted distribution over solutions (Zhang et al., 2023). GNN+RL represents the widely used combination of graph neural networks and reinforcement learning (Khalil et al., 2017; Barrett et al., 2020; Ahn et al., 2020); our policy maximizes expected solution reward. *Exact completion complements both learning objectives*: the same parameterized procedure recovers improvements left unused by either trained solver.

A fractional increase in mean MIS size represents whole-vertex improvements in individual solutions, averaged over graphs, samples, and training seeds. N-FPT reaches the best value permitted by each solution's advice, regardless of how well the neural model is trained. If exact completion leaves an individual solution's objective unchanged, it *certifies that solution as best achievable under its fixed advice*. Once this value is reached, further gains require better advice. The additional gains from NIC-FPT and NRD-FPT demonstrate the value of consolidated prediction.

Results across Optimization Problems. We next apply the framework to MaxCut and MVC using GFlowNet (Table 2). MaxCut tests an edge-based maximization objective, while MVC tests vertex-based minimization. N-FPT improves both objectives, while consolidated prediction enables further gains; NRD-FPT improves best-of-20 quality in every reported setting for both problems. Together with the cross-model results, these gains demonstrate applicability across *neural models and optimization problems*.

Generalization across Graph Sizes and Distributions. We evaluate GFlowNet trained on small sparse (SS) graphs for MIS across graph sizes, densities, and families, without retraining (Figure 3). Zero marks the performance of a standalone GFlowNet trained specifically for the target configuration; positive values exceed this reference, which is not a certified optimum.

Changing density can be more disruptive than increasing graph size alone. Across graph families, the same trained solver can also lose quality. N-FPT improves these transferred solutions, while NIC-FPT and NRD-FPT reduce the remaining losses and sometimes surpass the target-trained reference. Thus, *training on the test distribution is not required to benefit from the framework*.

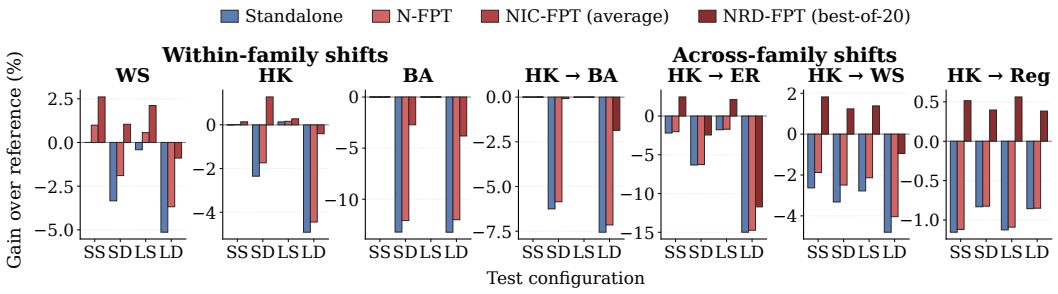


Figure 3: Gains in average solution size (left) and best-of-20 solution size (right). All methods and the reference use the same measure within each block.

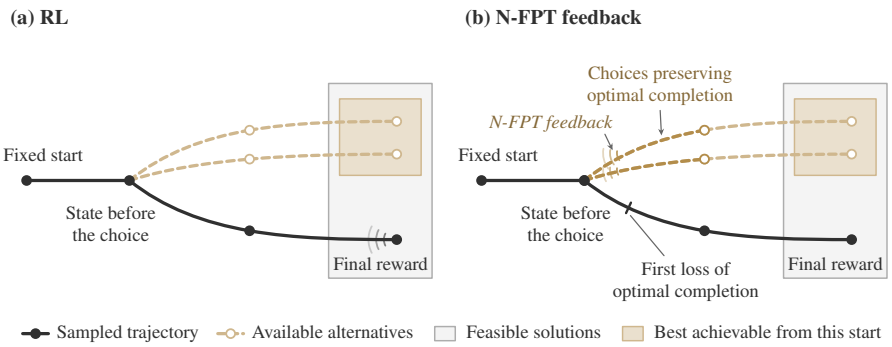


Figure 4: Learning signals from a partial state where exact completion is tractable. Dashed paths are available alternatives, not additional samples; gold waves mark N-FPT feedback.

4 Learning Optimal Completion with N-FPT Feedback

Training a neural solver for high solution rewards does not guarantee an optimal solution. In our RL baseline, a final reward neither locates the decision that lost optimal completion nor identifies alternatives that would have preserved it (Figure 4(a)). Obtaining optimal training targets for NP-hard problems can itself exceed practical budgets on large, difficult instances: an exact solver must both find an optimum and prove that no better solution exists. With no initial assignment, the target is *global optimality*; once fixed decisions rule out every global optimum, it is the *optimal completion conditioned on those decisions*. The missing supervision is **which decisions preserve the best achievable completion**, not just final solution quality.

N-FPT *feedback* supplies this supervision by retaining the dynamic program's optimal alternatives, not just one completed solution. As NC proceeds along a trajectory, feedback becomes available when the remaining problem becomes tractable. NC continues making selections, while these alternatives identify the first subsequent selection that loses the best achievable value and the choices that preserve it (Figure 4(b)). Feedback at the state before that selection teaches NC to preserve optimal completion on future trajectories; sampled choices remain unchanged. Neural progress enables exact feedback without requiring the original starting state to be exactly solvable within the computation budget.

Learning to Preserve Optimal Completion. We describe the construction for MIS. Let a_0 be a valid, nonterminal starting assignment. NC's policy $\pi_\theta(v \mid G, a)$ observes the full graph and all decisions made so far, including a_0. Each vertex is selected, excluded, or undecided. A move selects an undecided vertex and excludes its neighbors, producing the next assignment; selections continue until no vertex is undecided. Fixed decisions are never changed. Write a_t for the assignment after t moves. Let $s \geq 0$ denote the step at which a nonterminal remainder becomes tractable. At a_s, N-FPT computes $V_G(a_s)$ and activates feedback for the continuation. All decisions through a_s stay fixed; feedback concerns the best value achievable from that point. The optimal completions still available are

$$\mathcal{O}_t = \{S \in \mathcal{F}(G \mid a_t) : f_G(S) = V_G(a_s)\}, \qquad t \geq s. \tag{3}$$

Because each move fixes further decisions, these families are nested: $\mathcal{O}_t \subseteq \mathcal{O}_{t-1}$ for $t > s$. A continuation that ends below $V_G(a_s)$ must therefore have a *first harmful selection* after feedback becomes available, at $h = \min\{t > s : \mathcal{O}_t = \varnothing\}$. Before this selection, the best achievable value is still $V_G(a_s)$; after it, that value cannot be recovered without changing a fixed decision. Feedback at a_{h-1} identifies all next selections that preserve at least one member of $\mathcal{O}_{h-1}$. It accepts every such choice, without prescribing one optimal solution.

Reusable Feedback from the Dynamic Program. A single optimal solution from an exact solver such as Gurobi (Gurobi Optimization, LLC, 2024) is not enough to label every neural decision: a different choice may still belong to another optimum. Exhaustive enumeration can recover all optimal alternatives, but storing them individually can require exponential space even at treewidth one. For example, m disjoint edges admit 2^m maximum independent sets, while their dynamic program needs only two optimal local choices per edge. We exploit this sharing to retain *all optimal completions*

without listing them. For fixed residual width, the shared representation uses space polynomial in the decomposition size, even when the number of complete optima is exponential.

At a_s, we extend the exact completion procedure of N-FPT to retain optimal alternatives rather than a single traceback. Starting from all optimal root entries, we keep every child entry that agrees with its parent on shared vertices and attains the best subtree value under that shared assignment. The retained bag states and their compatibility links represent all of $\mathcal{O}_s$ without listing complete solutions. We build this representation once for a_s and reuse it whenever NC continues from that same partial assignment.

As NC continues beyond a_s, we follow its selections through the stored representation. Selecting a vertex and excluding its neighbors deactivates inconsistent bag states. A parent state is also deactivated if it loses all compatible states in any child bag, propagating the change toward the root. An optimal completion survives exactly when at least one root state remains active. The first selection that removes the last active root state is therefore the first harmful selection after a_s. We undo that update only in the stored representation and follow compatible active entries from the surviving roots: every undecided vertex occurring in these entries belongs to a remaining optimal completion and is an acceptable next selection. Root connectivity ensures that the feedback refers to a complete optimal solution, not merely a locally compatible bag assignment. Each continuation from a_s resets only the activity flags and support counts. Once the representation is available, tracking its selections requires *no further exact solves*.

Training RL with N-FPT Feedback. We use this feedback to train NC with a single policy objective that rewards good solutions while teaching decisions that preserve optimal completion. From the same starting assignment a_0, NC samples K complete trajectories from its own policy. Each trajectory is a sequence of valid selections that finishes the solution, without being forced to follow an optimal completion. As a trajectory reaches a tractable state a_s, exact feedback becomes available for its subsequent decisions; different trajectories can reach different such states. Our experiments use $K = 4$.

For each state a_s where feedback is available, a feedback example (a, B) consists of a partial assignment a at or after a_s and a set B of next selections known to preserve $V_G(a_s)$. We obtain these pairs along a randomly ordered exact completion of a_s and immediately before the first harmful selection in NC's continuation. Along the exact completion, B contains its remaining selected vertices; at a sampled mistake, B contains all next selections supported by the surviving optimal completions. Both use the same feedback rule: increase the policy's total probability on B. No particular order is imposed on these choices, and the sampled trajectories remain unchanged.

For trajectory i, let L_i be the number of additional selections made after a_0; decisions already fixed in a_0 do not count toward L_i. At step t, NC selects vertex $v_{i,t}$ from the current assignment $a_{i,t-1}$, producing $a_{i,t}$. All trajectories begin at $a_{i,0} = a_0$. The final solution S_i receives reward $r_i = f_G(S_i)$. Its advantage A_i is r_i minus the mean reward of the other $K - 1$ trajectories from the same start. This comparison requires at least two trajectories ($K \geq 2$).

Let q_{fb} sample feedback pairs uniformly within each of the two sources above, giving each source equal total weight when both are available. We minimize the combined loss

$$\mathcal{L}_{\text{NC}}(\theta) = -\frac{1}{K}\sum_{i=1}^{K} A_i \sum_{t=1}^{L_i} \log \pi_\theta(v_{i,t} \mid G, a_{i,t-1}) - \lambda\, \mathbb{E}_{(a,B)\sim q_{\text{fb}}} \log\left(\sum_{v\in B} \pi_\theta(v \mid G, a)\right), \quad (4)$$

where $\lambda \geq 0$ controls the feedback weight. Recorded trajectories, advantages, and feedback pairs are held fixed during the update. The feedback term is omitted when no feedback examples are available. The policy is normalized over all undecided vertices, so feedback changes what NC learns without restricting which valid trajectories it can sample. Even when none of those trajectories is optimal, the exact computation supplies decisions that would preserve the best achievable value.

The trained policy can complete a new partial solution on its own or supply further decisions before N-FPT finishes a tractable remainder. The feedback is exact; its effect on learned completion is evaluated below.

Experimental Overview. We compare *RL* and *RL with* N-FPT *feedback* on MIS, using matched architectures, training graphs, starting assignments, and optimizer-update counts. Tests use random geometric graphs (RGG) and king's graphs with random vertex deletions (King), with zero, two, or

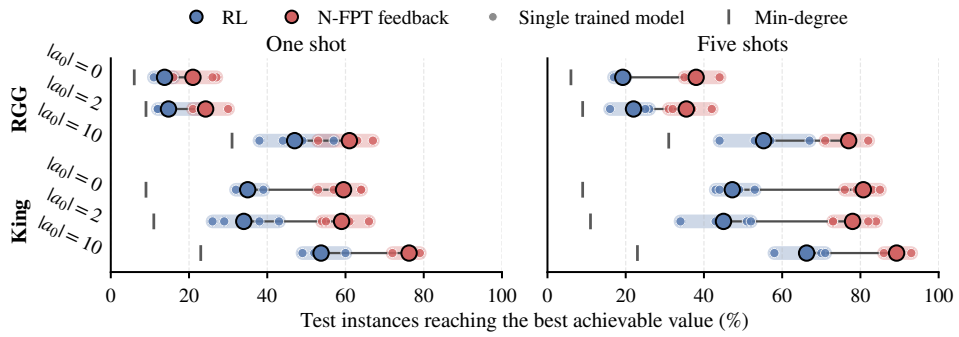


Figure 5: Optimal-completion frequency. Dots show four-seed means (large) and individual models (small); pale bars span seed ranges and gray ticks show min-degree. $|a_0|$ counts initial selections.

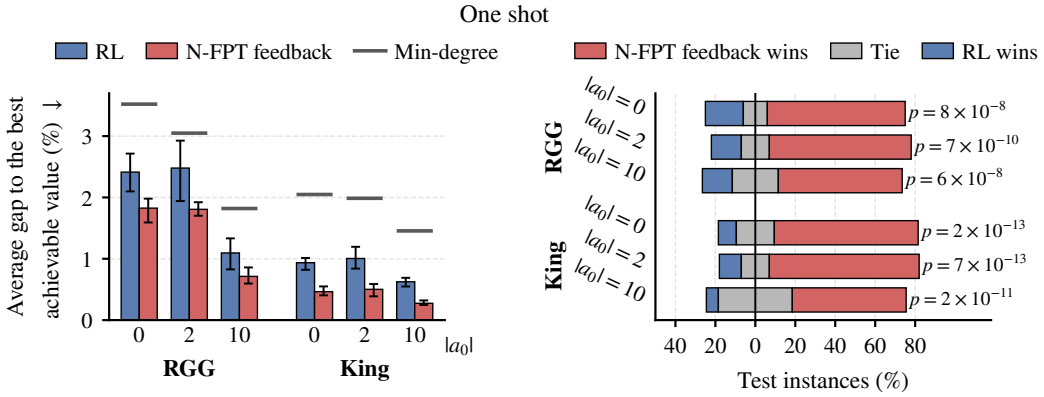


Figure 6: One-shot completion quality. Left: four-seed mean gaps with min–max whiskers; gray dashes show min-degree. Right: wins, ties, and losses based on four-seed mean solution sizes, with ties centered on zero. The p-values test equal win probabilities, excluding ties.

ten initial selections. NC completes each test solution without an exact solver and is assessed against its starting assignment's proven optimum. *One shot* selects the highest-probability vertex at each step; *five shots* returns the best of five sampled completions. These runs use $s = 0$ wherever exact feedback is available; the appendix gives the full protocol and additional five-shot results.

Reaching Optimal Completions More Often. Figure 5 shows a higher mean optimal-completion rate with N-FPT feedback in all twelve comparisons, with gains of up to 33.5 percentage points. The gains already appear in one-shot decoding, so they do not depend on additional test-time sampling. Under five-shot decoding, even the weakest of the four feedback-trained models outperforms the strongest RL baseline model in every setting. The improvement therefore extends beyond a favorable training seed.

With $|a_0| = 0$, reaching the best achievable value means finding a *global optimum*; for nonempty starts, it means finding the optimum compatible with the fixed decisions. Across both graph families, feedback-trained NC reaches this target more often and has smaller average gaps from ten-selection starts than from empty or two-selection starts (Figures 5 and 6). These larger starts leave fewer decisions for NC, while feedback improves over RL at every starting size. Its gains therefore cover both solving from scratch and completing partial solutions without revising their fixed decisions.

Solution Quality across Test Instances. Feedback lowers the average gap left by an RL baseline that already outperforms min-degree (Figure 6, left). The paired comparisons (right) show that it improves many more test instances than it worsens. The benefit is therefore not confined to a few favorable completions: NC produces better solutions across the test set, as well as reaching the best achievable value more often.

Preserving Optimal Completion along Trajectories. On one-shot test trajectories, N-FPT feedback keeps an optimal completion possible on more graphs as the policy makes its selections (Figure 7). The endpoints reproduce the one-shot optimal-completion rates in Figure 5. This connects the solution-quality gains to *preserving optimal completion during decoding*, the behavior that the feedback explicitly teaches.

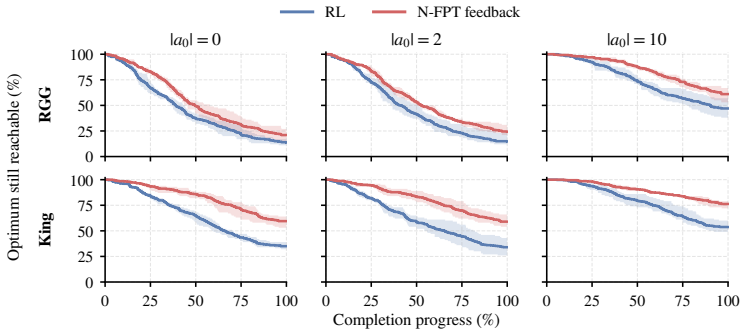


Figure 7: Optimal completion along one-shot test trajectories. Progress is the fraction of each trajectory's additional selections. Lines show four-seed means; bands span seed ranges.

Figure 8 shows that *suboptimal samples can teach optimal completion*. Early in training, nearly all samples miss the best achievable value, yet N-FPT provides exact corrections. Across both graph families and all starting sizes, fewer trajectories lose optimal completion over training; among remaining failures, the first harmful selection generally occurs later. Read together, these trends support the intended learning behavior: NC improves both its chance of reaching the optimum and its ability to preserve that possibility through successive decisions.

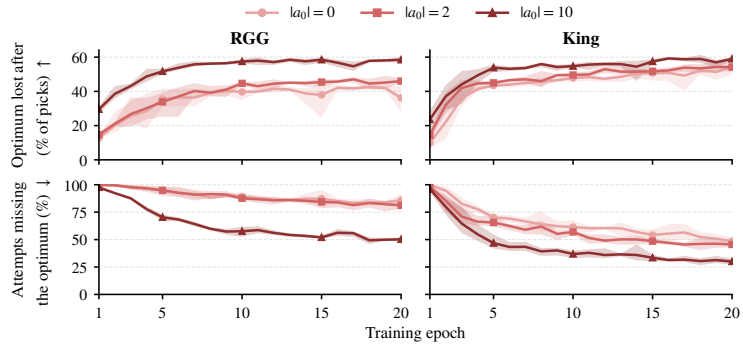


Figure 8: Training with N-FPT feedback. Top: the first harmful selection as a percentage of each suboptimal trajectory's additional selections, averaged over those trajectories. Lines show four-seed means; bands span seed ranges.

5 Conclusion

N-FPT uses parameterized algorithms to improve neural solutions and guide learning for graph combinatorial optimization. The neural-model-agnostic framework returns and certifies the *best achievable solution* under valid neural advice without retraining, and recovers a global optimum under perfect advice. Its NIC-FPT and NRD-FPT variants improve advice through consolidated prediction. As NC makes exact completion tractable, N-FPT feedback teaches which subsequent decisions preserve optimal completion, even when sampled solutions are suboptimal. Experiments show solution-quality gains across neural models and optimization problems, with robustness to graph-size and distribution shifts. Feedback-trained models also reach optimal completions more often than RL alone, without an exact solver at inference. Parameterized algorithms thus provide both *guarantees for completing neural advice* and *precise guidance for learning better decisions*.

AI USE STATEMENT

In this work, we used generative AI tools only for academic writing, plotting, literature review, and anonymizing submission materials. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. *ArXiv*, abs/2006.09607, 2020. URL https://api.semanticscholar.org/CorpusID:219721048.

Maria Chiara Angelini and Federico Ricci-Tersenghi. Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set. *Nature Machine Intelligence*, 5:29–31, 2022. URL https://api.semanticscholar.org/CorpusID:255334275.

Max Bannach, Sebastian Berndt, and Thorsten Ehlers. Jdrasil: A modular library for computing tree decompositions. In *The Sea*, 2017. URL https://api.semanticscholar.org/CorpusID:20171034.

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286 (5439):509–512, 1999.

Thomas Barrett, William Clements, Jakob Foerster, and Alex Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 3243–3250, 2020.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021.

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J. Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations, 2023. URL https://arxiv.org/abs/2111.09266.

Umberto Bertele and Francesco Brioschi. On non-serial dynamic programming. *J. Comb. Theory, Ser. A*, 14(2):137–148, 1973.

Vladimir Braverman, Prathamesh Dharangutte, Vihan Shah, and Chen Wang. Learning-Augmented Maximum Independent Set. In Amit Kumar and Noga Ron-Zewi (eds.), *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2024)*, volume 317 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 24:1–24:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-348-5. doi: 10.4230/LIPIcs.APPROX/RANDOM.2024.24. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.APPROX/RANDOM.2024.24.

Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85:12–75, 1990. URL https://api.semanticscholar.org/CorpusID:14435655.

Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, Saket Saurabh, and Marek Cygan. *Parameterized algorithms*, volume 5. Springer, 2015.

Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Algorithms with prediction portfolios. *Advances in neural information processing systems*, 35: 20273–20286, 2022.

Yinhao Dong, Pan Peng, and Ali Vakilian. Learning-augmented streaming algorithms for approximating max-cut, 2025. URL https://arxiv.org/abs/2412.09773.

P Erdös and A Rényi. On random graphs i. *Publ. math. debrecen*, 6(290-297):18, 1959.

Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 7474–7482, 2021.

David Gamarnik. Barriers for the performance of graph neural networks (gnn) in discrete random structures. *Proceedings of the National Academy of Sciences of the United States of America*, 120, 2023. URL https://api.semanticscholar.org/CorpusID:265042294.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL https://www.gurobi.com.

Rudolf Halin. S-functions for graphs. *Journal of geometry*, 8:171–186, 1976.

Qingyu Han, Linxin Yang, Qian Chen, Xiang Zhou, Dong Zhang, Akang Wang, Ruoyu Sun, and Xiaodong Luo. A gnn-guided predict-and-search framework for mixed-integer linear programming. *arXiv preprint arXiv:2302.05636*, 2023.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

Jialiang Li, Aneta Neumann, Frank Neumann, Hung Nguyen, and Mingyu Guo. Taming treewidth dp with modulators: A general booster for graph heuristics. *arXiv preprint arXiv:2608.04446*, 2026.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31, 2018.

Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pp. 777–789. SIAM, 2011.

Bertrand Marchand, Yannick Ponty, and Laurent Bulteau. Tree diet: reducing the treewidth to unlock fpt algorithms in rna bioinformatics. *Algorithms for Molecular Biology : AMB*, 17, 2021. URL https://api.semanticscholar.org/CorpusID:234349139.

Hans D Mittelmann. Benchmarking optimization software-a (hi) story. In *SN operations research forum*, volume 1, pp. 2. Springer, 2020.

Skander Moalla, Andrea Miele, Daniil Pyatko, Razvan Pascanu, and Caglar Gulcehre. No representation, no trust: connecting representation, collapse, and trust issues in ppo. *arXiv preprint arXiv:2405.00662*, 2024.

NetworkX. Networkx: Software for complex networks. URL https://networkx.org/.

Ling Pan, Nikolay Malkin, Dinghuai Zhang, and Yoshua Bengio. Better training of gflownets with local credit and incomplete trajectories. In *International Conference on Machine Learning*, pp. 26878–26890. PMLR, 2023.

Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. *Advances in Neural Information Processing Systems*, 31, 2018.

Neil Robertson and Paul D Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. *arXiv preprint arXiv:2406.01661*, 2024.

Martin J. A. Schuetz, J. Kyle Brubaker, and Helmut G. Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, April 2022. ISSN 2522-5839. doi: 10.1038/s42256-022-00468-6. URL http://dx.doi.org/10.1038/s42256-022-00468-6.

Yongho Shin, Changyeol Lee, Gukryeol Lee, and Hyung-Chan An. Improved learning-augmented algorithms for the multi-option ski rental problem via best-possible competitive analysis. In *International Conference on Machine Learning*, pp. 31539–31561. PMLR, 2023.

Angelika Steger and Nicholas C Wormald. Generating random regular graphs quickly. *Combinatorics, Probability and Computing*, 8(4):377–396, 1999.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in neural information processing systems*, 36:3706–3731, 2023.

Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. University of Massachusetts Amherst, 1984.

Dimitrios M Thilikos, Maria Serna, and Hans L Bodlaender. Cutwidth i: A linear time fixed parameter algorithm. *Journal of Algorithms*, 56(1):1–24, 2005.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

Rui Wang, Zhiming Zhou, Tao Zhang, Ling Wang, Xin Xu, Xiangke Liao, and Kaiwen Li. Learning to branch in combinatorial optimization with graph pointer networks, 2023. URL https://arxiv.org/abs/2307.01434.

Duncan J Watts and Steven H Strogatz. Collective dynamics of small-world networks. *nature*, 393 (6684):440–442, 1998.

Kexiong Yu, Hang Zhao, Yuhang Huang, Renjiao Yi, Kai Xu, and Chenyang Zhu. Disco: Efficient diffusion solver for large-scale combinatorial optimization problems. *arXiv preprint arXiv:2406.19705*, 2024.

Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial optimization problems with gflownets, 2023. URL https://arxiv.org/abs/2305.17010.

Shengyu Zhu, Ignavier Ng, and Zhitang Chen. Causal discovery with reinforcement learning. *arXiv preprint arXiv:1906.04477*, 2019.

A Proof and Further Results

Algorithm 1 gives the exact completion procedure in N-FPT. Its state validity and transition rules are instantiated for each optimization problem.

A.1 Proof of Proposition 3.1

Proof. Fix any valid advice a. By definition, $\mathcal{F}(G \mid a)$ contains exactly the valid complete solutions that agree with a on $\mathcal{M}_\eta$. The dynamic program fixes those advised decisions and enumerates every remaining bag assignment permitted by the problem-specific DP. The standard tree-decomposition transitions discard exactly the invalid or mutually inconsistent assignments. Because the dynamic program is exact, its reconstructed output is an optimizer over $\mathcal{F}(G \mid a)$. Its objective value is therefore $V_G(a)$, proving exact completion.

For consistency, suppose that a global optimum $S^\star$ agrees with a on $\mathcal{M}_\eta$. Then $S^\star \in \mathcal{F}(G \mid a)$. No solution in this restricted set can be better than a global optimum, while exact completion returns its best member. Hence the solution returned by N-FPT is globally optimal.

Algorithm 1 Exact completion in N-FPT

1: **Input:** Graph $G = (V, E)$; rooted tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$; modulator $\mathcal{M}_\eta$; valid advice $a : \mathcal{M}_\eta \to \{0, 1\}$; vertex-selection problem P.
2: **Output:** The best achievable solution consistent with a.
3: **for** $t \in V(T)$ in bottom-up order **do**
4: $U_t \leftarrow X_t \setminus \mathcal{M}_\eta$
5: Enumerate assignments $z \in \{0, 1\}^{U_t}$ and combine each with a restricted to $X_t \cap \mathcal{M}_\eta$.
6: Discard locally invalid assignments and assignments incompatible with the retained entries of a child bag.
7: Apply the problem-specific DP transition and retain the best entry for each assignment on the intersection with the parent bag.
8: **end for**
9: **return**the best entry at the root and its reconstructed solution.

For non-degradation, a_θ is the restriction of the complete valid neural solution S_θ to $\mathcal{M}_\eta$. Thus $S_\theta \in \mathcal{F}(G \mid a_\theta)$. Exact optimization over a set that contains S_θ cannot return a smaller value for maximization or a larger value for minimization. The inequality is strict precisely when $\mathcal{F}(G \mid a_\theta)$ contains a solution better than S_θ. □

A.2 More Results on Solution Quality

These tables supplement the cross-model and cross-problem comparisons in Section 3 with further graph families and reference values. Dataset specifications and the evaluation protocol are collected in Appendices B.2 and B.3.

Table conventions. The following solution-quality tables use GFlowNet. STANDALONE denotes the neural solver alone; NIC-FPT and NRD-FPT are the consolidated-prediction variants of N-FPT. Average results compare STANDALONE, N-FPT, and NIC-FPT; best-of-20 results compare STANDALONE, N-FPT, and NRD-FPT. Both groups follow the aggregation described in Appendix B.3.

Gaps are percentages relative to the starred Gurobi (600s) reference mean b, not certified optimality gaps. For reported mean size x, the gap is $100(1 - x/b)$ for MIS and MaxCut, and $100(x/b - 1)$ for MVC. Negative gaps indicate a better value than this reference. Within each sampling group, bold marks the best reported neural value in each column, including ties at the displayed precision. Table 3 reports the GFlowNet results from Table 1. Tables 9 and 8 give the full results behind Table 2, including solver references and reference gaps.

Table 3: GFlowNet results on Reg graphs for MIS. Cross-model results appear in Table 1; reporting conventions are in Appendix A.2.

METHOD	SS		SD		LS		LD	
	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓
GUROBI (60s)	280.29	3.31	170.84	2.39	358.03	4.69	216.68	4.09
GUROBI (600s)*	289.90	0.00	175.03	0.00	375.64	0.00	225.93	0.00
GFlowNet, average								
STANDALONE	286.46	1.19	170.40	2.65	372.55	0.82	221.34	**2.03**
N-FPT (ours)	286.55	1.16	170.41	2.64	372.65	0.80	**221.35**	**2.03**
NIC-FPT (ours)	**286.86**	**1.05**	**171.71**	**1.90**	**372.88**	**0.73**	221.34	**2.03**
GFlowNet, best-of-20								
STANDALONE	290.55	-0.22	174.67	0.21	377.22	-0.42	226.23	-0.13
N-FPT (ours)	290.63	-0.25	174.68	0.20	377.35	-0.46	226.25	-0.14
NRD-FPT (ours)	**293.90**	**-1.38**	**176.96**	**-1.10**	**381.78**	**-1.63**	**228.68**	**-1.22**

Table 4: MIS on WS graphs with GFlowNet. Reporting conventions are in Appendix A.2.

METHOD	SS		SD		LS		LD	
	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓
GUROBI (60s)	133.79	0.00	94.43	0.02	197.74	0.01	138.64	0.23
GUROBI (600s)*	133.79	0.00	94.45	0.00	197.75	0.00	138.96	0.00
GFlowNet, average								
STANDALONE	122.45	8.48	81.57	13.64	181.87	8.03	121.75	12.38
N-FPT (ours)	123.86	7.42	82.44	12.72	183.49	7.21	122.58	11.79
NIC-FPT (ours)	**125.09**	**6.50**	**83.44**	**11.65**	**185.31**	**6.29**	**124.10**	**10.69**
GFlowNet, best-of-20								
STANDALONE	125.06	6.52	84.43	10.61	184.86	6.52	125.56	9.64
N-FPT (ours)	126.30	5.60	85.14	9.86	186.30	5.79	126.28	9.12
NRD-FPT (ours)	**128.32**	**4.09**	**86.31**	**8.61**	**189.26**	**4.29**	**127.63**	**8.16**

Table 5: MIS on ER graphs with GFlowNet. Reporting conventions are in Appendix A.2.

METHOD	SS		SD		LS		LD	
	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓
GUROBI (60s)	137.19	2.36	64.76	3.52	142.88	7.91	64.03	11.73
GUROBI (600s)*	140.51	0.00	67.12	0.00	155.16	0.00	72.54	0.00
GFlowNet, average								
STANDALONE	133.96	4.66	63.42	5.51	150.62	2.93	71.19	1.86
N-FPT (ours)	134.17	4.51	63.47	5.44	150.72	2.86	71.20	1.85
NIC-FPT (ours)	**136.95**	**2.53**	**64.22**	**4.32**	**154.25**	**0.59**	**71.80**	**1.02**
GFlowNet, best-of-20								
STANDALONE	137.53	2.12	66.53	0.88	154.73	0.28	**74.16**	**-2.23**
N-FPT (ours)	137.74	1.97	66.57	0.82	154.80	0.23	**74.16**	**-2.23**
NRD-FPT (ours)	**139.91**	**0.43**	**66.80**	**0.48**	**156.86**	**-1.10**	73.85	-1.81

On ER-LD (Table 5), NRD-FPT has a slightly lower best-of-20 mean than N-FPT. Randomized deferral changes the neural advice, so it need not improve every sampled result. Each returned solution is nevertheless certified as best achievable under its own fixed advice; the difference concerns advice quality, not the exactness of completion.

Table 6: MIS on HK graphs with GFlowNet. Reporting conventions are in Appendix A.2.

METHOD	SS		SD		LS		LD	
	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓
GUROBI (60s)	332.69	0.00	225.52	0.80	488.54	0.01	327.94	1.36
GUROBI (600s)*	332.70	0.00	227.33	0.00	488.57	0.00	332.47	0.00
GFlowNet, average								
STANDALONE	331.74	0.29	219.89	3.27	486.20	0.49	323.81	2.61
N-FPT (ours)	331.83	0.26	221.30	2.65	486.80	0.36	325.14	2.20
NIC-FPT (ours)	**332.05**	**0.19**	**222.96**	**1.92**	**487.33**	**0.25**	**327.24**	**1.57**
GFlowNet, best-of-20								
STANDALONE	332.10	0.18	221.98	2.35	486.97	0.33	326.33	1.85
N-FPT (ours)	332.16	0.16	223.32	1.76	487.45	0.23	327.53	1.49
NRD-FPT (ours)	**332.57**	**0.04**	**226.37**	**0.42**	**488.30**	**0.05**	**331.23**	**0.38**

Table 7: MIS on BA graphs with GFlowNet. Reporting conventions are in Appendix A.2.

METHOD	SS		SD		LS		LD	
	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓
GUROBI (60s)	408.10	0.00	182.61	1.66	601.55	0.00	264.69	2.94
GUROBI (600s)*	408.10	0.00	185.69	0.00	601.55	0.00	272.70	0.00
GFlowNet, average								
STANDALONE	408.02	0.02	177.18	4.58	601.50	**0.01**	261.50	4.11
N-FPT (ours)	408.02	0.02	178.20	4.03	601.50	**0.01**	262.92	3.59
NIC-FPT (ours)	**408.04**	**0.01**	**180.95**	**2.55**	**601.51**	**0.01**	**266.62**	**2.23**
GFlowNet, best-of-20								
STANDALONE	408.05	0.01	180.09	3.01	601.51	0.01	265.17	2.76
N-FPT (ours)	408.05	0.01	181.01	2.52	601.52	**0.00**	266.44	2.30
NRD-FPT (ours)	**408.09**	**0.00**	**184.15**	**0.83**	**601.54**	**0.00**	**270.28**	**0.89**

Table 8: MVC on Reg graphs with GFlowNet. Reporting conventions are in Appendix A.2.

METHOD	SS		SD		LS		LD	
	SIZE ↓	GAP% ↓	SIZE ↓	GAP% ↓	SIZE ↓	GAP% ↓	SIZE ↓	GAP% ↓
GUROBI (60s)	566.33	1.47	678.80	0.50	743.05	2.35	885.11	0.94
GUROBI (600s)*	558.10	0.00	675.45	0.00	725.96	0.00	876.85	0.00
GFlowNet, average								
STANDALONE	561.59	0.62	680.14	0.70	729.77	0.52	881.69	**0.55**
N-FPT (ours)	561.50	0.61	680.13	0.69	729.67	0.51	**881.68**	**0.55**
NIC-FPT (ours)	**561.19**	**0.55**	**680.07**	**0.68**	**729.44**	**0.48**	881.69	**0.55**
GFlowNet, best-of-20								
STANDALONE	557.50	-0.11	675.87	0.06	725.10	-0.12	876.80	-0.01
N-FPT (ours)	557.42	-0.12	675.86	0.06	724.97	-0.14	876.78	-0.01
NRD-FPT (ours)	**554.15**	**-0.71**	**673.58**	**-0.28**	**720.54**	**-0.75**	**874.35**	**-0.29**

Table 9: MaxCut on Reg graphs with GFlowNet. Reporting conventions are in Appendix A.2.

METHOD	SS		SD		LS		LD	
	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓	SIZE ↑	GAP% ↓
GUROBI (60s)	2006.10	0.00	4644.71	0.00	2601.79	0.00	6006.83	0.00
GUROBI (600s)*	2006.15	0.00	4644.71	0.00	2601.82	0.00	6006.83	0.00
GFlowNet, average								
STANDALONE	1944.50	3.07	4458.06	4.02	2582.00	0.76	5767.26	3.99
N-FPT (ours)	1967.72	1.92	4474.58	3.66	2587.64	0.55	5792.78	3.56
NIC-FPT (ours)	**1996.30**	**0.49**	**4490.85**	**3.31**	**2608.50**	**-0.26**	**5824.44**	**3.04**
GFlowNet, best-of-20								
STANDALONE	1963.73	2.11	4492.65	3.27	2601.64	0.01	5804.41	3.37
N-FPT (ours)	1986.51	0.98	4509.91	2.90	2606.80	-0.19	5830.97	2.93
NRD-FPT (ours)	**2007.57**	**-0.07**	**4544.66**	**2.15**	**2618.95**	**-0.66**	**5894.62**	**1.87**

A.3 MORE RESULTS ON GENERALIZATION

Figures 9–12 extend Figure 3 with LD-trained models and further source families. The reference and sampling measures follow Appendix B.3. Each figure places SS-trained models above LD-trained models.

All plots report the gain over the target-trained neural reference, $100(B/A-1)$, as in Figure 3; higher is better. Within-family results use average solution size and compare STANDALONE, N-FPT, and NIC-FPT. Cross-family results use best-of-20 and compare STANDALONE, N-FPT, and NRD-FPT. These gains are not certified optimality gaps.

The additional source families show that the size of the transfer loss depends on where the neural model was trained. Exact completion can improve transferred solutions across these families. The variants do not remove every deficit relative to the target-trained reference: the quality of the fixed

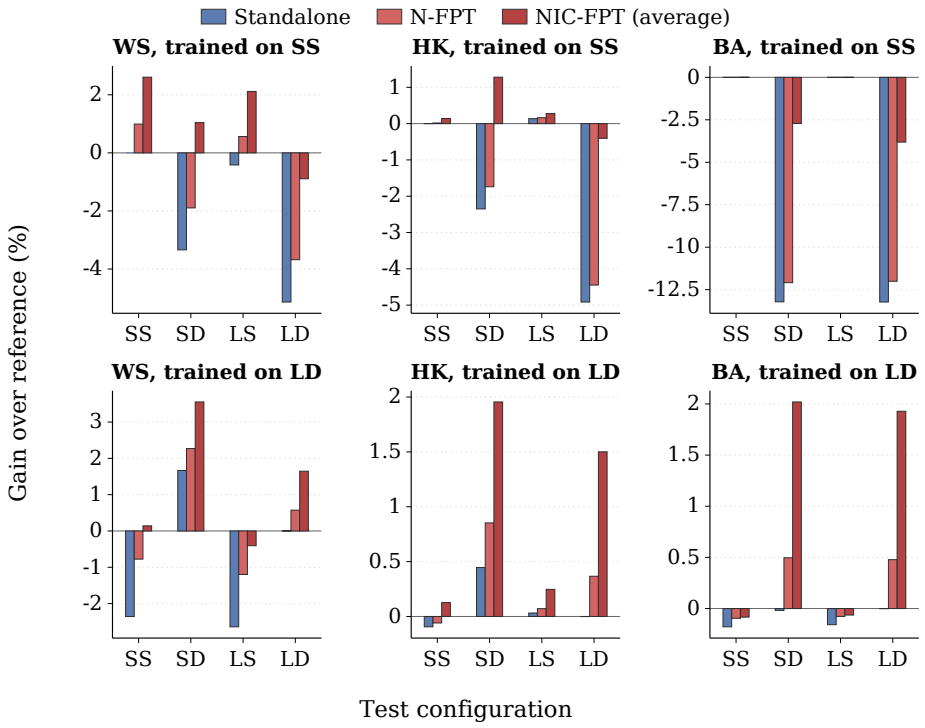


Figure 9: Within-family generalization: percentage gains in average solution size. Higher is better.

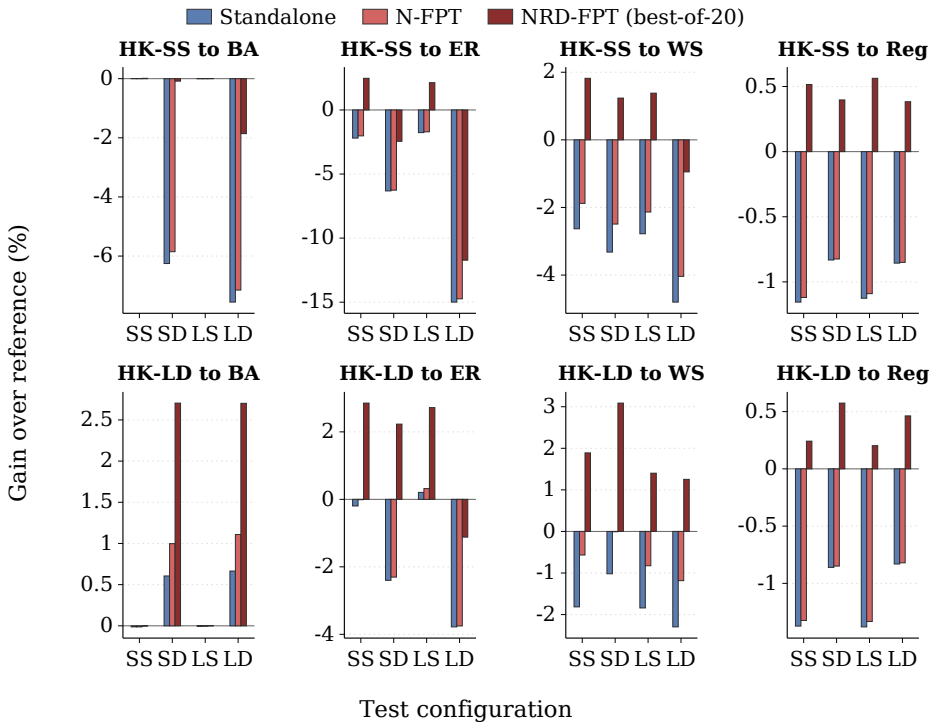


Figure 10: Cross-family generalization from HK graphs: percentage gains in best-of-20 solution size. Higher is better.

advice still matters. SS and LD bars also have different target-trained references, so their heights compare relative performance, not absolute solution sizes.

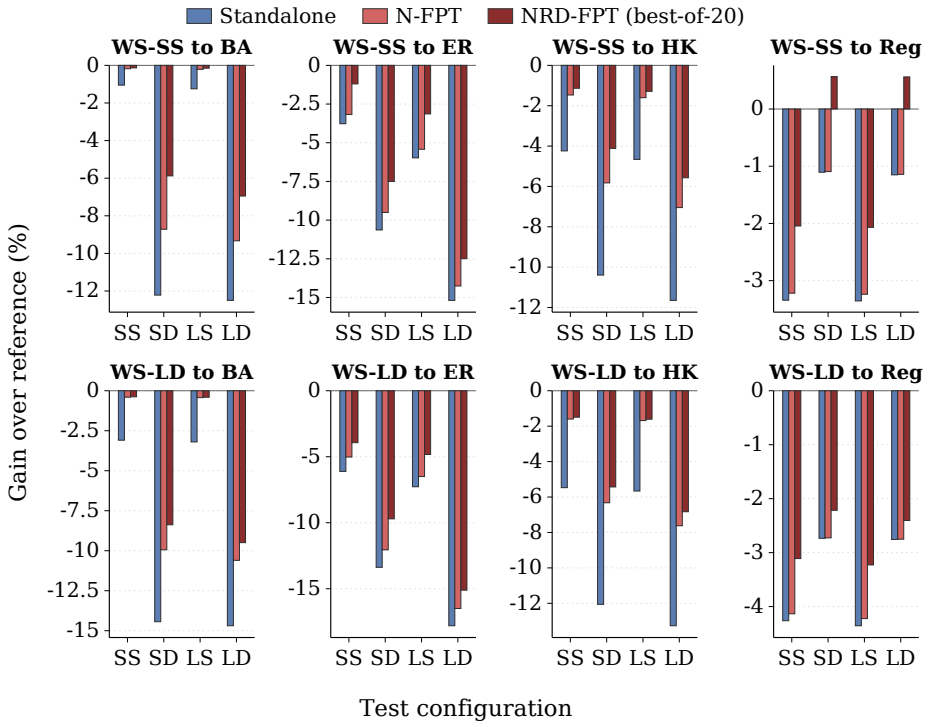


Figure 11: Cross-family generalization from WS graphs: percentage gains in best-of-20 solution size. Higher is better.

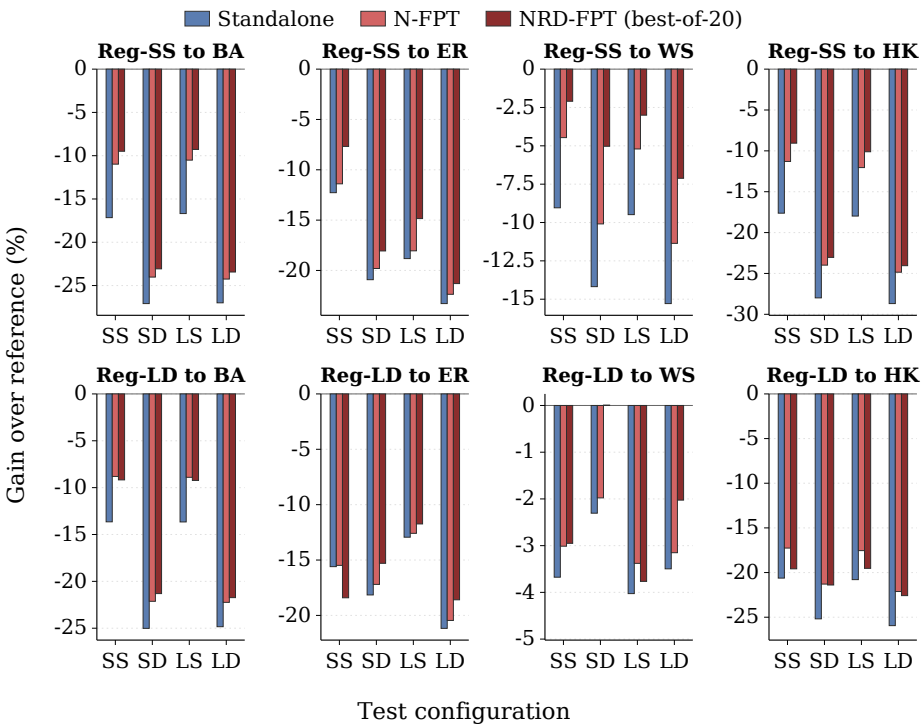


Figure 12: Cross-family generalization from Reg graphs: percentage gains in best-of-20 solution size. Higher is better.

A.4 Further Results on Learned Completion

For each decoded solution S, we compute the percentage gap $100(1 - f_G(S)/V_G(a_0))$, then average over test graphs and training seeds. For the paired comparisons, we average each method's solution size over its four trained models on each test graph, then record a win, tie, or loss for RL with N-FPT feedback relative to RL. The plotted p-values are from two-sided exact sign tests on non-tied graphs, testing whether either method is equally likely to win.

Under one-shot decoding (Figure 6), feedback improves 57–75 of the 100 test instances per setting and worsens only 6–19. Figure 13 reports the corresponding five-shot results: each trained model returns its best of five sampled completions, with all other settings unchanged. Feedback improves 49–78 instances per setting and worsens only 2–9. Under both decoding budgets, feedback lowers the mean gap in every setting, and all sign tests give $p < 10^{-7}$. The advantage of feedback therefore persists when both methods can sample several completions.

Preserving Optimal Completion during Training. Figure 8 tracks how often sampled trajectories lose optimal completion and where the first harmful selection occurs. Early in training, nearly all sampled trajectories miss the best achievable value, yet they already provide exact corrective lessons. As training progresses, fewer trajectories lose optimal completion and, among those that still do, the first harmful selection generally occurs later. These trends support the decision-level benefit observed on test trajectories in Figure 7.

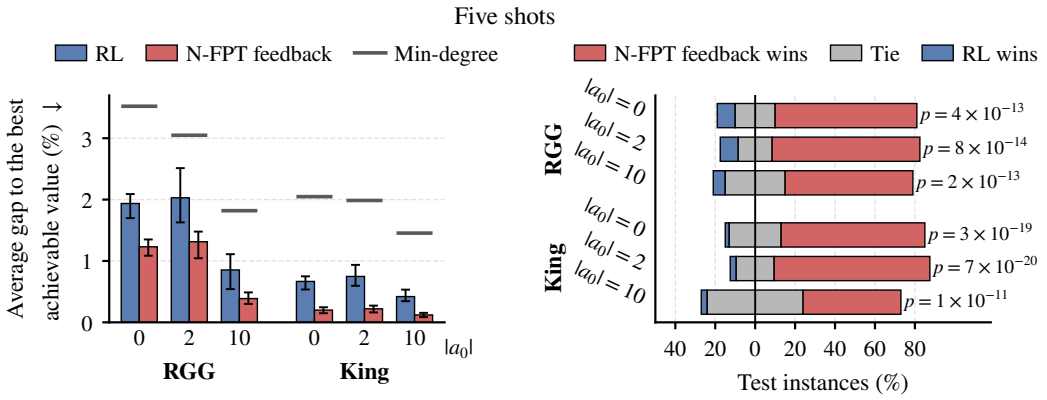


Figure 13: Five-shot completion quality; visual conventions follow Figure 6.

B Reproduction Notes

B.1 Hardware

The original experiments used an Intel Xeon Platinum 8360Y CPU at 2.40 GHz, up to 32 GB of RAM, and a single NVIDIA A100-SXM4 GPU. Additional experiments ran on NCI's Gadi supercomputer, primarily using CPUs and NVIDIA V100 GPUs, consuming approximately 20,000 service units. We also used a Slurm-managed cluster, consuming approximately 20,000 service units under that system's accounting.

B.2 Inference Datasets

We vary graph families and cross graph size with density within each family to test whether gains depend on a particular structure or distribution. The families include Erdős–Rényi (ER) (Erdös & Rényi, 1959), Barabási–Albert (BA) (Barabási & Albert, 1999), Watts–Strogatz (WS) (Watts & Strogatz, 1998), random regular (Reg) (Steger & Wormald, 1999), and Holme–Kim (HK) graphs. We generate them with NetworkX (NetworkX), following the parameters in Table 10. Each configuration contains 100 training and 100 test graphs generated with distinct random seeds. Vertex counts are sampled uniformly from the listed ranges; the other parameter names follow NetworkX. We remove self-loops and isolated vertices and relabel vertices with consecutive integers for implementation purposes.

The four configurations are small sparse (SS), small dense (SD), large sparse (LS), and large dense (LD). For the Reg graphs in Tables 1 and 2, small instances have 800–900 vertices and large instances have 1,000–1,200; the sparse and dense settings use degrees 6 and 16, respectively. Cross-size tests hold the other generation parameters fixed; cross-distribution tests change those parameters while retaining the same vertex-count range. Vertex count alone does not determine the difficulty of exact completion: the residual width controls its exponential cost, while the modulator determines how many decisions rely on neural advice.

B.3 Inference Evaluation Protocol

Tables 1 and 2 use 100 test graphs per configuration and three training seeds for each neural model. We retain 20 final samples per graph. The average result is their mean objective value; best-of-20 takes the largest value for MIS and MaxCut and the smallest for MVC. Both are then averaged over test graphs and training seeds. MaxCut values count crossing edges; MIS and MVC values count selected vertices. Standalone denotes the trained neural solver alone. Each N-FPT result exactly completes the modulator advice from its paired neural sample. Residual-width settings and the exact construction are in Appendix B.8. NIC-FPT and NRD-FPT use the five-round consolidated-prediction protocols in Appendix B.10. They use additional inference sampling, not additional training. Training details are in Appendices B.5 and B.6.

Table 10: Graph-generation settings and mean vertex/edge counts for the original datasets. Parameter names follow NetworkX (NetworkX) and are independent of the notation elsewhere in the paper.

DATASET	CONFIGURATION	MEAN $(\lvert V\rvert, \lvert E\rvert)$
$\mathrm{ER}_{\mathrm{SS}}$	$[700, 800], p = 0.03$	$(748.05, 8408.72)$
$\mathrm{ER}_{\mathrm{SD}}$	$[700, 800], p = 0.08$	$(750.54, 22551.89)$
$\mathrm{ER}_{\mathrm{LS}}$	$[1000, 1200], p = 0.03$	$(1102.32, 18258.46)$
$\mathrm{ER}_{\mathrm{LD}}$	$[1000, 1200], p = 0.08$	$(1103.03, 48750.79)$
$\mathrm{BA}_{\mathrm{SS}}$	$[700, 800], m = 3$	$(748.46, 1840.49)$
$\mathrm{BA}_{\mathrm{SD}}$	$[700, 800], m = 15$	$(749.43, 10655.76)$
$\mathrm{BA}_{\mathrm{LS}}$	$[1000, 1200], m = 3$	$(1101.31, 2702.49)$
$\mathrm{BA}_{\mathrm{LD}}$	$[1000, 1200], m = 15$	$(1090.02, 15538.75)$
$\mathrm{WS}_{\mathrm{SS}}$	$[700, 800], k = 15, p = 0.1$	$(748.05, 5236.35)$
$\mathrm{WS}_{\mathrm{SD}}$	$[700, 800], k = 25, p = 0.1$	$(750.54, 9006.48)$
$\mathrm{WS}_{\mathrm{LS}}$	$[1000, 1200], k = 15, p = 0.1$	$(1102.32, 7716.24)$
$\mathrm{WS}_{\mathrm{LD}}$	$[1000, 1200], k = 25, p = 0.1$	$(1103.03, 13236.36)$
$\mathrm{Reg}_{\mathrm{SS}}$	$[800, 900], d = 6$	$(848.05, 2544.15)$
$\mathrm{Reg}_{\mathrm{SD}}$	$[800, 900], d = 16$	$(850.54, 6804.32)$
$\mathrm{Reg}_{\mathrm{LS}}$	$[1000, 1200], d = 6$	$(1102.32, 3306.96)$
$\mathrm{Reg}_{\mathrm{LD}}$	$[1000, 1200], d = 16$	$(1103.03, 8824.24)$
$\mathrm{HK}_{\mathrm{SS}}$	$[700, 800], m \in [3, 7], p = 0.3$	$(748.46, 3299.16)$
$\mathrm{HK}_{\mathrm{SD}}$	$[700, 800], m \in [10, 15], p = 0.3$	$(746.66, 8743.89)$
$\mathrm{HK}_{\mathrm{LS}}$	$[1000, 1200], m \in [3, 7], p = 0.3$	$(1101.31, 4941.53)$
$\mathrm{HK}_{\mathrm{LD}}$	$[1000, 1200], m \in [10, 15], p = 0.3$	$(1096.27, 13172.38)$

For the MIS comparison in Table 1, GFlowNet and Gurobi use the original graph instances; GNN+RL and KaMIS use newly generated instances with the same configuration ranges. Within each neural model, the standalone solver and all N-FPT variants share checkpoints and test graphs. Thus, the paired evidence for improvement is within each neural model, not a ranking of solvers across independently generated datasets.

Gurobi provides a reference across the optimization problems (Mittelmann, 2020). We use 60s and 600s limits with a single thread, retaining the best incumbent when a run reaches its limit, and average over 10 runs with different random seeds. KaMIS runs for 600 seconds per instance on the GNN+RL MIS test graphs.

Table 1 reports the reference gap as $100(1 - x/b)$, where x is the reported mean size and b is the original Gurobi (600s) reference mean marked with $*$. These values are not certified optima, nor are they paired references for the newly generated GNN+RL instances. Negative gaps indicate a size above the reference mean, not improvement over a proven optimum. Arrows indicate the preferred direction: larger size and smaller reference gap. Table 2 reports objective values only; the full tables and problem-specific gap definitions are in Appendix A.2.

Generalization. The generalization experiments use GFlowNet on MIS. Within each of the WS, HK, and BA families, we train on SS or LD and test on all four configurations. The matching training configuration is an in-distribution control; the others change graph size, density, or both. Across families, we train on HK, WS, or Reg graphs, using SS or LD, and test on all four configurations of each other family. Model weights remain fixed throughout evaluation.

For each target family and size–density configuration, the reference A is the result of a standalone GFlowNet trained specifically on that configuration. The compared value B comes from the source-trained model, either standalone or within an N-FPT method. Within-family results use average solution size; across-family results use best-of-20, with the same measure for A and B. Figure 3 reports $100(B/A - 1)$: zero matches the target-trained reference, and positive values indicate larger solutions. This is a comparison with a neural reference, not an optimality gap. The figure uses the SS-trained models, with HK as the cross-family source; Appendix A.3 includes LD-trained models and the other source families.

B.4 Problem Definitions and Gurobi

For MIS, the valid solution family and objective are

$$\mathcal{F}_{\mathrm{MIS}}(G) = \{S \subseteq V : \forall (u,v) \in E,\ u \notin S \text{ or } v \notin S\}, \qquad \max_{S \in \mathcal{F}_{\mathrm{MIS}}(G)} |S|.$$

For MVC, they are

$$\mathcal{F}_{\mathrm{MVC}}(G) = \{S \subseteq V : \forall (u,v) \in E,\ u \in S \text{ or } v \in S\}, \qquad \min_{S \in \mathcal{F}_{\mathrm{MVC}}(G)} |S|.$$

For MaxCut, every subset $S \subseteq V$ defines the cut $\delta(S) = \{(u,v) \in E : |\{u,v\} \cap S| = 1\}$, and the objective is $\max_{S \subseteq V} |\delta(S)|$.

B.5 More Details on GFlowNet Training.

We train GFlowNet following prior work (Zhang et al., 2023). In addition, we make two modifications to improve performance. First, we apply the non-annealing setting during training. Second, we replace the average pooling layer from the original policy network with *a sum pooling layer*. The previous study limits training to 20 epochs, with convergence often occurring within 5. However, due to the more dynamic and heterogeneous nature of our dataset, and to further investigate the best GFlowNet can achieve, we increase the training budget to 50 epochs for each configuration. Models are saved at every epoch, and evaluation is conducted using the model that achieves the best performance. We train each configuration with three random seeds and report the mean performance across these runs.

B.6 GNN+RL Training and Evaluation

The GNN+RL policy is a five-layer graph attention network with hidden dimension 256 and four attention heads. It observes each vertex's decision status and the number of undecided neighbors. At each step, it samples an undecided vertex, selects it, and excludes its neighbors. We train with REINFORCE on 100 training graphs per configuration for 50 epochs, using batches of up to 64 graphs and four sampled solutions per graph. Each solution's advantage is its size minus the mean size of the other three samples from the same graph. We use Adam with learning rate 10^{-3} and gradient clipping at norm 1. Results use the final checkpoint from each of three training seeds and 20 sampled solutions per test graph.

B.7 Neural Completion Training and Evaluation

The learning experiments in Section 4 use MIS on random geometric graphs (RGG) and king's graphs with random vertex deletions (King), each with about 300 vertices. Each family has 500 training graphs, 50 validation graphs, and 100 test graphs. We draw a target size n_{target} uniformly from the integers 270–330. RGG places this many points uniformly in the unit square and joins points within distance 0.103. King starts from a square grid of side length $\mathrm{round}(\sqrt{n_{\mathrm{target}}/0.7})$, retains each vertex independently with probability 0.7, and joins retained vertices that are horizontal, vertical, or diagonal neighbors. We remove isolated vertices and self-loops and relabel vertices consecutively.

The three starting-state settings use the same graphs and contain zero, two, or ten initial selections. For each graph and setting, we generate one starting assignment by uniformly sampling an undecided vertex at each selection and excluding its neighbors. This assignment stays fixed across epochs and is shared by both methods.

Both RL and RL with N-FPT feedback use a five-layer graph attention network with hidden dimension 256 and four attention heads of dimension 64. The inputs are vertex-status embeddings and projected log-transformed counts of undecided neighbors. The network uses residual connections and ELU activations, without dropout or normalization layers, followed by a two-layer output network with a ReLU activation. Policies are trained from scratch for 20 epochs with matched seeds 1–4. We train separate policies for each graph family and starting-state setting.

We use Adam with learning rate 10^{-4} and gradient-norm clipping at 1. Each epoch visits all 500 training assignments in random order, with four sampled trajectories and one optimizer update per

assignment, giving 10,000 updates per run. Gradient computation processes at most 64 graph-state copies per network pass. The feedback weight in Equation 4 is $\lambda = 1$; the RL baseline omits that term. Architecture, training graphs, starting assignments, and optimizer-update counts are otherwise matched. After each epoch, we evaluate greedy completion on all 50 validation graphs and retain the checkpoint with the largest mean solution size for testing.

For these runs, exact feedback is prepared at the given starting assignments when affordable, corresponding to $s = 0$ in Section 4. We cap the exact computation at 200,000 valid assignments per bag and the stored optimal-completion representation at 2,000,000 bag states per starting assignment. The representations are cached and reused across epochs. Unavailable feedback is omitted; these graphs remain in RL training rather than being removed from the dataset.

Gurobi proves the reference value $V_G(a_0)$ for every test graph under the same fixed assignment. These references only assess the outputs; each trained model constructs its test completions without an exact solver. One-shot decoding selects the highest-probability vertex at each step; five-shot decoding returns the best of five sampled completions. Every setting is evaluated against its own best achievable value. We report optimal-completion frequency, mean percentage gap, and paired instance-level comparisons; aggregation and statistical tests are specified in Appendix A.4.

Trajectory diagnostics. Figure 8 uses the trajectories sampled during each training epoch for which exact feedback is available at a_0 ($s = 0$). For each suboptimal trajectory, we divide the index of its first harmful selection by its number of additional selections and report the mean as a percentage. The failure rate is the fraction of all sampled trajectories from these starts that end below $V_G(a_0)$. Starting assignments without the stored representation are excluded from both diagnostics.

Figure 7 replays the one-shot test trajectories through the same type of stored representation. Completion progress is the fraction of a trajectory's additional selections, not the fraction of graph vertices decided. At each progress level, we count the test graphs on which no harmful selection has yet occurred. Each curve therefore starts at 100% and ends at the corresponding one-shot optimal-completion frequency. All replayed solution sizes match the original evaluation. For these test diagnostics, exact computation only diagnoses completed trajectories; it neither chooses nor changes their selections. In both figures, lines average the four training seeds and shaded bands span their minimum and maximum.

B.8 Tree Decomposition, Modulator, and Exact Completion

We construct the initial tree decomposition using the min-degree heuristic. For a vertex u, let $N(u)$ and $N[u] = N(u) \cup \{u\}$ denote its open and closed neighborhoods. As shown in Algorithm 2, the heuristic repeatedly selects a minimum-degree vertex u, adds fill edges that make $N(u)$ a clique, creates the bag $N[u]$, and removes u. This produces a valid decomposition efficiently, although not necessarily one of minimum width (Bannach et al., 2017).

Algorithm 2 Min-degree tree-decomposition heuristic

Require: Graph $G = (V, E)$
Ensure: A valid tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$
$H \leftarrow G$; initialize an empty elimination sequence L
while $V(H) \neq \emptyset$ **do**
  Choose a minimum-degree vertex $u \in V(H)$
  Record the bag $X_u = \{u\} \cup N_H(u)$ in L
  Add fill edges that make $N_H(u)$ a clique
  Remove u and its incident edges from H
end while
Create one tree node for every recorded bag X_u
In reverse elimination order, connect each non-root bag X_u to a later bag that contains $X_u \setminus \{u\}$
returnthe constructed decomposition $\mathcal{T}$

The minimum modulator for the fixed decomposition is obtained from the binary program

$$\begin{aligned} \min_{z\in\{0,1\}^V} \quad & \sum_{v\in V} z_v \\ \text{subject to} \quad & \sum_{v\in X_t} (1-z_v) \le \eta + 1 \qquad \text{for every } t \in V(T). \end{aligned} \tag{5}$$

Here $z_v = 1$ means $v \in \mathcal{M}_\eta$. The objective therefore returns the minimum modulator for the given decomposition, not an approximation to it. We solve this MILP exactly. We use target width $\eta = 10$ for MIS and MVC and $\eta = 6$ for MaxCut. The vertex-deletion version used here follows Li et al. (2026); an edge-deletion variant on a fixed decomposition was studied by Marchand et al. (2021).

For completeness, deleting $\mathcal{M}_\eta$ from every bag preserves the tree-decomposition axioms for $G \setminus \mathcal{M}_\eta$. Vertex coverage holds for every remaining vertex, every remaining edge stays covered by its original bag, and the bags containing any remaining vertex are unchanged and thus still connected. The MILP constraints then bound every remaining bag by $\eta + 1$ vertices.

We also prune a bag when it is contained in an adjacent bag and reconnect its neighbors through the containing bag. This preserves vertex and edge coverage; for every vertex appearing on both sides of the removed bag, the containing bag preserves the running-intersection property. Figure 14 shows the three local configurations used by this operation.

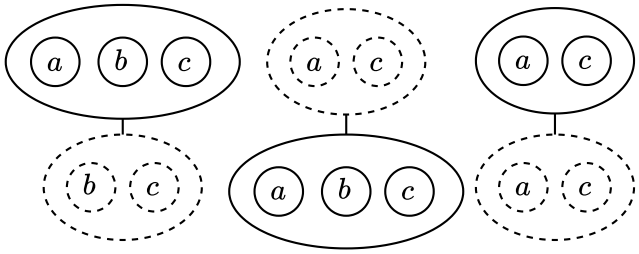


Figure 14: Three valid bag-pruning configurations. A dashed bag is removed when an adjacent bag contains it; reconnecting its neighbors through the containing bag preserves the tree-decomposition axioms.

B.9 Advice Validity and Boundary Propagation

The advice used in our experiments is the restriction of a complete valid neural solution to $\mathcal{M}_\eta$, so it always admits that neural solution as a completion. The dynamic program in N-FPT also propagates the constraints created at the boundary of the modulator. For MIS, every selected modulator vertex forces its neighbors outside the modulator to be excluded. For MVC, every excluded modulator vertex forces its neighbors outside the modulator to be selected. MaxCut permits every binary assignment on the modulator and needs no analogous repair. These rules make the DP enumerate exactly $\mathcal{F}(G \mid a)$. If advice is supplied by another source and is invalid within the modulator, a task-specific repair is applied before exact completion; the guarantees then apply to the repaired advice.

B.10 Consolidated-Prediction Variants

NIC-FPT. At each round, the neural sampler draws K valid completions $S^{(1)}, \ldots, S^{(K)}$ conditioned on the current consensus. For an undecided vertex v, its selection count is

$$c_v = \sum_{j=1}^{K} \mathbf{1}\{v \in S^{(j)}\}.$$

NIC-FPT fixes v as selected when $c_v > \kappa$ and excludes its neighbors. We use five rollout rounds with $K = 20$ and $\kappa = 0.8K$; the first four rounds update the consensus and the fifth completes the solution. Because $\kappa > K/2$ and every sampled MIS solution is an independent set, two adjacent vertices cannot both pass the threshold: their counts sum to at most K. The consensus is therefore valid and conditions the next sampling round. The final valid neural sample supplies modulator advice a; NIC-FPT returns its best achievable completion $V_G(a)$.

NRD-FPT. Starting from a complete valid neural sample, NRD-FPT independently defers each selected vertex with probability $\rho = 0.8$. Retained selections and their implied neighbor exclusions stay fixed; the sampler resamples a valid completion over the other vertices. We use five rollout rounds, applying deferral after the first four. The final valid neural sample supplies modulator advice a, and NRD-FPT returns its best achievable completion $V_G(a)$.

C Practical Runtime Performance

We report empirical runtime results for the neural solver and exact completion in N-FPT. For the neural solver, we reuse the GFlowNet implementation from Zhang et al. (2023). The cost of exact completion depends on the number of bag assignments considered by the dynamic program.

Following the prior metrics, we compute runtime per dataset, with each dataset processed in mini-batch. These batches serve as inputs to the training models during evaluation. We present the time consumption of each component separately to show their individual contributions.

Table 11: Runtime (seconds per dataset) of the neural and exact-completion components of N-FPT. One-time preprocessing is excluded.

Dataset	SS		SD		LS		LD	
	Neural	Exact	Neural	Exact	Neural	Exact	Neural	Exact
Reg	95.79	147.48	60.33	39.30	159.04	183.29	102.76	53.54
ER	44.40	43.09	26.14	18.38	71.37	39.52	44.38	21.36
HK	113.52	498.87	78.41	248.59	234.93	776.04	163.07	343.75
BA	128.30	712.77	63.82	134.72	266.63	1335.09	131.62	218.33
WS	49.70	265.10	38.65	165.46	85.82	366.77	57.87	243.94

Another relevant consideration is the time required to construct the tree decomposition and compute its minimum modulator. In our experiments, this preprocessing takes approximately *one second* per instance on average. We exclude it from the component-level runtimes above because it is a one-time initialization cost and is negligible relative to the reported total.

D Related Work

Neural Solvers for Graph Combinatorial Optimization. Neural solvers span two inference patterns. *One-shot* solvers, including GNN-based neural solvers (Schuetz et al., 2022; Gasse et al., 2019; Wang et al., 2023), attention-based models (Vinyals et al., 2015; Kool et al., 2018) and diffusion models (Sun & Yang, 2023; Sanokowski et al., 2024; Yu et al., 2024), directly output solution in a single or several steps. Some further improve the results via MCTS approaches (Han et al., 2023; Fu et al., 2021). These are trained supervised or via handcrafted objectives. They often learn latent representations that enable graph combinatorial optimization problems to be formulated as a vertex classification task. Despite strong inference-time performance, solution quality often degrades, especially for GNN-only models (Angelini & Ricci-Tersenghi, 2022; Gamarnik, 2023).

A second approach frames graph combinatorial optimization as a Markov decision process (MDP), making RL a natural option (Khalil et al., 2017; Bello et al., 2016; Ahn et al., 2020). RL-based neural solvers sequentially complete solutions through vertex decisions. These models use different neural architectures as *state encoders*, such as Transformer (Zhu et al., 2019), GNNs (Li et al., 2018), RNNs (Vinyals et al., 2015), etc. GFlowNet (Bengio et al., 2023; 2021) enhances sampling through flow-based losses (Pan et al., 2023), balancing solution quality and diversity.

RL-based approaches make vertex decisions autoregressively; some models generate multiple decisions simultaneously, incorporating modified environments and reward shaping to guide learning. However, RL-based methods can suffer policy collapse (Moalla et al., 2024) and credit assignment difficulties (Sutton, 1984), limiting exploration. Despite their successes, neural solvers can struggle with solution-quality guarantees and generalization. N-FPT uses their decisions as advice and certifies the best achievable completion under that advice.

Parameterized Algorithms. The complexity of graph problems is often measured by the number of vertices and edges. Parameterized algorithms also account for a chosen parameter (Fomin et al., 2019), such as optimal solution size, treewidth (Bertele & Brioschi, 1973; Halin, 1976; Robertson & Seymour, 1984), or cutwidth (Thilikos et al., 2005). FPT algorithms have polynomial dependence on input size for a fixed parameter value. Structural parameters such as treewidth can also reveal which parts of an instance contribute most to the computational difficulty.

Learning-Augmented Algorithms. Learning-augmented algorithms use predictions as advice to improve algorithmic performance. Much of the literature focuses on online algorithms (Purohit et al., 2018), though recent work extends to graph combinatorial optimization, including MaxCut (Dong et al., 2025) and independent set (Braverman et al., 2024). This connection with predictive models has also motivated practical applications (Shin et al., 2023; Dinitz et al., 2022). The advice may be imperfect; algorithmic guarantees specify how the result depends on its quality.

E Future Directions

Additional neural models, such as *diffusion models*, can supply advice to N-FPT. Similarly, other parameterized algorithms can take the place of treewidth dynamic programming. For instance, if we replace treewidth dynamic programming with parameterized approximation schemes or randomized parameterized algorithms, we may obtain other learning-augmented algorithms with different runtime and solution-quality guarantees. The learning-augmented framework also motivates *beyond-worst-case* analysis combining structural assumptions, such as small treewidth modulators, with assumptions on neural advice.