Infeasibility Aware Large Language Models for Combinatorial Optimization

Anonymous ACL submission

Abstract

Large language models (LLMs) are increasingly being explored for NP-hard combinatorial optimization problems, but most existing methods emphasize feasible-instance solution generation and do not explicitly address infeasibility detection. We propose an infeasibility-aware framework that combines certifiable dataset construction, supervised fine-tuning, and LLM-assisted downstream search. For the minor-embedding problem, we introduce a new mathematical programming formulation together with provable zero-phase infeasibility screening, which enables scalable construction of training instances labeled either as feasible with structured certificates or as certifiably infeasible. Using training data generated through this exact optimization pipeline, we show that an 8B-parameter LLM can be fine-tuned to learn feasibility-status prediction and structured solution generation. We further utilize LLM outputs as warm starts for downstream local search, providing a practical way to accelerate optimization even when the LLM outputs are imperfect. Experiments show that our fine-tuned model improves overall accuracy by up to 20% over GPT-5.2; Meanwhile, LLM-guided warm starts improved the success rate of local search compared to starting from scratch.

1 Introduction

Large language models (LLMs) have rapidly progressed beyond traditional natural language processing (NLP) tasks and are increasingly viewed as general-purpose reasoning and decision-making agents (Yao et al., 2022; Huang et al., 2022). This evolution has sparked growing interest in whether LLMs can provide useful structured predictions, heuristics, or initialization signals for NP-hard combinatorial optimization (CO) problems (Yang et al., 2023; Xiao et al., 2023; AhmadiTeshnizi et al., 2023), which arise ubiquitously in real-world applications (Da Ros et al., 2025; Bengio et al., 2021).

Recent research on LLMs for combinatorial optimization falls into two paradigms: The first leverages LLMs as assistive tools to generate executable code, discover heuristics, or interact with mathematical optimization solvers (Xiao et al., 2023; Jiang et al., 2024; Huang et al., 2025; Ye et al., 2024). While effective, these approaches still require substantial domain expertise; Another paradigm explores LLMs that directly generate candidate solutions for CO instances in a zero-shot or few-shot setting, without relying on problem-specific templates or external solver calls (Yang et al., 2023; Jiang et al., 2025). However, the applicability of these approaches to more complex and practically relevant CO problems remains largely unexplored. Moreover, existing end-to-end LLM solvers are not reliable enough since they typically lack infeasibility detection and do not incorporate systematic post-processing mechanisms to repair or validate incorrect solutions. A further challenge lies in the construction of training datasets: For some classical CO problems, well-established algorithms can generate reliable ground truth solutions at scale. In contrast, for emerging and structurally complex problems such as minor-embedding, current optimization solvers are limited in producing solutions or infeasibility certificates within reasonable time (Bernal et al., 2020), making supervised learning particularly challenging.

Motivated by these gaps, we propose an infeasibility-aware framework for applying LLMs to NP-hard combinatorial optimization problems. The framework consists of three stages: (i) construction of training datasets containing both feasible instances and certifiably infeasible instances, (ii) supervised fine-tuning of a small language model to jointly predict structured solutions or infeasibility, and (iii) LLM-guided downstream search, where model outputs are used to initialize local search procedures. This design allows the LLM to serve two roles: predicting infeasibility

when no feasible solution exists, and producing structured candidate solutions that can be verified or repaired by downstream algorithms. We focus on the minor-embedding problem as our primary case study because it is a fundamental task in quantum computing (Choi, 2008) where even determining feasibility is NP-hard (Lobe and Lutz, 2021). Its intricate constraints often exceed the capabilities of classical solvers like Gurobi (Gurobi Optimization, LLC, 2026) or OR-Tools (Perron and Furnon, 2025), making it an ideal testbed for LLM-based initialization. To demonstrate that the resulting workflow is not specific to minor-embedding, we also evaluate it on k-coloring as a second NP-hard problem.

Our contributions are summarized as follows:

- We propose an infeasibility-aware workflow for LLM-based combinatorial optimization that combines three components: the construction of training datasets containing both feasible and certifiably infeasible instances, supervised fine-tuning for structured solution prediction and infeasibility detection, and LLM-assisted initialization of downstream local search.
- For the minor-embedding problem, we develop a new dataset construction methodology based on mathematical programming formulations and infeasibility certification, enabling scalable generation of labeled instances for a problem where both structured feasible solutions and reliable infeasibility labels are difficult to obtain.
- We show that a fine-tuned 8B-parameter LLM can learn useful patterns for feasibility-status prediction and structured candidate-solution generation from certifiably labeled instances. When used as warm starts, these outputs also improve downstream local search. Across minor-embedding and k-coloring, the resulting approach improves overall accuracy by up to 20% over a general-purpose LLM and increases the success rate of downstream local search compared to starting from scratch.

2 Related Work

This section reviews prior work most relevant to our setting: LLMs for combinatorial optimization and reasoning under hard constraints, and existing methods for the minor-embedding problem.

LLMs for combinatorial optimization. Recent work has explored several roles for LLMs in combinatorial optimization. One line of research uses LLMs to generate reusable solver components, such as heuristics, search operators, or executable code, as in FunSearch (Romera-Paredes et al., 2024), ReEvo (Ye et al., 2024), and subsequent reinforcement-learning-based extensions (Surina et al., 2025). A second line studies LLMs as auto-formulation engines that translate natural-language descriptions into mathematical models, solver code, or optimization pipelines, including Chain-of-Experts (Xiao et al., 2023), OptiMUS (AhmadiTeshnizi et al., 2023, 2024), tree-search-based formulation methods (Jiang et al., 2024), and ORLM (Huang et al., 2025). A third line evaluates LLMs as solvers that directly produce structured answers for problem instances, either by zero-shot generation or through iterative refinement (Jiang et al., 2025; Yang et al., 2023; Liu et al., 2024). Related hybrid methods further combine language models with symbolic or logic-based reasoning modules to improve correctness on constrained tasks (Ye et al., 2023; Zhang et al., 2024). Despite these advances, prior work primarily emphasizes candidate generation, code synthesis, or solver-aided reasoning over infeasibility detection (Da Ros et al., 2025). While some recent work uses LLMs to diagnose faulty optimization models (Chen et al., 2024), directly predicting infeasibility for combinatorial instances remains largely unexplored. Recent boolean satisfiability problem (SAT) oriented studies also suggest that feasibility reasoning remains challenging for current language models, especially on hard or UNSAT-style instances (Pan et al., 2024; Hazra et al., 2025; Wei et al., 2025). Unlike solver-aided reasoning methods that depend on inference-time symbolic backends, our framework uses exact optimization strictly offline for data construction. This difference is especially relevant for structured combinatorial optimization, where infeasibility often depends on problem-specific certificates rather than generic logical encodings that can be unnatural or unscalable in practice.

Minor-embedding. The minor-embedding problem is a fundamental preprocessing step in quantum annealing (QA) (Kadowaki and Nishimori, 1998), requiring the logical problem graph to be mapped into the sparse hardware graph of a quantum processing unit by representing each logical variable

as a connected chain of physical qubits (Choi, 2008, 2011). This problem is NP-hard (Lobe and Lutz, 2021), and embedding quality directly impacts solution quality, as longer chains increase susceptibility to chain breaks during annealing (Fang and Warburton, 2020). The dominant practical tool is the `minorminer` heuristic (Cai et al., 2014), complemented by polynomial-time clique embedding for fully connected graphs (Boothby et al., 2016) and various alternatives including layout-aware (Zbinden et al., 2020), simulated-annealing-based (Sugie et al., 2021), and reinforcement-learning-based approaches (Nembrini et al., 2025). While effective, these heuristics share a critical limitation: they provide neither optimality guarantees nor certificates of infeasibility. Consequently, a failed embedding attempt cannot distinguish between instances that are truly infeasible and those that merely fall outside the scope of the heuristic algorithm. This ambiguity creates a costly bottleneck in QA workflows, where quantum processing unit (QPU) access is limited and expensive. While exact methods based on algebraic geometry (Dridi et al., 2018) and integer programming (Bernal et al., 2020) can certify infeasibility, they remain tractable only for small instances. Our work bridges this gap: we use exact solvers to generate provably labeled training data for both feasible embeddings and infeasibility certificates, then train an LLM to generalize these patterns, enabling fast and reliable infeasibility screening at scale.

3 Methodology

This section describes the proposed framework and its core components. Figure 1 illustrates the architectural overview of the system. We first present the dataset construction methodology in Section 3.1. We then detail the supervised fine tuning process in Section 3.2. Finally, we discuss inference and post processing in Section 3.3.

3.1 Mathematical programming for dataset generation

Training the LLM requires high-quality ground truth labels for each problem instance. Mathematical programming is a standard approach for combinatorial optimization because it provides optimality guarantees or certificates of infeasibility when processed by solvers such as Gurobi or OR-Tools. To facilitate this, we provide a formal problem definition and the mathematical programming formulation for the tasks considered in this study.

3.1.1 Mathematical programming formulation for minor-embedding

In this section, we present a new mathematical programming formulation for the minor-embedding problem, which we use to construct ground-truth solutions for dataset instances. In addition, we propose a property that enables fast infeasibility detection for a subset of instances.

We consider the minor-embedding of a problem graph $P = (V_P, E_P)$ into a hardware graph $G = (V_G, E_G)$. A minor-embedding assigns to each problem vertex $i \in V_P$ a connected set of hardware vertices (called a *chain*) such that chains assigned to distinct problem vertices are pairwise disjoint, and every problem edge is mapped to at least one hardware edge between the corresponding chains.

Admissible chains. As noted in Fang and Warburton (2020); Bernal et al. (2020), longer chains are more susceptible to chain breaks; therefore, limiting chain size is good practice. We exploit this observation to derive a compact integer programming model. Specifically, we assume that a family $\mathcal{C} \subseteq 2^{V_G}$ of admissible connected chains in G has been enumerated in advance, optionally with a size bound $|C| \leq L$. For each chain $C \in \mathcal{C}$, let $s(C) = |C|$, and define $\Gamma(C) := \{D \in \mathcal{C} : \exists \{u, v\} \in E_G \text{ with } u \in C,\ v \in D\}$, the set of chains adjacent to C in the hardware graph. Let $A_P := \{(i, k), (k, i) : \{i, k\} \in E_P\}$ denote the directed arc set induced by the edges of graph P.

Decision variables. For each problem vertex $i \in V_P$ and chain $C \in \mathcal{C}$, let $x_{iC} \in \{0, 1\}$ indicate whether vertex i is assigned to chain C.

Formulation. The minor-embedding problem is to minimize (1) subject to (2)–(4):

$$\min \sum_{i,C} s(C) x_{iC} \tag{1}$$

$$\text{s.t.} \quad \textstyle\sum_C x_{iC} = 1 \qquad \forall i \in V_P \tag{2}$$

$$\textstyle\sum_{i,C:j\in C} x_{iC} \leq 1 \qquad \forall j \in V_G \tag{3}$$

$$x_{iC} \leq \textstyle\sum_{D\in\Gamma(C)} x_{kD} \qquad \forall (i,k) \in A_P, C \in \mathcal{C} \tag{4}$$

Constraints (2) ensure unique chain assignment, (3) enforce disjointness, and (4) maintain adjacency between problem and hardware graphs.

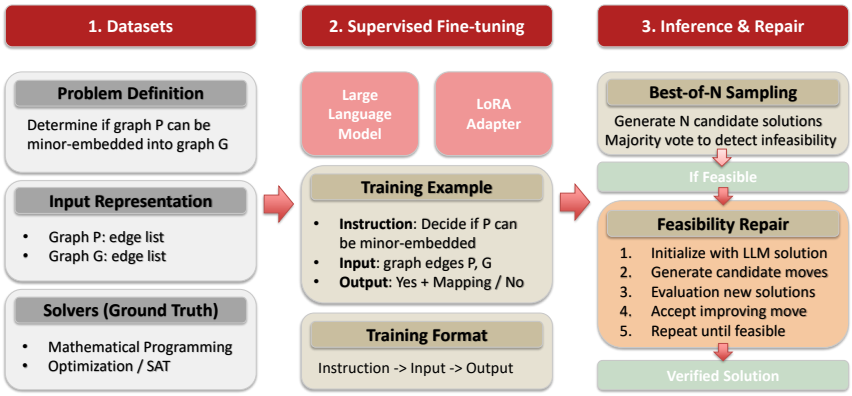


Figure 1: Overview of the proposed infeasibility-aware LLM framework.

For a prescribed chain family $\mathcal{C}$, the formulation uses $O(|V_P||\mathcal{C}|)$ binary variables and $O(|V_P|+|V_G|+|E_P||\mathcal{C}|)$ constraints. When $\mathcal{C}$ consists of all connected chains of size at most L, we have $|\mathcal{C}| = O(|V_G|^L)$ in the worst case, implying a polynomial-size formulation for fixed L. In sparse hardware graphs, the number of connected chains is often much smaller in practice. Nevertheless, the existence of a polynomial-size formulation for fixed L does not make the problem easy: even with the admissible chain family given explicitly, the feasibility version of the problem remains computationally hard. We formalize this below.

Theorem 1. *Consider a problem graph $P = (V_P, E_P)$, a hardware graph $G = (V_G, E_G)$, and an explicitly listed family $\mathcal{C}$ of connected subsets of V_G, with $\{v\} \in \mathcal{C}$ for all $v \in V_G$. Deciding whether there exists an assignment satisfying constraints (2)–(4) (denoted as* MEEC-Feas*) is NP-complete.*

The proof is given in Appendix E.

Zero-phase infeasibility detection. Although the mathematical programming formulation can certify infeasibility and recover an embedding when one exists, solving it can be expensive when generating large benchmark datasets. We therefore apply a *zero-phase infeasibility detection* step before chain enumeration and optimization. Instances rejected at this stage are provably infeasible, while instances that pass are simply not ruled out and are forwarded to the solver. In the following, let $\Delta_H := \max_{v \in V_H} \deg_H(v)$, and assume that every chain has size at most L.

Proposition 1 (Zero-phase infeasibility certificate)**.** *Suppose a problem graph $P = (V_P, E_P)$ admits a minor-embedding into H with maximum chain size L. Then the following necessary conditions hold:*

$$\deg_P(i) \le L(\Delta_H - 2) + 2 \quad \forall i \in V_P, \tag{5}$$

$$\sum_{i \in V_P} s_i^{\min} \le |V_H|, \tag{6}$$

$$|E_P| + \sum_{i \in V_P} (s_i^{\min} - 1) \le |E_H|, \tag{7}$$

where $s_i^{\min} := \max\left(1, \left\lceil \frac{\deg_P(i)-2}{\Delta_H - 2} \right\rceil\right)$ $(\Delta_H \ge 3)$ is a lower bound on the chain size required for problem vertex i. Consequently, violation of any one of (5)–(7) certifies infeasibility.

Proposition 1 follows from the fact that each problem vertex is mapped to a connected chain in the hardware graph and that a chain of size s in a graph of maximum degree Δ_H can support at most $s(\Delta_H - 2) + 2$ external adjacencies (Choi, 2008; Fang and Warburton, 2020). A complete proof is provided in Appendix E.

We use Proposition 1 as an initial infeasibility screening step. We leave more details of implementation in Appendix C.1. We note that a mathematical programming formulation for minor-embedding was previously proposed in Bernal et al. (2020). We present the details of that formulation and compare it with ours in Appendix C.2.

3.1.2 Mathematical programming formulation for k-coloring

We also employ the graph coloring problem as a benchmark for our framework. This problem is particularly instructive because it encompasses two distinct variants: a decision-based version and an optimization-based version. While both versions are computationally difficult (NP-complete/NP-hard) (Karp, 2010), our numerical experiments

demonstrate that instances involving infeasibility are significantly more challenging for LLMs to resolve directly. We leave details of k-coloring problem to Appendix D.

3.2 Supervised fine-tuning

We perform parameter-efficient supervised fine-tuning using Low-Rank Adaptation (LoRA) (Hu et al., 2022) on models from the Llama family (Grattafiori et al., 2024). LoRA adapts the pretrained weight matrix W as $W' = W + BA$, where $A \in \mathbb{R}^{r\times k}$ and $B \in \mathbb{R}^{d\times r}$ are trainable low-rank matrices, W remains frozen, and $r \ll \min(d, k)$ denotes the LoRA rank. This substantially reduces the number of trainable parameters and memory cost. We adopt the standard SFT pipeline: Let x denote the input instance and $y = (y_1, \ldots, y_T)$ denote the target output sequence. We train the model using the standard autoregressive next-token prediction objective $\mathcal{L}_{\text{SFT}} = -\sum_{t=1}^{T} \log p_\theta(y_t \mid x, y_{<t})$, where θ denotes the trainable LoRA parameters, while the pretrained backbone parameters remain fixed. Notably, our key design choice about the target format is that we adopt a unified and compact output representation in which the model directly emits either a machine-verifiable solution certificate for feasible instances or an explicit infeasibility declaration for infeasible instances (see Appendix L for details). No intermediate reasoning or chain-of-thought (CoT) is required. This design encourages concise outputs, enabling faster inference and reducing token usage. The unified format allows the model to jointly learn feasibility recognition and solution generation within a single autoregressive process. To further stabilize learning, we enforce strict consistency in formatting, ordering, and solution representation across the dataset.

3.3 Inference and feasibility repairing

Since autoregressive generation does not guarantee structural correctness, the model may still output invalid solutions even when feasibility is predicted correctly. To address this issue, we propose two mechanisms: Best-of-N inference and feasibility jump.

Best-of-N inference. We adapt Best-of-N strategy (Wang et al., 2022) to select the highest quality solution certificates. At inference time, we sample the fine-tuned model N times to obtain candidates $\hat{y}^{(1)}, \ldots, \hat{y}^{(N)}$. A verification module identifies valid certificates by setting $v_n = 1$ for feasible solutions and $v_n = 0$ otherwise. If at least one valid solution exists, we return the candidate $n^\star$ such that $n^\star \in \arg\min_{n:v_n=1} f(\hat{y}^{(n)}; x)$. If no valid certificate is found ($\sum v_n = 0$), we predict the feasibility status $\hat{c}(x)$ via a majority vote over the feasibility claims $c_n \in \{\texttt{yes}, \texttt{no}\}$ extracted from each sample:

$$\hat{c}(x) = \begin{cases} \texttt{no}, & \text{if } \sum_{n=1}^{N} \mathbf{1}\{c_n = \texttt{no}\} > N/2, \\ \texttt{yes}, & \text{otherwise.} \end{cases}$$

This asymmetric policy prioritizes verified certificates as decisive evidence and utilizes majority voting only as a fallback for cases where verification fails.

Feasibility Jump. Feasibility Jump (FJ) (Luteberget and Sartor, 2023) is a problem-agnostic primal heuristic for Mixed Integer Linear Programs (MILPs) (see Appendix A). While FJ searches directly for solutions from arbitrary assignments, it lacks a mechanism to warm start from high-quality solutions to accelerate convergence. Our framework bridges this gap by using LLMs to provide initial configurations.

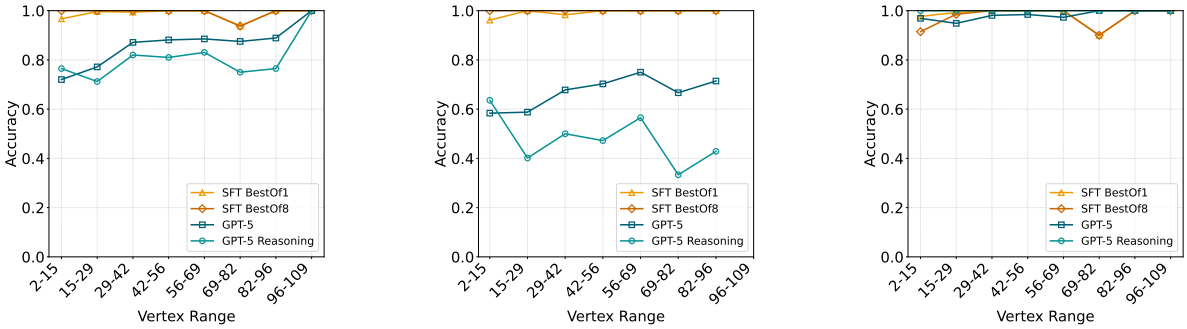
Starting from an initial assignment, FJ iteratively performs “jumps" (optimal single-variable changes) to minimize the weighted violation objective. If FJ hits a local minimum, it increases the weights of violated constraints to escape. We parse LLM outputs and check them for feasibility: feasible solutions are accepted directly, while infeasible ones serve as the starting point for FJ. This combination can be effective because LLM outputs are often structurally informative yet imperfect, precisely the type of input FJ is designed to repair. Appendix F provides a detailed description of the repair procedure and update rules.

4 Experiments

We fine-tune the Llama-3.1-8b model (Grattafiori et al., 2024) as a structured predictor whose outputs are externally verified and, when needed, passed to downstream repair. For each problem type, we construct 10,000 labeled instances. We use balanced datasets with around 50% satisfiable (SAT) and 50% unsatisfiable (UNSAT) instances. Training is performed with Unsloth (Daniel Han and team, 2023), and inference uses top-p sampling with $p = 0.7$. We compare against GPT-5.2, GPT-5.2-Reasoning, DeepSeek-V3.2, Llama-3.3-70B, and Grok-4.1-Fast. We use the same task

Model	Inf. Det.	SAT Acc.	UNSAT Acc.	Time (s)
Deepseek-V3.2	55.1%	74.2%	36.0%	17.37 ± 71.72
Llama3.3-70B	73.0%	79.0%	67.0%	3.72 ± 5.09
Grok-4.1-Fast (Reasoning)	71.2%	42.4%	**100%**	114.40 ± 107.72
GPT5.2	78.5%	61.2%	95.8%	8.67 ± 5.55
GPT5.2-Reasoning	77.5%	55.0%	**100%**	49.68 ± 54.36
Llama-3.1-8b	54.2%	11.6%	96.8%	3.81 ± 43.37
Llama-3.1-8b SFT BestOf1	98.5%	98.0%	99.0%	2.30 ± 2.66
Llama-3.1-8b SFT BestOf8	**99.9%**	**100%**	99.8%	3.86 ± 3.68

Table 1: Evaluation of infeasibility detection accuracy (Inf. Det.), breakdown accuracy for satisfiable (SAT) and unsatisfiable (UNSAT) instances on the minor-embedding problem. Bold values indicate the best performance for each metric.



(a) Overall (b) SAT (c) UNSAT

Figure 2: Accuracy breakdown by the problem graph vertices number. Subplots show results for (a) the full dataset, (b) SAT instances only, and (c) UNSAT instances only.

prompt format for the fine-tuned model and larger general-purpose baselines to isolate the effect of task adaptation under a fixed interface, following same-prompt comparisons between smaller SFT and frontier models in prior LLM-for-optimization work (Jiang et al., 2024). Detailed training hyperparameters and dataset construction procedures are provided in the Appendix G. We also compare token usage and show that our fine-tuned model is more efficient than the benchmark models; see Appendix H. Together with the reasoning-mode baselines in Tables 1 and 2, the token-usage analysis in Figure 5 suggests that simply increasing test-time reasoning or token budget does not remove the constructive failure in our setting (CoT analysis in Appendix K). We therefore use compact, verifiable outputs, while leaving specialized CoT or trajectory-supervised variants to future work.

4.1 Main results

Minor-embedding. Table 1 and Figure 2 summarize the minor-embedding results. Under the same prompt format and verification protocol, our fine-tuned model achieves substantially higher overall accuracy than general-purpose baselines, while maintaining balanced performance across SAT and UNSAT instances. LLAMA-8B SFT already achieves 98.5% overall accuracy with Best-of-1, including 98.0% SAT accuracy and 99.0% UNSAT accuracy. Best-of-8 further improves performance to 99.9% overall accuracy, with 100% SAT accuracy and 99.8% UNSAT accuracy.

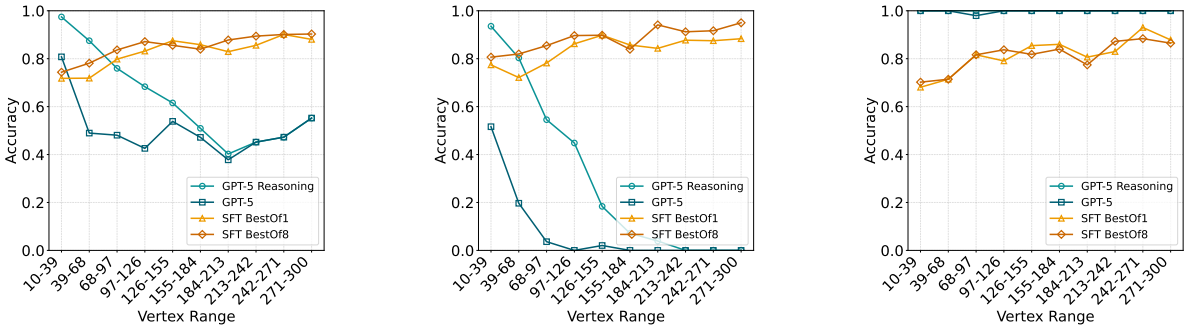
In contrast, the frontier models are much more asymmetric. GPT-5.2 and GPT-5.2-Reasoning perform well on UNSAT instances (95.8% and 100%), but are much weaker on SAT instances (61.2% and 55.0%), indicating a bias toward predicting infeasibility rather than constructing valid embeddings. Figure 2 shows that this advantage remains consistent across instance sizes.

Graph coloring. Table 2 and Figure 3 summarize the performance on the 3-coloring problem.

Table 2 and Figure 3 report the corresponding results on 3-coloring. The same qualitative pattern appears. GPT-5.2 and GPT-5.2-Reasoning again achieve near-perfect performance on UNSAT instances, but their SAT accuracy collapses to 5.89% and 28.33%, respectively. This produces high apparent confidence on infeasibility detection

Model	Inf. Det.	SAT Acc.	UNSAT Acc.	Time (s)
Deepseek-V3.2	48.1%	26.9%	71.7%	11.75 ± 56.95
Llama3.3-70B	47.7%	1.3%	99.2%	1.08 ± 0.92
Grok-4.1-Fast (Reasoning)	60.0%	23.5%	99.2%	111.42 ± 61.20
GPT5.2	50.4%	5.89%	99.79%	7.92 ± 10.14
GPT5.2-Reasoning	62.3%	28.33%	**100%**	42.98 ± 55.19
Llama-3.1-8b	40.6%	41.9%	39.3%	92.06 ± 193.35
Llama-3.1-8b SFT BestOf1	83.1%	83.84%	82.28%	11.22 ± 11.58
Llama-3.1-8b SFT BestOf8	**89.2%**	**95.82%**	81.86%	16.55 ± 12.58

Table 2: Evaluation of infeasibility detection (Inf. Det.), breakdown accuracy for satisfiable (SAT) and unsatisfiable (UNSAT) instances on the 3-coloring problem. Bold values indicate the best performance for each metric.



(a) Overall (b) SAT (c) UNSAT

Figure 3: Accuracy breakdown by instance size. Subplots show results for (a) the full dataset, (b) SAT instances only, and (c) UNSAT instances only.

but poor balanced performance overall.

In contrast, our fine-tuned model remains much more symmetric across the two classes. Best-of-1 already reaches 83.1% infeasibility detection, with 83.84% SAT accuracy and 82.28% UNSAT accuracy, while Best-of-8 improves overall infeasibility detection to 89.2% and raises SAT accuracy to 95.82%. Although UNSAT accuracy is slightly lower than that of GPT-5.2-Reasoning, the resulting performance is far more balanced and therefore much more useful in practice. Figure 3 shows that this advantage is especially pronounced as the instance size grows: GPT-5.2 increasingly defaults to negative answers on larger SAT instances, while our fine-tuned model remains comparatively stable.

4.2 Warm-starting feasibility jump

Next, we evaluate whether LLM outputs can serve as effective initializations for downstream local search. We use Feasibility Jump with a 30-seconds time limit and evaluate only on SAT instances. If a model predicts UNSAT or fails to return a valid structured output, we fall back to a fixed zero initialization. Since GPT-5.2 and GPT-5.2-Reasoning are the strongest non-fine-tuned baselines in infeasibility detection, we restrict the warm-start comparison to these models. To ensure consistency, we perform 5 trials for each method and instance.

Minor-embedding. Table 3 shows that better LLM-generated starting points translate into better downstream FJ performance. Among all methods, *Llama-3.1-8b SFT BestOf8* produces the lowest average initialization error (13.58), the highest FJ success rate (91.40%), the lowest final optimality gap (10.00%), and the fastest convergence time (4.06 ± 9.20 seconds). *GPT-5.2-Reasoning* achieves the highest rate of directly feasible SAT outputs before repair (31.2%), but its downstream optimization performance remains weaker than the SFT model once initialization quality and final gap are taken into account.

3-coloring. The same pattern appears in Table 4. Although direct feasible outputs remain rare for all models, *Llama-3.1-8b SFT BestOf8* produces the best starting points overall, with the lowest average initialization error (12.30), the highest FJ success rate (98.56%), and the fastest convergence time (0.96 ± 3.98 seconds). This supports the broader claim of the paper: even when the model is not

Model	SAT Sol. Feas.	Avg. Error	FJ Succ.	FJ time (s)	Gap↓
Random	N/A	N/A	87.16%	5.05 ± 10.29	16.73%
GPT5.2	8.6%	20.64	87.32%	4.99 ± 10.20	18.78%
GPT5.2-Reasoning	**31.2%**	14.33	87.76%	4.86 ± 10.12	15.10%
Llama-3.1-8b SFT BestOf1	11.2%	15.20	87.32%	4.97 ± 10.25	16.33%
Llama-3.1-8b SFT BestOf8	23.4%	**13.58**	**91.40%**	4.06 ± 9.20	**10.00%**

Table 3: Feasibility Jump results for minor-embedding. SAT Sol. Feas. is the LLM's solution feasibility rate specifically for the satisfiable (SAT) instances, while Avg. Error measures intial solution's constraint violations. These metrics are N/A for the Random baseline as it is not LLM warm-started. FJ Succ. and FJ time denote success rate and runtime within 30s; Gap represents the relative objective gap ((method obj − best known obj)/method obj).

Model	SAT Sol. Feas.	Avg. Error	FJ Succ.	FJ time (s)
Random	N/A	N/A	94.98%	2.12 ± 6.83
GPT5.2	4.37%	30.14	94.71%	2.21 ± 7.01
GPT5.2-Reasoning	**24.33%**	12.38	95.25%	2.08 ± 6.70
Llama-3.1-8b SFT BestOf1	2.28%	15.51	94.80%	2.13 ± 6.87
Llama-3.1-8b SFT BestOf8	6.84%	**12.30**	**98.56%**	**0.96 ± 3.98**

Table 4: Feasibility Jump Results of 3-coloring

Method	SAT Succ.	Gap↓	UNSAT Succ.	Time (s)
Gurobi	54.6%	6.23%	17.2%	47.02
Gurobi NoRel Heuristic	72.8%	5.76%	N/A	25.32
Llama-3.1-8b SFT BestOf8 + FJ	93.8 %	8.02%	99.8%	10.22

Table 6: Comparison with Gurobi. Llama-3.1-8b time include LLM generation time + FJ time.

Problem	Inf. Det.	SAT Acc.	UNSAT Acc.
3-coloring	88.00%	94.40%	81.60%
minor-embedding	87.70%	99.8%	75.6%

Table 5: Out of distribution performance.

fully correct on its own, its structured predictions can still provide useful guidance for conventional optimization algorithms. Appendix D.3 provides an ablation on infeasibility-aware supervision: Removing infeasible instances from the framework shows that generating low-violation solutions is easier than making reliable infeasibility decisions.

4.3 Out-of-distribution performance

Table 5 reports OOD generalization under two shifts: graph coloring uses two held-out problem-graph families, and minor-embedding combines held-out Pegasus hardware with SBM problem graphs (Appendix G). The model remains strong, with 88.0% and 87.7% infeasibility detection on 3-coloring and minor-embedding, respectively. SAT accuracy stays high (94.4% and 99.8%), while UNSAT accuracy drops but remains nontrivial, indicating transfer beyond the training distributions.

4.4 Comparison with MILP solver

We compare against model-then-solve baselines, where an LLM generates the optimization model and solver code (Huang et al., 2025; Jiang et al., 2024; AhmadiTeshnizi et al., 2024); details are in Appendix M. We run Gurobi default mode for 60 seconds and NoRel for 30 seconds: default mode can certify UNSAT instances, while NoRel only searches for feasible solutions. Table 6 reports SAT success, relative gap, UNSAT certification when available, and runtime. Table 6 shows that our SFT+FJ pipeline has the highest SAT and UNSAT success and the shortest runtime. Gurobi attains slightly smaller gaps on the SAT instances it solves, but with much lower coverage.

5 Conclusion

This paper introduces an infeasibility-aware LLM framework for combinatorial optimization, combining certifiable data construction, supervised fine-tuning, and LLM-guided warm starts. Across minor-embedding and graph coloring, the fine-tuned 8B model substantially improves infeasibility detection and provides useful initializations for Feasibility Jump.

Limitations

While our framework successfully integrates infeasibility detection into LLM-based combinatorial optimization, we acknowledge several limitations. First, our current evaluation is restricted to two specific problem domains (minor-embedding and k-coloring). Extending this methodology to a broader range of heavily constrained combinatorial optimization problems where infeasibility is a critical concern remains an important next step. Second, we did not explicitly explore specialized CoT prompting for infeasibility detection. Designing highly specialized reasoning trajectories for feasibility screening is a promising direction for future research.

References

Ali AhmadiTeshnizi, Wenzhi Gao, Herman Brunborg, Shayan Talaei, Connor Lawless, and Madeleine Udell. 2024. OptiMUS-0.3: Using large language models to model and solve optimization problems at scale. *arXiv preprint arXiv:2407.19633*.

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. 2023. OptiMUS: Optimization modeling using MIP solvers and large language models. *arXiv preprint arXiv:2310.06116*.

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

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. 2021. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421.

David E Bernal, Kyle EC Booth, Raouf Dridi, Hedayat Alghassi, Sridhar Tayur, and Davide Venturelli. 2020. Integer programming techniques for minor-embedding in quantum annealers. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 112–129. Springer.

Timo Berthold. 2013. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614.

Tomas Boothby, Andrew D King, and Aidan Roy. 2016. Fast clique minor generation in Chimera qubit connectivity graphs. *Quantum Information Processing*, 15(1):495–508.

Jun Cai, William G Macready, and Aidan Roy. 2014. A practical heuristic for finding graph minors. *arXiv preprint arXiv:1406.2741*.

Hao Chen, Gonzalo E Constante-Flores, and Can Li. 2024. Diagnosing infeasible optimization problems using large language models. *INFOR: Information Systems and Operational Research*, 62(4):573–587.

Vicky Choi. 2008. Minor-embedding in adiabatic quantum computation: I. The parameter setting problem. *Quantum Information Processing*, 7(5):193–209.

Vicky Choi. 2011. Minor-embedding in adiabatic quantum computation: II. Minor-universal graph design. *Quantum Information Processing*, 10(3):343–353.

D-Wave Systems Inc. 2025. dwave-networkx: A networkx extension providing graphs and algorithms relevant to working with the d- wave system. `https://github.com/dwavesystems/dwave-networkx`. Version 0.8.18; accessed 2026-03-28.

Francesca Da Ros, Michael Soprano, Luca Di Gaspero, and Kevin Roitero. 2025. Large language models for combinatorial optimization: A systematic review. *ACM Computing Surveys*.

Michael Han Daniel Han and Unsloth team. 2023. Unsloth.

Emilie Danna, Edward Rothberg, and Claude Le Pape. 2005. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90.

Raouf Dridi, Hedayat Alghassi, and Sridhar Tayur. 2018. A novel algebraic geometry compiling framework for adiabatic quantum computations. *arXiv preprint arXiv:1810.01440*.

P ERDdS and A R&wi. 1959. On random graphs i. *Publ. math. debrecen*, 6(290-297):18.

Yan-Long Fang and P A Warburton. 2020. Minimizing minor embedding energy: an application in quantum annealing. *Quantum Information Processing*, 19:191.

Matteo Fischetti, Fred Glover, and Andrea Lodi. 2005. The feasibility pump. *Mathematical Programming*, 104(1):91–104.

Matteo Fischetti and Andrea Lodi. 2003. Local branching. *Mathematical programming*, 98(1):23–47.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Gurobi Optimization, LLC. 2026. Gurobi Optimizer Reference Manual.

Aric Hagberg, Pieter J Swart, and Daniel A Schult. 2007. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Laboratory (LANL).

Rishi Hazra, Gabriele Venturato, Pedro Zuidberg Dos Martires, and Luc De Raedt. 2025. Have large language models learned to reason? a characterization via 3-sat phase transition. *arXiv preprint arXiv:2504.03930*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. 2025. Orlm: A customizable framework in training large models for automated optimization modeling. *Operations Research*.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. 2024. Llmopt: Learning to define and solve general optimization problems from scratch. *arXiv preprint arXiv:2410.13213*.

Xia Jiang, Yaoxin Wu, Minshuo Li, Zhiguang Cao, and Yingqian Zhang. 2025. Large language models as end-to-end combinatorial optimization solvers. *arXiv preprint arXiv:2509.16865*.

Tadashi Kadowaki and Hidetoshi Nishimori. 1998. Quantum annealing in the transverse Ising model. *Physical Review E*, 58(5):5355–5363.

Richard M Karp. 2010. *Reducibility Among Combinatorial Problems*, pages 219–241. Springer.

Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. 2024. Large language models as evolutionary optimizers. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE.

Elisabeth Lobe and Annette Lutz. 2021. Minor embedding in broken Chimera and Pegasus graphs is NP-complete. *arXiv preprint arXiv:2110.08325*.

Andrea Lodi. 2009. Mixed integer programming computation. In *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 619–645. Springer.

Bjørnar Luteberget and Giorgio Sartor. 2023. Feasibility jump: an lp-free lagrangian mip heuristic. *Mathematical Programming Computation*, 15(2):365–388.

Riccardo Nembrini, Maurizio Ferrari Dacrema, and Paolo Cremonesi. 2025. Minor embedding for quantum annealing with reinforcement learning. *arXiv preprint arXiv:2507.16004*.

Leyan Pan, Vijay Ganesh, Jacob Abernethy, Chris Esposo, and Wenke Lee. 2024. Can transformers reason logically? a study in sat solving. *arXiv preprint arXiv:2410.07432*.

Laurent Perron and Vincent Furnon. 2025. Or-tools.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475.

Yuki Sugie, Yuki Yoshida, Normann Mertig, Takashi Takemoto, Hiroshi Teramoto, Atsushi Nakamura, Ichigaku Takigawa, Shin-ichi Minato, Masanao Yamaoka, and Tamiki Komatsuzaki. 2021. Minor-embedding heuristics for large-scale annealing processors with sparse hardware graphs of up to 102,400 nodes. *Soft Computing*, 25(3):1731–1749.

Anja Surina, Amin Mansouri, Lars Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. 2025. Algorithm discovery with llms: Evolutionary search meets reinforcement learning. *arXiv preprint arXiv:2504.05108*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

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

Anjiang Wei, Yuheng Wu, Yingjia Wan, Tarun Suresh, Huanmi Tan, Zhanke Zhou, Sanmi Koyejo, Ke Wang, and Alex Aiken. 2025. Satbench: Benchmarking llms' logical reasoning via automated puzzle generation from sat formulas. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 33820–33837.

Laurence A Wolsey and George L Nemhauser. 2014. *Integer and combinatorial optimization*. John Wiley & Sons.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and 1 others. 2023. Chain-of-experts: When llms meet complex operations research problems. In *The twelfth international conference on learning representations*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. 2024. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2023. Satlm: Satisfiability-aided language models using declarative prompting. *Advances in Neural Information Processing Systems*, 36:45548–45580.

Simon Zbinden, Andreas Bärtschi, Hristo Djidjev, and Stephan Eidenbenz. 2020. Embedding algorithms for quantum annealers with Chimera and Pegasus connection topologies. In *International Conference on High Performance Computing*, pages 187–206. Springer.

Yu Zhang, Hui-Ling Zhen, Zehua Pei, Yingzhao Lian, Lihao Yin, Mingxuan Yuan, and Bei Yu. 2024. Dila: Enhancing llm tool learning with differential logic layer. *arXiv preprint arXiv:2402.11903*.

A Background of Mixed-Integer Linear Programming and Primal Heuristic

Mixed-integer linear programming. Mixed-integer linear programming (MILP) is a standard modeling framework for decision problems with a linear objective and linear constraints, where some decision variables are additionally required to take integer values, often binary values in $\{0, 1\}$ (Wolsey and Nemhauser, 2014; Lodi, 2009). A generic MILP can be written as

$$\min\ c^\top x \quad \text{s.t. } Ax \le b, \ell \le x \le u, x_j \in \mathbb{Z}\ \ \forall j \in I,$$

where the integrality constraints encode discrete decisions such as selection, assignment, activation, or ordering. This makes MILP a flexible framework for many combinatorial optimization problems.

Modern MILP solvers typically rely on linear programming (LP) based branch-and-bound (together with cutting planes, presolve, and other enhancements) (Wolsey and Nemhauser, 2014). The basic idea is to first drop the integrality requirements and solve the resulting linear programming relaxation. If the LP solution already satisfies integrality, then it is an optimal solution of the MILP. Otherwise, the solver branches on a fractional variable, creating smaller subproblems that are explored in a search tree. During this process, the solver maintains the best feasible integer solution found so far, called the *incumbent*, as well as a bound from the remaining unexplored nodes. The gap between these two quantities measures how far the current incumbent may still be from optimality.

Primal heuristics. While branch-and-bound is designed to certify optimality, finding a good feasible solution early is often just as important in practice. This is the role of *primal heuristics*: procedures that attempt to construct feasible integer solutions quickly, without necessarily proving optimality (Fischetti et al., 2005; Berthold, 2013). In a branch-and-bound solver, primal heuristics are used to improve the incumbent, which can immediately strengthen pruning and accelerate the overall search.

Primal heuristics can be very simple or quite sophisticated. Basic examples include rounding or diving heuristics based on the LP relaxation, while more advanced examples include *Feasibility Pump* (Fischetti et al., 2005), *Local Branching* (Fischetti and Lodi, 2003), and *Relaxation Induced Neighborhood Search (RINS)* (Danna et al., 2005). Conceptually, these methods trade exactness for speed: they try to find a feasible solution that is good enough to guide the search, even if they cannot certify that the solution is globally optimal.

B Background of Quantum Annealing and Minor-Embedding

Quantum annealing (QA) is a hardware-based approach for approximately solving combinatorial optimization problems on a quantum processing unit (QPU) (Kadowaki and Nishimori, 1998). In the standard QA setting, the objective is written as an Ising model, or equivalently a quadratically unconstrained binary optimization (QUBO), over variables with pairwise interactions. This induces a logical problem graph $P = (V_P, E_P)$, where each vertex represents a logical variable and each edge represents a nonzero interaction that must be realized on the device. The QPU, however, does not provide arbitrary all-to-all connectivity. Instead, it implements a fixed sparse hardware graph $G = (V_G, E_G)$ whose vertices are physical qubits and whose edges are programmable couplers.

Two representative hardware topologies are shown in Figure 4: Chimera and Pegasus. Pegasus offers higher local connectivity than Chimera, but both remain much sparser than many logical problem graphs of interest. Consequently, the logical graph usually cannot be mapped to the hardware graph directly. One must first construct a *minor-embedding* of P into G before the instance can be executed on the device (Choi, 2008, 2011).

In a minor-embedding, each problem vertex

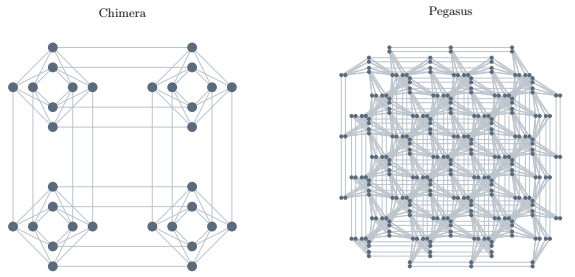


Figure 4: Representative sparse hardware topologies used in quantum annealing: Chimera (left) and Pegasus (right). Pegasus has higher local connectivity than Chimera, but both remain sparse compared with many logical problem graphs. The figure was generated using the open-source `dwave_networkx` package from exact Chimera and Pegasus graph constructions (D-Wave Systems Inc., 2025); Chimera is shown with size parameter $(m = 2, n = 2, t = 4)$ and Pegasus with $m = 4$.

$i \in V_P$ is assigned to a nonempty connected set of hardware vertices, called a *chain*. The qubits in a chain are coupled so that, during the anneal, they act as a single effective logical variable. Chains assigned to different problem vertices must be disjoint, and for every problem edge $\{i, k\} \in E_P$, there must be at least one hardware edge connecting the chain of i to the chain of k. Thus, hardware couplers between chains realize the logical interactions required by the original problem.

This preprocessing step is essential because sparse hardware connectivity is an architectural constraint rather than a modeling choice. The embedding problem is also computationally hard in its own right (Lobe and Lutz, 2021). In addition, embedding quality matters: longer chains use more hardware resources and are more prone to chain breaks during annealing, which can degrade solution quality (Fang and Warburton, 2020). For this reason, the main body focuses not only on whether an embedding exists, but on whether one exists under an explicit chain-length limit.

C Additional Formulation Details for Minor-embedding

C.1 Implementation details of the chain-enumeration formulation

We now provide additional implementation details for the chain-enumeration formulation. Although the formulation (1)–(4) is compact once the admissible chain family is fixed, the practical performance depends strongly on preprocessing, variable pruning, and the manner in which adjacency constraints (4) are enforced.

Chain enumeration and preprocessing. Given a hardware graph $G = (V_G, E_G)$ and a prescribed maximum chain size L, we first enumerate all connected hardware subgraphs $C \subseteq V_G$ with $|C| \le L$. For each enumerated chain $C \in \mathcal{C}$, we precompute its size $s(C)$, its set of internal hardware vertices, the number of internal hardware edges induced by C, and its external neighborhood

$$\partial(C) := \{w \in V_G \backslash C : \exists\, u \in C \text{ with } \{u, w\} \in E_G\}.$$

We also precompute the set of neighboring chains

$$\Gamma(C) := \{D \in \mathcal{C} : \exists\, \{u, v\} \in E_G \text{ with } u \in C,\ v \in D\},$$

which is later used to enforce adjacency compatibility between chains assigned to adjacent problem vertices.

Zero-phase infeasibility screening. Before constructing the optimization model, we apply Proposition 1 as a zero-phase infeasibility screen. Let $\Delta_G := \max_{v \in V_G} \deg_G(v)$, and for each $i \in V_P$ define

$$s_i^{\min} := \max\left(1, \left\lceil \frac{\deg_P(i) - 2}{\Delta_G - 2} \right\rceil\right),$$

which is the minimum chain size required by (5). If $s_i^{\min} > L$ for any i, then no embedding with maximum chain size L can exist. We then test the global vertex and edge bounds

$$\sum_{i \in V_P} s_i^{\min} \le |V_G|, \qquad |E_P| + \sum_{i \in V_P} (s_i^{\min} - 1) \le |E_G|.$$

Violation of any of these three conditions certifies infeasibility, so the instance is discarded before chain enumeration or mixed-integer optimization.

Sparse variable generation. Rather than creating all variables x_{iC} for $i \in V_P$ and $C \in \mathcal{C}$, we generate them selectively. A variable x_{iC} is introduced only if the chain C is large enough to support the problem vertex i, i.e.,

$$s(C) \ge s_i^{\min},$$

and if the chain has sufficient external neighbors to realize all incident problem edges, i.e.,

$$|\partial(C)| \ge \deg_P(i).$$

Equivalently, for each problem vertex i, we define the candidate chain set

$$\mathcal{C}_i := \{C \in \mathcal{C} : s(C) \ge s_i^{\min},\ |\partial(C)| \ge \deg_P(i)\},$$

and create variables only for pairs (i, C) with $C \in \mathcal{C}_i$. If $\mathcal{C}_i = \emptyset$ for some i, the instance is certified infeasible. This filtering step can remove a substantial fraction of candidate variables and significantly reduces model size.

Edge-budget strengthening. In addition, we include a global edge-budget inequality. Let $e(C)$ denote the number of internal hardware edges induced by chain C. Since each selected chain of size $s(C)$ consumes internal hardware edges, and each problem edge requires at least one hardware edge between two selected chains, any feasible embedding must satisfy

$$\sum_{i \in V_P} \sum_{C \in \mathcal{C}_i} e(C) x_{iC} + |E_P| \leq |E_G|.$$

This inequality is a valid strengthening cut that helps eliminate structurally impossible assignments early in the search.

Adjacency enforcement. The adjacency requirements in (4) are enforced via *lazy cuts* in Gurobi. We first solve the model with only (2)–(3). For any incumbent integer solution, let $C_i^\star$ be the selected chain for each problem vertex i. For every problem edge $(i, k) \in A_P$, we check whether $C_i^\star$ and $C_k^\star$ are compatible, i.e., whether $C_k^\star \in \Gamma(C_i^\star)$. If not, then the adjacency condition for (i, k) is violated, and we lazily add the missing constraint

$$x_{i,C_i^\star} \leq \sum_{D \in \Gamma(C_i^\star)} x_{kD}.$$

Hence, adjacency constraints are generated only for chain selections that actually appear in incumbent solutions.

C.2 Alternative formulation for minor-embedding

As an alternative to enumerating admissible chains, one may model the minor-embedding problem directly on the hardware graph. In this approach, each problem vertex is assigned a subset of hardware vertices, and connectivity is enforced dynamically through feasibility cuts rather than by explicit inequalities. Following Bernal et al. (2020), this leads to a decomposition-based formulation in which the master problem enforces assignment, disjointness, and edge realization, while connectivity is checked separately.

Sets. Let $G = (V_G, E_G)$ denote the undirected hardware graph and let $P = (V_P, E_P)$ denote the problem graph.

Decision variables. For each $i \in V_P$ and $j \in V_G$, let $z_{ij} \in \{0, 1\}$ indicate whether hardware vertex j is assigned to the chain of problem vertex i. For each problem edge $\{i, k\} \in E_P$ and hardware edge $\{u, v\} \in E_G$, let $y_{ikuv} \in \{0, 1\}$ indicate whether hardware edge $\{u, v\}$ is selected to realize problem edge $\{i, k\}$. Optionally, for each $i \in V_P$ and $\{u, v\} \in E_G$, let $f_{iuv} \in \{0, 1\}$ indicate whether hardware edge $\{u, v\}$ is used internally within the chain of i.

Master problem. The master problem is

$$\min \sum_{i \in V_P} \sum_{j \in V_G} z_{ij} \tag{8}$$

$$\text{s.t.} \sum_{i \in V_P} z_{ij} \leq 1 \qquad \forall j \in V_G, \tag{9}$$

$$\sum_{j \in V_G} z_{ij} \geq 1 \qquad \forall i \in V_P, \tag{10}$$

$$\sum_{\{u,v\} \in E_G} y_{ikuv} = 1 \quad \forall \{i, k\} \in E_P, \tag{11}$$

$$y_{ikuv} \leq z_{iu} + z_{iv} \qquad \forall \{i, k\} \in E_P,\ \forall \{u, v\} \in E_G, \tag{12}$$

$$y_{ikuv} \leq z_{ku} + z_{kv} \qquad \forall \{i, k\} \in E_P,\ \forall \{u, v\} \in E_G, \tag{13}$$

$$z_{ij} \in \{0, 1\} \qquad \forall i \in V_P,\ \forall j \in V_G, \tag{14}$$

$$y_{ikuv} \in \{0, 1\} \qquad \forall \{i, k\} \in E_P,\ \forall \{u, v\} \in E_G. \tag{15}$$

The objective (8) minimizes the total number of hardware vertices used. Constraints (9) enforce disjointness of chains, constraints (10) require each problem vertex to receive a nonempty chain, and constraints (11)–(13) ensure that each problem edge is realized by one hardware edge connecting the corresponding two chains.

Connectivity cuts. The master problem does not explicitly enforce that the hardware vertices assigned to a problem vertex induce a connected subgraph. Thus, connectivity is imposed dynamically through lazy cuts. For a current incumbent solution z^* and a fixed $i \in V_P$, define

$$\phi(i) := \{j \in V_G : z_{ij}^* = 1\}. \tag{16}$$

If $\phi(i)$ is disconnected in the hardware graph, let C be one connected component of $\phi(i)$ and let

$$N(C) := \{w \in V_G \setminus C : \exists\, u \in C, \{u, w\} \in E_G\} \tag{17}$$

denote the neighbors of C. Choose any $u \in C$ and any $v \in \phi(i) \setminus C$. Then any connected assignment containing both u and v must select at least one hardware vertex in $N(C)$, which yields the valid inequality

$$z_{iu} + z_{iv} - 1 \leq \sum_{w \in N(C)} z_{iw}. \tag{18}$$

At the current disconnected solution, both u and v are selected while no vertex in $N(C)$ is selected, so (18) is violated. The master problem is re-solved with such cuts added until every assigned chain is connected.

Chain-size restrictions. We still impose an upper bound L on the size of each chain. This yields the following constraints

$$\sum_{j \in V_G} z_{ij} \leq L \quad \forall i \in V_P, \tag{19}$$

$$z_{iu} + z_{iv} \leq 1 \quad \forall i \in V_P,\ \forall u, v \in V_G, d_G(u, v) \geq L, \tag{20}$$

where $d_G(u, v)$ denotes the shortest-path distance in G. Indeed, if a connected subgraph contains at most L vertices, then any two vertices in it must be at graph distance at most $L - 1$.

Optional strengthening. The master problem can be strengthened by introducing edge-selection variables f_{iuv} to capture internal edges of a chain. A simple necessary condition for connectivity is that a connected subgraph on t vertices must contain at least $t - 1$ edges. This yields

$$f_{iuv} \leq z_{iu}, \forall i \in V_P,\ \forall \{u, v\} \in E_G, \tag{21}$$

$$f_{iuv} \leq z_{iv}, \forall i \in V_P,\ \forall \{u, v\} \in E_G, \tag{22}$$

$$\sum_{\{u,v\} \in E_G} f_{iuv} \geq \sum_{j \in V_G} z_{ij} - 1, \forall i \in V_P, \tag{23}$$

$$f_{iuv} \in \{0, 1\}, \forall i \in V_P,\ \forall \{u, v\} \in E_G. \tag{24}$$

These inequalities do not by themselves guarantee connectivity, but they tighten the relaxation by ruling out some disconnected assignments.

Formulation size. The master problem contains $|V_P||V_G|$ binary variables z_{ij} and $|E_P||E_G|$ binary variables y_{ikuv}, plus optionally $|V_P||E_G|$ binary variables f_{iuv}. The number of constraints is polynomial in $|V_P|$, $|E_P|$, $|V_G|$, and $|E_G|$, with connectivity enforced iteratively through the feasibility cuts (18).

Comparison to Chain-enumeration formulation We compare the two formulations on instances that are not eliminated by the zero-phase infeasibility screening. The results are reported in Table 7.

Overall, the chain-enumeration formulation is more effective on this test set. It solves more instances within the time limit, detects infeasibility more reliably, and yields smaller final gaps. It also requires less average total runtime. These results support its use as the main formulation for dataset generation.

D Formulation Details for k-coloring

D.1 k-coloring

Given an undirected graph $G = (V, E)$ and a fixed integer $k \geq 1$, the k-coloring feasibility problem asks whether the vertices of G can be assigned at most k colors so that adjacent vertices receive different colors.

Decision variables. For each vertex $i \in V$ and color $c \in K$, let $x_{ic} \in \{0, 1\}$ indicate whether vertex i is assigned color c.

Formulation. The k-coloring feasibility problem can be written as

$$\sum_{c \in K} x_{ic} = 1 \quad \forall i \in V, \tag{25}$$

$$x_{ic} + x_{jc} \leq 1 \quad \forall \{i, j\} \in E,\ \forall c \in K, \tag{26}$$

Constraints (25) ensure that each vertex receives exactly one color, while constraints (26) enforce that adjacent vertices cannot be assigned the same color. Thus, the graph is k-colorable if and only if the above system is feasible.

Note that the 2-coloring problem is equivalent to determining whether a graph is bipartite, which is solvable in linear time. So in this paper we consider the 3-coloring problem.

D.2 min-coloring

Given an undirected graph $G = (V, E)$, the min-coloring problem asks for a proper coloring of the vertices using as few colors as possible. Equivalently, the goal is to determine the chromatic number of G.

Formulation	Solved	UNSAT	Gap↓	Time (s)
Decomposition	65.41%	6.20%	15.15%	42.95
Chain-enumeration	96.40%	100.00%	5.14%	17.88

Table 7: Comparison of the two formulations on 2000 instances with a 60-second Gurobi time limit per instance. The table reports results on the 1100 instances that were solved by at least one method or certified infeasible. *Solved* is the fraction of instances solved within the time limit, *UNSAT* is the fraction of infeasible instances certified infeasible, *Gap* is the mean final relative optimality gap at termination (returned by Gurobi), computed as (upper bound − lower bound)/|upper bound|, and *Time (s)* is the mean total runtime.

Candidate colors. Let $K := \{1, 2, \ldots, |V|\}$ be an upper-bounding set of candidate colors. Since every graph on $|V|$ vertices can be colored with at most $|V|$ colors, this choice is always valid.

Decision variables. For each vertex $i \in V$ and color $c \in K$, let $x_{ic} \in \{0, 1\}$ indicate whether vertex i is assigned color c. In addition, for each color $c \in K$, let $y_c \in \{0, 1\}$ indicate whether color c is used by at least one vertex.

Formulation. The min-coloring problem can be written as

$$\min \sum_{c \in K} y_c \tag{27}$$

subject to

$$\sum_{c \in K} x_{ic} = 1 \qquad \forall i \in V, \tag{28}$$

$$x_{ic} + x_{jc} \leq 1 \qquad \forall \{i, j\} \in E,\ \forall c \in K, \tag{29}$$

$$x_{ic} \leq y_c \qquad \forall i \in V,\ \forall c \in K, \tag{30}$$

Constraints (28) assign exactly one color to each vertex, constraints (29) ensure that adjacent vertices receive different colors, and constraints (30) activate y_c whenever color c is assigned to some vertex. The objective (27) therefore minimizes the number of colors used.

Symmetry breaking. Because colors are interchangeable, the formulation contains substantial symmetry. A common strengthening is to impose

$$y_c \geq y_{c+1} \qquad \forall c = 1, \ldots, |V| - 1, \tag{31}$$

which enforces that lower-indexed colors are used before higher-indexed ones.

D.3 Difficulty of infeasibility vs. optimality

Table 8 presents the results for the min-coloring problem. Unlike the 3-coloring task, this problem does not include ground-truth infeasible instances. Although the min-coloring problem shares constraints with 3-coloring and involves a larger decision space (e.g., determining the optimal number of colors), the model performs substantially better in terms of feasibility rate and edge-violation ratio, even though exact feasibility remains challenging. (Compare to SAT instances feasibility rate in table 4). While techniques like reinforcement learning, as proposed by Jiang et al. (2025), could further enhance the fine-tuned model's performance, we leave such enhancement for future work, as this study mainly focus on infeasibility detection.

E Proofs

E.1 Proof of Theorem 1

Proof. Membership in NP is immediate. Given a binary assignment vector x, one can verify in polynomial time that each problem vertex is assigned exactly one chain, that the selected chains are pairwise disjoint, and that for every problem edge the corresponding selected chains are adjacent in the hardware graph.

To prove NP-hardness, we reduce from *3-Dimensional Matching (3DM)*. An instance of *3DM* consists of three pairwise disjoint sets X, Y, Z with $|X| = |Y| = |Z| = n$, and a set of triples $T \subseteq X \times Y \times Z$. The question is whether there exists a subset $M \subseteq T$ with $|M| = n$ such that no two triples in M share an element.

We now construct an instance of *MEEC-Feas* from (X, Y, Z, T).

Hardware graph. Given an instance (X, Y, Z, T) of *3DM*, we construct a hardware graph $G = (V_G, E_G)$ with three types of vertices.

First, for each part $B \in \{X, Y, Z\}$, create four special vertices

$$a_B,\ b_B^1,\ b_B^2,\ b_B^3,$$

and add all six edges among them. Thus, for each

Model	Feasibility↑	Optimality Gap↓	Error Ratio↓
GPT5.2	22.2%	70.58%	17.04%
Llama-8b SFT BestOf1	10.1%	16.64%	3.90%
Llama-8b SFT BestOf8	24.1%	10.04%	2.06%

Table 8: Evaluation of feasibility and optimality gap (min-coloring). Bold indicates optimal performance per metric. Error Ratio is defined as the percentage of violated edges among all edges.

$B \in \{X, Y, Z\}$, these four vertices induce a complete graph on four vertices, denoted by K_4.

Second, create a corresponding hardware vertex for each element in $X \cup Y \cup Z$. Add edges from the a-vertices to these element-vertices: $\{a_X, x\}$ for all $x \in X$, $\{a_Y, y\}$ for all $y \in Y$, and $\{a_Z, z\}$ for all $z \in Z$.

Finally, for each triple $t = (x, y, z) \in T$, create one additional hardware vertex p_t, and connect it to the three element-vertices appearing in that triple: $\{p_t, x\}, \{p_t, y\}$, and $\{p_t, z\}$. There are no other hardware edges.

Admissible chain family. Let

$$\mathcal{C} := \big\{\{v\} : v \in V_G\big\} \cup \big\{C_t : t \in T\big\},$$

where for each triple $t = (x, y, z) \in T$, we define $C_t := \{p_t, x, y, z\}$. Each C_t is connected in G (inducing a star centered at p_t), and by construction, every singleton chain $\{v\}$ is included in $\mathcal{C}$.

Problem graph. Construct a problem graph $P = (V_P, E_P)$ as follows.

For each part $B \in \{X, Y, Z\}$, create four vertices $u_B^0, u_B^1, u_B^2, u_B^3$ and add all six edges among them, ensuring that each set of four induces a K_4 in P. In addition, create a set of n selector vertices, denoted as $S = \{s_r \mid 1 \le r \le n\}$. For every $s_r \in S$ and each part $B \in \{X, Y, Z\}$, add the edge $\{s_r, u_B^0\}$. There are no other edges in P.

We now show that the constructed *MEEC-Feas* instance is feasible if and only if the original *3DM* instance has a matching of size n. We rely on two key observations: First, each K_4 in the problem graph must be mapped to one of the three K_4 formed by singleton chains in the hardware graph. Second, once established, each selector vertex is forced to choose a chain of the form C_t.

Claim 1. The only collections of four pairwise disjoint admissible chains that are pairwise adjacent are

$$\big\{\{a_B\}, \{b_B^1\}, \{b_B^2\}, \{b_B^3\}\big\}$$

for $B \in \{X, Y, Z\}$.

Proof of Claim 1. If four pairwise disjoint singleton chains are pairwise adjacent, their underlying hardware vertices must form a K_4 in G. By construction, the only K_4 subgraphs of G are the three gadgets corresponding to X, Y, and Z.

It remains to rule out the possibility that one of the four chains is a nonsingleton chain C_t. Suppose $C_t = \{p_t, x, y, z\}$ is selected. Any admissible chain disjoint from C_t and adjacent to C_t must contain a vertex adjacent in G to one of p_t, x, y, z.

Because p_t connects exclusively to x, y, and z, any singleton chain $\{p_t\}$ intersects C_t and is not disjoint. Furthermore, singletons $\{x\}, \{y\}$, and $\{z\}$ also intersect C_t. The only disjoint singleton chains adjacent to C_t are therefore $\{a_X\}, \{a_Y\}, \{a_Z\}$ (via the edges $\{a_X, x\}, \{a_Y, y\}, \{a_Z, z\}$), along with any singleton $\{p_{t'}\}$ where t' shares an element with t. However, the vertices a_X, a_Y, and a_Z are pairwise nonadjacent in G, and all p-vertices are pairwise nonadjacent to each other. Consequently, it is impossible to find three mutually adjacent chains among them. Hence, C_t cannot belong to any family of four pairwise disjoint, pairwise adjacent chains. $\diamond$

Since each set $\{u_B^0, u_B^1, u_B^2, u_B^3\}$ induces a K_4 in P, Claim 1 implies that in any feasible solution these four problem vertices must be assigned to one of the three hardware K_4 gadgets. Because these gadgets are pairwise disjoint and chain disjointness is enforced globally, the three K_4 subgraphs of P must occupy the three hardware gadgets bijectively.

Claim 2. Within each hardware K_4 gadget, the vertex u_B^0 must be assigned to the singleton chain $\{a_{B'}\}$ of that gadget (for some $B' \in \{X, Y, Z\}$), and not to one of the chains $\{b_{B'}^1\}, \{b_{B'}^2\}, \{b_{B'}^3\}$.

Proof of Claim 2. The vertex u_B^0 is adjacent in P not only to the other three vertices of its K_4 but also to all selector vertices in S. If u_B^0 were assigned to, say, $\{b_{B'}^1\}$, then every selector s_r would have to be assigned to a chain adjacent to $\{b_{B'}^1\}$. But in G, the only vertices adjacent to $b_{B'}^1$ are the other three vertices of the same hardware K_4 gadget, namely

$a_{B'}, b^2_{B'}, b^3_{B'}$. Those singleton chains are already occupied by the four vertices of that problem K_4, and no nonsingleton chain in $\mathcal{C}$ contains any of these gadget vertices. Hence no selector vertices can be assigned a chain adjacent to $\{b^1_{B'}\}$, contradiction. The same argument applies to $\{b^2_{B'}\}$ and $\{b^3_{B'}\}$. Therefore, u^0_B must be assigned to $\{a_{B'}\}$. $\diamond$

Up to a permutation of the labels X, Y, and Z, it follows that the three distinguished vertices u^0_X, u^0_Y, u^0_Z are strictly assigned to $\{a_X\}, \{a_Y\}$, and $\{a_Z\}$.

Now consider any selector vertex s_r. Since s_r is adjacent in P to all three u^0 vertices, its assigned chain must be adjacent to all three singleton chains $\{a_X\}, \{a_Y\}, \{a_Z\}$. We evaluate the admissible chains to find which possess this property:

- A singleton chain $\{b^j_B\}$ is adjacent only to the other three vertices in its own K_4, so it cannot be adjacent to all three of $\{a_X\}, \{a_Y\}, \{a_Z\}$.
- A singleton chain $\{a_B\}$ is already used by one of the vertices u^0_X, u^0_Y, u^0_Z, and is therefore unavailable by disjointness.
- A singleton chain $\{x\}$ with $x \in X$ is adjacent only to $\{a_X\}$ among these three singleton chains. Similarly, $\{y\}$ with $y \in Y$ is adjacent only to $\{a_Y\}$, and $\{z\}$ with $z \in Z$ is adjacent only to $\{a_Z\}$.
- A singleton chain $\{p_t\}$ is adjacent only to the three element-vertices of the triple t, and so it is adjacent to none of $\{a_X\}, \{a_Y\}, \{a_Z\}$.
- For a triple $t = (x, y, z)$, the chain $C_t = \{p_t, x, y, z\}$ is adjacent to all three singleton chains $\{a_X\}, \{a_Y\}, \{a_Z\}$ through the edges $\{a_X, x\}$, $\{a_Y, y\}$, and $\{a_Z, z\}$.

Therefore, every selector vertex in S must be assigned to a chain of the form C_t.

Since each p_t is unique to its triple, two chains C_t and $C_{t'}$ intersect if and only if the corresponding triples t and t' share an element of $X \cup Y \cup Z$. Hence, choosing n pairwise disjoint selector chains is equivalent to choosing n triples from T that share no element, i.e., to a 3-dimensional matching of size n.

MEEC-Feas⇒3DM. If the constructed *MEEC-Feas* instance is feasible, then the n selector vertices determine n pairwise disjoint chains C_t, and hence n triples in T that are pairwise disjoint. Therefore the original *3DM* instance has a matching of size n.

MEEC-Feas⇐3DM. Conversely, suppose the *3DM* instance has a matching $M = \{t_r \mid 1 \le r \le n\} \subseteq T$ of size n. Assign each of the three K_4 subgraphs of P to one of the three hardware K_4 gadgets (e.g., $u^0_B \mapsto \{a_B\}$ and $u^j_B \mapsto \{b^j_B\}$). Then, assign $s_r \mapsto C_{t_r}$ for all $1 \le r \le n$. Because the matching triples are pairwise disjoint, the selected chains $\{C_{t_r} \mid 1 \le r \le n\}$ are pairwise disjoint. Furthermore, each C_{t_r} connects to the three singleton chains $\{a_X\}, \{a_Y\}$, and $\{a_Z\}$, thereby realizing all problem edges $\{s_r, u^0_B\}$ for $B \in \{X, Y, Z\}$. Thus all constraints (2)–(4) are satisfied.

The construction is polynomial in the size of the *3DM* instance. Moreover, all admissible nonsingleton chains introduced by the reduction have cardinality 4. Therefore, NP-hardness already holds in this restricted special case, and therefore *MEEC-Feas* is NP-complete. $\square$

E.2 Proof of the zero-phase infeasibility certificate

For each problem vertex $i \in V_P$, let $C_i \subseteq V_H$ denote its chain, and let $s_i := |C_i|$. Since C_i is connected, the subgraph induced by C_i contains at least $s_i - 1$ internal hardware edges.

Proof. Fix any problem vertex $i \in V_P$ and let $s_i := |C_i|$. Since each hardware vertex has degree at most Δ_H, the total degree contributed by the vertices of C_i is at most $s_i \Delta_H$. Because C_i is connected, it contains at least $s_i - 1$ internal hardware edges, and each such edge consumes two units of degree. Therefore, the number of hardware edges leaving C_i is at most

$$s_i \Delta_H - 2(s_i - 1) = s_i(\Delta_H - 2) + 2.$$

Each neighbor of i in the problem graph must be realized by at least one hardware edge leaving C_i, since chains assigned to distinct problem vertices are disjoint and every problem edge must be witnessed by a hardware edge between the corresponding chains. Hence

$$\deg_P(i) \le s_i(\Delta_H - 2) + 2 \le L(\Delta_H - 2) + 2,$$

which proves (5). Rearranging this inequality gives the lower bound on the chain size required for vertex i.

Because the chains are pairwise disjoint, any embedding uses at least $s_i^{\min}$ hardware vertices for

each $i \in V_P$. Summing over all problem vertices yields

$$\sum_{i \in V_P} s_i^{\min} \le |V_H|,$$

which proves (6).

For (7), note that each connected chain C_i with $|C_i| = s_i$ contains at least $s_i - 1$ internal hardware edges. In addition, each problem edge in E_P requires at least one hardware edge between two distinct chains. Internal chain edges lie entirely within a single chain, whereas witness edges for problem edges connect two distinct chains; therefore these two classes of hardware edges are disjoint. It follows that any embedding must use at least

$$|E_P| + \sum_{i \in V_P} (s_i - 1)$$

hardware edges, and hence at least

$$|E_P| + \sum_{i \in V_P} (s_i^{\min} - 1).$$

Therefore,

$$|E_P| + \sum_{i \in V_P} (s_i^{\min} - 1) \le |E_H|,$$

which proves (7). □

F Details of Feasibility Jump

For completeness, we summarize the Feasibility Jump (FJ) repair procedure used in our framework. Let $N := \{1, \dots, n\}$, $M := \{1, \dots, m\}$, denote the index sets of variables and constraints, respectively. We consider the linear feasibility problem

$$\text{find } x \in \mathcal{X} \quad \text{s.t.} \quad a_i^\top x \le b_i \qquad \forall i \in M,$$

where $\mathcal{X}$ encodes bounds and integrality restrictions. FJ works with the weighted infeasibility objective

$$F_w(x) = \sum_{i \in M} w_i f_i(x), f_i(x) = \max\{0,\, a_i^\top x - b_i\},$$

where $w_i \ge 0$ is the current weight of constraint i. Given a current assignment $\bar{x}$, FJ evaluates for each variable x_j the best one-variable move with all other variables fixed. Let $G_j(t \mid \bar{x}_{k \ne j})$ denote the weighted infeasibility obtained by setting $x_j = t$ while keeping all other variables fixed. The *jump value* of variable j is

$$\begin{aligned} v_j &:= \mathrm{Jump}_j(\bar{x}_{k \ne j}) \\ &= \operatorname*{argmin}_{t \in [l_j, u_j] \cap D_j,\ t \ne \bar{x}_j} G_j(t \mid \bar{x}_{k \ne j}), \end{aligned}$$

where $D_j = \mathbb{Z}$ for integer variables and $D_j = \mathbb{R}$ otherwise. Its score is

$$s_j = G_j(\bar{x}_j \mid \bar{x}_{k \ne j}) - G_j(v_j \mid \bar{x}_{k \ne j}),$$

which measures the reduction in weighted infeasibility obtained by moving x_j to its jump value. The set of promising variables is $P := \{j \in N : s_j > 0\}$. If $P = \varnothing$ and the current solution is still infeasible, FJ regards the current point as a local minimum. It then increases the weights of violated constraints, updates the affected scores, and continues the search from the reweighted objective.

The pseudocode is given in Algorithm 1.

Phase two of feasibility jump. After a feasible solution has been found, we run a second local-improvement phase that minimizes the number of used hardware vertices while preserving feasibility. Let $\phi(x)$ denote the total number of hardware vertices used by the current embedding. Phase 2 considers only deletion moves: for each chain of size at least two, it proposes removing a small number of hardware vertices from that chain. A candidate move is accepted only if the updated solution remains feasible and strictly decreases $\phi(x)$. In our implementation, at most 60 candidate deletions are generated per iteration, at most 20 are probed, and the search terminates after a fixed number of consecutive non-improving iterations. The best feasible solution found during this phase is returned. The pseudocode is given in Algorithm 2. In the experiments reported in Table 3, Phase 2 is executed as long as the total runtime remains within the 30-second limit, with the patience parameter set to 40.

G Experiment Setting

This section provides details of dataset generation and training hyperparameters. All our codes, data and trained model weights will be available upon acceptance.

G.1 Dataset generation

For graph generation, we use several standard graph families chosen to span different structural regimes. Specifically, we use *Erdős–Rényi*

Algorithm 1: Feasibility Jump Repair from an LLM Candidate

Input : LLM candidate $\hat{x}$, iteration limit T
Output : A feasible solution $x \in \mathcal{X}$ if found; otherwise failure

```
Parse x̂ into an initial assignment x ∈ 𝒳
Initialize w_i ← 1 for all i ∈ M
Compute v_j and s_j for all j ∈ N
P ← {j ∈ N : s_j > 0}
x^best ← x
for t ← 1 to T do
    if F_w(x) = 0 then
        return x
    end
    if P ≠ ∅ then
        Choose a subset P* ⊆ P with at most 25 variables
        Select j* ∈ arg max_{j∈P*} s_j
    else
        U ← {i ∈ M : f_i(x) > 0}
        foreach i ∈ U do
            w_i ← w_i + 1
        end
        Recompute v_j and s_j for all j ∈ N with a_ij ≠ 0 for some i ∈ U
        P ← {j ∈ N : s_j > 0}
        Choose i* uniformly at random from U
        Select j* ∈ arg max_{j∈N: a_{i*j}≠0} s_j
    end
    x_{j*} ← v_{j*}
    if F_w(x) < F_w(x^best) then
        x^best ← x
    end
    Recompute v_{j*} and s_{j*}
    Recompute v_j and s_j for all j ≠ j* sharing a constraint with j*
    P ← {j ∈ N : s_j > 0}
end
if F_w(x^best) = 0 then
    return x^best
end
return failure
```

graphs (ERDdS and R&wi, 1959), *Barabási–Albert* graphs (Barabási and Albert, 1999), *Watts–Strogatz* graphs (Watts and Strogatz, 1998), random regular graphs, Delaunay graphs, and stochastic block models. We generate *Erdős–Rényi*, *Barabási–Albert*, *Watts–Strogatz*, and random regular graphs using standard NetworkX implementations (Hagberg et al., 2007).

Graph coloring. For graph coloring, we generate in-distribution instances from *Erdős–Rényi* graphs. Specifically, we generate $G(n, p)$ instances with $n \in [10, 300]$ and set $p = d/(n-1)$, where the target average degree satisfies $d \in [3.0, 5.2]$. After generation, isolated vertices are removed. To obtain ground-truth labels for 3-coloring, we solve each instance with Google OR-Tools CP-SAT and label it as SAT or UNSAT; for SAT instances, we also extract an explicit coloring. For min-coloring, we solve the optimization problem with Gurobi under a 60-second time limit.

For OOD evaluation, we generate instances from two held-out graph families: *Barabási–Albert* and *Watts–Strogatz*. We again use $n \in [10, 300]$ and

Algorithm 2: Phase 2 of Feasibility Jump for minor-embedding

```
Input  : A feasible assignment x, patience threshold P_max
Output : A feasible assignment with improved objective, if found
φ(x) ← total number of used hardware vertices
x^best ← x
φ^best ← φ(x)
p ← 0
while p < P_max do
    Construct candidate deletion moves M as follows:
        For each problem vertex, if its chain has size at least two, sample up to four hardware
          vertices from that chain and create moves that unassign those vertices
        Stop once |M| = 60
    if M = ∅ then
        return x^best
    end
    Sample a probe set M* ⊆ M with |M*| ≤ 20
    improved ← false
    foreach m ∈ M* do
        Apply m to obtain trial solution x'
        Re-evaluate all feasibility constraints for x'
        if x' is feasible and φ(x') < φ(x) then
            x ← x'
            if φ(x) < φ^best then
                x^best ← x
                φ^best ← φ(x)
            end
            p ← 0
            improved ← true
            break
        end
    end
    if improved = false then
        p ← p + 1
    end
end
return x^best
```

target average degree $d \in [3.0, 5.2]$. To ensure sufficient UNSAT coverage, we implant a K_4 clique into a subset of instances. For *Barabási–Albert* graphs, we set $m = \mathrm{round}(\min(d/2, 8))$ with $m \geq 1$. For *Watts–Strogatz* graphs, we set $k = \mathrm{round}(d)$ rounded to the nearest even integer with $k \geq 4$, and sample the rewiring probability as $\beta \sim \mathrm{Uniform}[0.2, 0.7]$.

Minor-embedding. For minor embedding, problem graphs are sampled from four families: *Erdős–Rényi*, *Barabási–Albert*, *Watts–Strogatz*, and random regular. Specifically, *Erdős–Rényi* source graphs use edge probability $p \in [0.1, 0.6]$; *Barabási–Albert* graphs use attachment parameter $m \in [3, \min(8, n-1)]$; *Watts–Strogatz* graphs use $k \in \{2, 4, 6, 8, 10\}$ with k clamped to $n-1$ and forced even, and rewiring probability $p \in [0.05, 0.3]$; and random regular graphs use degree $d \in \{4, 6, 8, 10\}$ with parity correction when needed.

Hardware graphs are drawn from a 50%/50% mixture of Chimera and random topologies. For Chimera, we fix $t = 4$ and vary $(m, n) \in \{1, 2, 3, 4\}^2$, yielding 16 hardware shapes. For random hardware, we sample one family from

Method	Inf. Cert.	Feas.	Coverage	Opt. Cert.	Avg. Gap ↓
Gurobi+Zero	4,816	5,281	10,097 (50.5%)	82.0%	0.47%
minorminer	N/A	6,947	6,947 (34.7%)	N/A	2.21%

Table 9: Comparison of *Gurobi+Zero-Phase Infeasibility Detection* and *minorminer* on 20,000 minor-embedding instances. *Inf. Cert.* is the number of instances certified infeasible. *Feas.* is the number of feasible solutions found. *Coverage* counts all instances resolved by the method, i.e., infeasibility certificates plus feasible solutions. *Opt. Cert.* reports the fraction of feasible solutions additionally proved globally optimal. *Avg. Gap* is the average relative gap on the 4,842 instances solved by both methods, computed against the best objective found by either method.

Erdős–Rényi, random regular, and *Watts–Strogatz*. *Erdős–Rényi* hardware uses $n \in [20, 100]$ and $p \in [0.4, 0.7]$, resampling until connected. Random regular hardware uses $n \in [20, 100]$ and degree $d \in \{6, 8, 10, 12\}$ with parity correction. *Watts–Strogatz* hardware uses $n \in [20, 100]$, $k \in \{6, 8, 10, 12\}$ with $k < n$ and even parity, and rewiring probability $p \in [0.05, 0.2]$. For each problem–hardware pair, we sample the source size $|V_P|$ from $[\max(6, \lfloor 0.2|V_H| \rfloor), |V_H|]$, where $|V_H|$ is the size of the hardware graph. We reject and resample any pair violating the trivial lower bounds $|V_H| < |V_P|$ or $|E_H| < |E_P|$.

Since many minor-embedding instances cannot be solved to optimality within a practical time budget, we label them using both exact and heuristic methods. We run Gurobi on the chain-enumeration formulation with maximum chain length $L = 3$ and a 60-second time limit. In parallel, we run *minorminer* as a heuristic baseline up to 10 times per instance, or until 60 seconds total, again enforcing chain length at most $L = 3$. This gives a dataset in which exact optimization provides infeasibility certificates and quality guarantees when possible, while *minorminer* supplies additional feasible solutions on instances that are harder to solve exactly. Results of this comparison are reported in Table 9.

For OOD evaluation, we introduce both a hardware-topology shift and a problem-graph shift. On the hardware side, we use Pegasus graphs with $m \in \{2, 3\}$, which differ structurally from the Chimera and random hardware graphs seen in training. On the problem graph side, we generate graphs from a stochastic block model (SBM). Specifically, we sample the number of source vertices uniformly from $[0.20|V_H|, |V_H|]$, partition them into $B \in [2, 5]$ blocks with at least three nodes per block, and sample the intra-block density as $p_{\text{in}} \in [0.20, 0.85]$ and the inter-block density as $p_{\text{out}} \in [0.01, \min(0.35, p_{\text{in}} - 0.05)]$.

G.2 Supervised fine tuning hyperparameters

We detail the key hyperparameters used for training in table 10.

H Token Usage for Benchmark Models

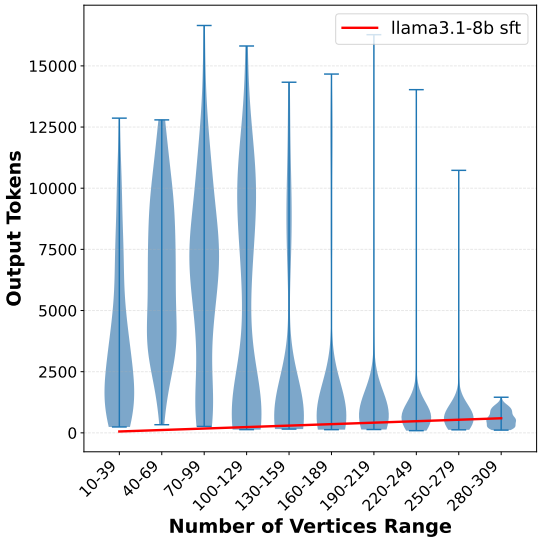
Figure 5 compares output token usage across tasks. While our fine-tuned model's token usage grows predictably with instance size due to its structured format, GPT-5.2 and Grok-4.1 often use fewer tokens on larger instances by prematurely defaulting to an "infeasible" verdict. Although this reduces inference cost, it represents a failure mode where models bypass the constructive search required for feasible solutions. This suggests that the lower token usage for these benchmarks is a byproduct of conservative negative bias rather than genuine efficiency, which aligns with the results in Tables 1 and 2.

I Detailed Results on UNSAT Instances for Minor-embedding

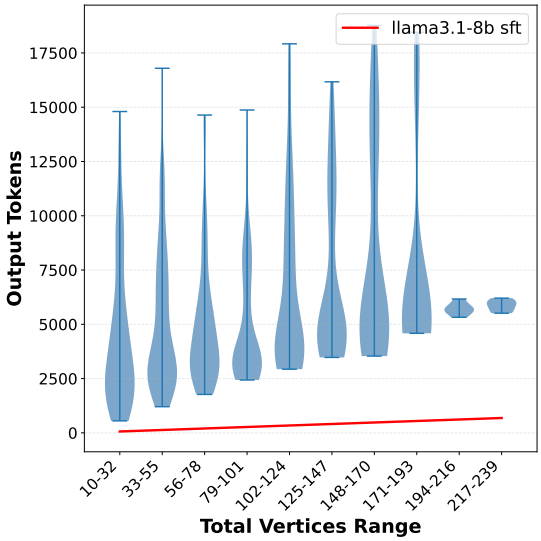
We report a breakdown of results for the UNSAT instances that passed zero-phase detection, which are presented in Table 11. Among the UNSAT instances, zero-phase infeasibility detection certifies 68.6% instances. The remaining 31.4% instances are not ruled out by zero phase and are therefore passed to the downstream optimization stage. We observe a slight drop in accuracy on these harder UNSAT instances, but the overall detection performance remains strong.

J Versatility Test with a Smaller Fine-tuned LLM

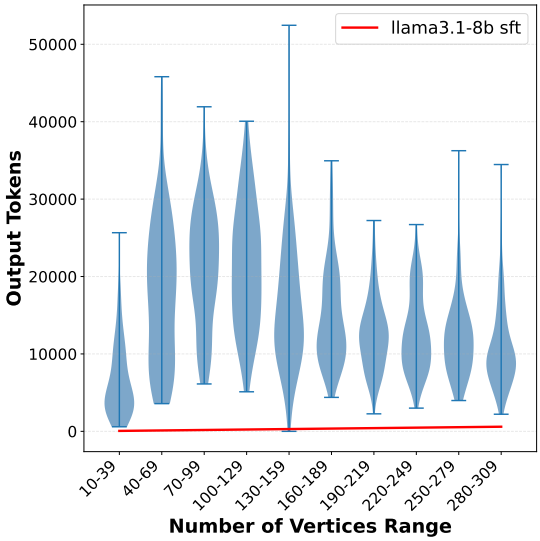
To assess whether the proposed framework depends on the capacity of an 8B model or on LoRA-based adaptation, we additionally fine-tune a smaller Llama-3.2-3B model using full fine-tuning on the same training and testing datasets. The results are reported in Table 12. Despite the smaller model size, the fine-tuned 3B model achieves strong and



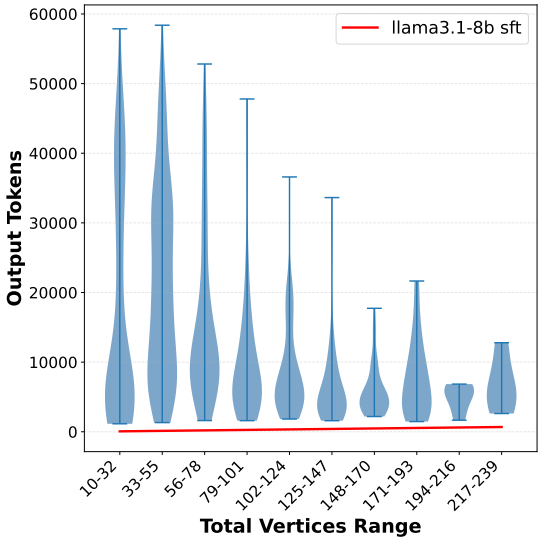
(a) GPT5.2: 3-coloring



(b) GPT5.2: minor-embedding



(c) Grok4.1: 3-coloring



(d) Grok4.1: minor-embedding

Figure 5: Token usage for GPT-5.2 and Grok-4.1 across tasks. As instance size increases, these models often default to an "infeasible" verdict, bypassing the search for valid solutions to save tokens. In contrast, our fine-tuned model's token usage scales linearly with problem size due to its structured format.

Setting	Value
Base model	Llama-3 8B
Framework	Unsloth + TRL SFTTrainer
LoRA rank r	32
LoRA alpha α	32
LoRA target modules	q, k, v, o, gate, up, down proj
Max sequence length	20000
Optimizer	AdamW
Learning rate	2×10^{-4}
Weight decay	10^{-3}
LR scheduler	Linear
Warmup steps	20
Per-device train batch size	2
Gradient accumulation	4
Effective batch size	8
Validation split	10%
Eval / save steps	200 / 200
Precision	bf16
Random seed	3407

Table 10: Main supervised fine-tuning hyperparameters.

Method	Rejected by zero phase	Passed zero phase
Llama-8B SFT Best-of-1	99.71%	97.45%
Llama-8B SFT Best-of-8	100.00%	99.36%

Table 11: Accuracy on UNSAT instances, separated by whether the instance is certified infeasible by zero-phase detection or passes zero phase and must be handled by the downstream optimization stage.

balanced performance across SAT and UNSAT instances.

These results suggest that the proposed infeasibility-aware training framework is not specific to the 8B LoRA setting. Rather, the performance gains appear to come primarily from the task-specific infeasibility-aware supervision and verifiable output format. This supports the versatility of the framework across model scales and fine-tuning strategies.

Model	Inf. Det.	SAT Acc.	UNSAT Acc.	Time (s)
Llama3.2-3b	37.9%	55.6%	20.2%	63.45 ± 61.34
Llama3.2-3b SFT BestOf1	**96.6%**	**97%**	96.2%	1.83 ± 2.42
Llama3.2-3b SFT BestOf8	97.5%	97.4%	97.6%	3.11 ± 3.65

Table 12: Evaluation of infeasibility detection for Llama3.2-3B

K Chain-of-Thought Analysis

We also inspect a representative chain-of-thought (CoT) trajectory generated by DeepSeek-V3.2 for a minor-embedding instance. (Detail in box K.1 and box K.2) The trajectory shows that the model can identify some relevant local structures, such as high-degree problem vertices, sparse regions of the hardware graph, and possible chain-length bottlenecks. However, the reasoning remains heuristic rather than certifying. In particular, the model moves from local observations about degree and connectivity to a global infeasibility conclusion without constructing a complete embedding search or deriving a valid infeasibility certificate. This example suggests that generic CoT is not necessarily a reliable mechanism for infeasibility-aware framework. The main difficulty is not only reasoning length, but whether the reasoning trajectory is sound, complete, and externally verifiable. This motivates our compact-output design: instead of asking the model to perform long internal search, we train it to output either a verifiable solution certificate or an infeasibility declaration, with downstream verification and repair. Designing problem-specific CoT or proof-guided reasoning for infeasibility detection remains an interesting future direction.

K.1 Input graphs

Original Input Graph Metadata

P edges: [[0,1],[0,2],[0,3],[0,4],[0,5],[0,6],[0,7],[1,4],[1,5],[1,6],[2,6],[3,4],[4,5],[4,7],[5,7]]
G edges: [[0,4],[0,5],[0,6],[0,7],[0,8],[1,4],[1,5],[1,6],[1,7],[1,9],[2,4],[2,5],[2,6],[2,7],[2,10],[3,4],[3,5],[3,6],[3,7],[3,11],[8,12],[8,13],[8,14],[8,15],[8,16],[9,12],[9,13],[9,14],[9,15],[9,17],[10,12],[10,13],[10,14],[10,15],[10,18],[11,12],[11,13],[11,14],[11,15],[11,19], [16,20],[16,21],[16,22],[16,23],[17,20],[17,21],[17,22],[17,23],[18,20],[18,21],[18,22],[18,23],[19,20],[19,21],[19,22],[19,23]]

K.2 CoT details

DeepSeek Chain-of-Thought Response

Let's reason step by step.

Step 1: Understanding the problem
We have:

- **P graph**: 8 nodes (0–7), edges listed. It is a star-like graph: node 0 connected to all others (1–7), plus some extra edges among the leaves:
  - 0: connected to 1,2,3,4,5,6,7
  - 1: connected to 4,5,6
  - 2: connected to 6
  - 3: connected to 4
  - 4: connected to 5,7
  - 5: connected to 7
  - 6: connected to [0,1,2]
  - 7: connected to [0,4,5]

So **P degree**: node 0: degree 7; node 1: degree 4; node 2: degree 2; node 3: degree 2; node 4: degree 5; node 5: degree 4; node 6: degree 3; node 7: degree 3.

Step 2: Hardware graph G structure
G has 24 nodes (0–23), edges given. Looking at edges, nodes 0–3 each connect to 4,5,6,7 and one of 8,9,10,11. For example, node 0 connects to 4,5,6,7,8; node 1 connects to 4,5,6,7,9; node 2 connects to 4,5,6,7,10; node 3 connects to 4,5,6,7,11.
The graph can be visualized as levels: 0–3 connected to 4–7 and 8–11; 8–11 connected to 12–15 and one of 16–19; and 16–19 connected to 20–23.

Step 3: Embedding constraints
We need chains of size at most 3 nodes per problem node. Since node 0 in P has degree 7, its chain must connect to chains of all other problem nodes. A single hardware node cannot provide enough adjacency, so node 0 likely needs a chain of at least two hardware nodes.

Step 4: Trying to embed
A possible attempt is to place problem node 0 on hardware chain [0,8]. The neighbors of this chain include 4,5,6,7,12,13,14,15,16, which may cover chains for problem nodes 1–7. However, the remaining problem nodes also have edges among themselves, such as (1,4), (1,5), (1,6), (4,5), (4,7), and (5,7), which must be realized by hardware edges between their chains.

Step 5: Difficulty from G's structure
In G, hardware nodes 4–7 are not connected to each other; they only connect back to nodes 0–3. Therefore, if several problem-node chains are placed entirely in 4–7, their mutual problem edges cannot be realized. To realize the dense leaf-to-leaf edges in P, chains would need to span multiple parts of G, but the chain size is limited to at most 3 and chains must be disjoint.

Step 6: Conclusion
Given P's edges among leaves, especially (1,4), (1,5), (1,6), (4,5), (4,7), (5,7), (3,4), and (2,6), and given G's restricted layered structure, it appears impossible to construct disjoint connected chains of size at most 3 that realize all required problem edges.

Answer:
no

L Training Dataset Example

Example Training Instance of minor-embedding

Instruction: Given a problem graph P with 9 nodes labeled 0..8 and a hardware graph G with 24 nodes, both undirected and given by edge lists, determine whether P can be minor-embedded into G. A valid embedding maps each problem node to a connected chain of hardware nodes, chains for different problem nodes are disjoint, and every problem edge (u,v) must be realized by at least one hardware edge between the two corresponding chains. Limit the chain size up to 3 nodes. Among feasible embeddings, minimize the total number of hardware nodes used. The input also provides, for each node, up to 2 neighbors with highest degree in the form Ni:[a,b,#c,#d], where a,b are neighbors and #c,#d are their degrees. Output exactly one of the following formats: yes, embedding: {problem_node: [hardware_nodes], ...}, total nodes used: {n_nodes_used} or no.

Input: P edges: [[0,4],[0,5],[0,7],[0,8],[1,2],[1,3],[1,6],[1,8],[2,4],[2,7],[2,8],[3,5],[3,6],[3,7],[4,6],[4,8],[5,6],[5,7]] P top2 neighbor-degree info: N0:[4,5,#4,#4]; N1:[2,3,#4,#4]; N2:[1,4,#4,#4]; N3:[1,5,#4,#4]; N4:[0,2,#4,#4]; N5:[0,3,#4,#4]; N6:[1,3,#4,#4]; N7:[0,2,#4,#4]; N8:[0,1,#4,#4] G edges: [[0,4],[0,5],[0,6],[0,7],[1,4],[1,5],[1,6],[1,7],[2,4],[2,5],[2,6],[2,7],[3,4],[3,5],[3,6],[3,7],[4,12],[5,13],[6,14],[7,15],[8,12],[8,13],[8,14],[8,15],[9,12],[9,13],[9,14],[9,15],[10,12],[10,13],[10,14],[10,15],[11,12],[11,13],[11,14],[11,15],[12,20],[13,21],[14,22],[15,23],[16,20],[16,21],[16,22],[16,23],[17,20],[17,21],[17,22],[17,23],[18,20],[18,21],[18,22],[18,23],[19,20],[19,21],[19,22],[19,23]] G top2 neighbor-degree info: N0:[4,5,#5,#5]; N1:[4,5,#5,#5]; N2:[4,5,#5,#5]; N3:[4,5,#5,#5]; N4:[12,0,#6,#4]; N5:[13,0,#6,#4]; N6:[14,0,#6,#4]; N7:[15,0,#6,#4]; N8:[12,13,#6,#6]; N9:[12,13,#6,#6]; N10:[12,13,#6,#6]; N11:[12,13,#6,#6]; N12:[4,20,#5,#5]; N13:[5,21,#5,#5]; N14:[6,22,#5,#5]; N15:[7,23,#5,#5]; N16:[20,21,#5,#5]; N17:[20,21,#5,#5]; N18:[20,21,#5,#5]; N19:[20,21,#5,#5]; N20:[12,16,#6,#4]; N21:[13,16,#6,#4]; N22:[14,16,#6,#4]; N23:[15,16,#6,#4]:

Output: yes, embedding: "0": [6,14], "1": [1,7,15], "2": [4], "3": [16,23], "4": [3,5], "5": [18,22], "6": [8,13,21], "7": [10,12,20], "8": [0], total nodes used: 19

Example Training Instance of 3-Coloring

Instruction: Given an undirected graph with 12 nodes labeled 0..11 and an edge list, decide whether the graph is 3-colorable. A valid 3-coloring assigns each node i a color $c_i \in \{0, 1, 2\}$ such that for every edge (u, v), $c_u \neq c_v$. The input also provides, for each node, up to 2 neighbors with highest degree in the form Ni:[a,b,#c,#d], where a,b are neighbors and #c,#d are their degrees. Output exactly one of: No OR Yes, coloring: [c0,c1,. . . ,c(n-1)].

Input: Edges: [(0,9),(0,10),(0,11),(0,4),(1,11),(1,4),(1,7),(2,6),(2,5),(3,8),(3,9),(4,7),(4,5),(4,10),(5,9),(5,11),(7,10),(8,9),(9,11)]
N0:[4,9,#5,#5]; N1:[4,11,#5,#4]; N2:[5,6,#4,#1]; N3:[9,8,#5,#2]; N4:[0,5,#4,#4]; N5:[4,9,#5,#5]; N6:[2,#2]; N7:[4,1,#5,#3]; N8:[9,3,#5,#2]; N9:[0,5,#4,#4]; N10:[4,0,#5,#4]; N11:[9,0,#5,#4]

Output: Yes, coloring: [0, 1, 1, 2, 2, 0, 0, 0, 0, 1, 1, 2]

Example Training Instance of Min Coloring

Instruction: Given an undirected graph with 48 nodes labeled 0..47 and an edge list, find a coloring that uses the minimum possible number of colors. A valid coloring assigns each node i a color c_i (a nonnegative integer) such that for every edge (u, v), $c_u \neq c_v$. The input also provides, for each node, up to 2 neighbors with highest degree in the form Ni:[a,b,#c,#d], where a,b are neighbors and #c,#d are their degrees. Output exactly: min_colors: K, coloring: [c0,c1,...,c(n-1)].

Input: Edges: [(0,44),(0,41),(0,34),(0,43),(0,38),(1,12),(1,23),(1,6),(2,45),(2,6),(2,24),(2,3),(2,12),(3,40),(3,24),(3,33),(3,45),(4,45),(4,13),(4,31),(4,33),(4,42),(4,14),(5,37),(5,21),(5,15),(6,11),(6,45),(6,26),(6,12),(7,19),(7,37),(7,30),(7,22),(8,41),(8,22),(8,35),(8,44),(8,30),(9,28),(9,45),(10,36),(10,40),(10,24),(10,16),(10,41),(11,23),(11,39),(12,29),(12,15),(13,42),(13,14),(13,18),(13,36),(14,18),(15,25),(15,29),(15,37),(16,40),(16,18),(16,36),(17,23),(17,32),(17,29),(17,47),(18,36),(18,31),(18,40),(18,33),(19,32),(19,29),(19,22),(20,23),(20,39),(21,37),(21,34),(21,27),(21,38),(21,30),(22,30),(22,32),(22,47),(23,29),(23,39),(24,25),(24,41),(24,43),(24,38),(24,40),(25,38),(26,45),(26,42),(27,34),(27,35),(27,46),(28,42),(29,32),(29,37),(32,47),(33,40),(34,46),(35,44),(35,46),(39,42),(39,47),(41,44),(41,43),(44,46)]
N0:[41,44,#6,#5]; N1:[6,23,#6,#6]; N2:[24,6,#8,#6]; N3:[24,40,#8,#6]; N4:[45,13,#6,#5]; N5:[21,15,#6,#5]; N6:[45,2,#6,#5]; N7:[22,37,#6,#5]; N8:[22,41,#6,#6]; N9:[45,28,#6,#2]; N10:[24,40,#8,#6]; N11:[6,23,#6,#6]; N12:[29,6,#7,#6]; N13:[18,4,#7,#6]; N14:[18,4,#7,#6]; N15:[29,12,#7,#5]; N16:[18,40,#7,#6]; N17:[29,23,#7,#6]; N18:[40,13,#6,#5]; N19:[29,22,#7,#6]; N20:[23,39,#6,#5]; N21:[37,27,#5,#4]; N22:[8,32,#5,#5]; N23:[29,39,#7,#5]; N24:[40,41,#6,#6]; N25:[24,15,#8,#5]; N26:[6,45,#6,#6]; N27:[21,34,#6,#4]; N28:[42,9,#5,#2]; N29:[23,12,#6,#5]; N30:[21,22,#6,#6]; N31:[18,4,#7,#6]; N32:[29,22,#7,#6]; N33:[18,4,#7,#6]; N34:[21,0,#6,#5]; N35:[8,44,#5,#5]; N36:[18,10,#7,#5]; N37:[29,21,#7,#6]; N38:[24,21,#8,#6]; N39:[23,42,#6,#5]; N40:[24,18,#8,#7]; N41:[24,0,#8,#5]; N42:[4,13,#6,#5]; N43:[24,41,#8,#6]; N44:[41,0,#6,#5]; N45:[4,6,#6,#6]; N46:[44,27,#5,#4]; N47:[22,32,#6,#5]

Output: min_colors: 4, coloring: [3, 0, 1, 0, 0, 0, 3, 1, 0, 3, 0, 0, 2, 1, 3, 1, 2, 0, 0, 0, 0, 3, 3, 2, 3, 0, 0, 1, 0, 3, 2, 3, 2, 3, 2, 3, 3, 2, 2, 3, 1, 2, 2, 0, 1, 2, 0, 1]

M Model-then-solve Details

Model-then-solve Prompt. Full response from ChatGPT is available in the supplementary material.

You are an expert in Operations Research, Mixed-Integer Linear Programming (MILP), and graph theory. Your task is to mathematically formulate and solve a Graph Minor-Embedding problem using Python and the `gurobipy` library.

Problem Description
We are given two undirected graphs:

1. A Problem Graph, P, with `n` nodes labeled `0` to `n-1`, and a set of edges E_P.
2. A Hardware Graph, G, with `m` nodes labeled `0` to `m-1`, and a set of edges E_G.

We want to find a valid minor-embedding of P into G. A valid embedding maps each problem node i in P to a subset of hardware nodes S_i in G (called a "chain"), subject to the following rules:

1. **Disjoint Chains:** For any two distinct problem nodes i and j, their corresponding chains must not overlap (i.e., $S_i \cap S_j = \emptyset$).
2. **Chain Connectivity:** For every problem node i, the subgraph induced by S_i in G must be connected.
3. **Chain Size Limit:** The maximum number of hardware nodes in any chain S_i is strictly limited to 3 (i.e., $|S_i| \leq 3$).

4. **Edge Realization:** For every problem edge (i, j) in E_P, there must exist at least one hardware edge (u, v) in E_G such that $u \in S_i$ and $v \in S_j$.

Objective
Among all feasible valid embeddings, minimize the total number of hardware nodes utilized across all chains (i.e., minimize the sum of $|S_i|$ for all i).

Task 1: Mathematical Formulation
Write a complete, rigorous Mixed-Integer Linear Programming (MILP) formulation for this problem. You must strictly provide:

- **Sets and Indices:** Clearly define all sets (e.g., nodes, edges).
- **Parameters:** Define any given constants.
- **Decision Variables:** Clearly define all binary/integer/continuous variables.
- **Objective Function:** The mathematical expression to minimize.
- **Constraints:** All rules stated above translated into linear inequalities. Provide brief, logical explanations for each constraint block.

Task 2: Python + Gurobi Implementation
Write a self-contained Python script using the `gurobipy` library to solve this model.

- Encapsulate the logic in a function with the following signature:

```
def solve_minor_embedding(n: int, P_edges: list, m: int, G_edges: list) ->
                                    dict:
```

- The function should return a dictionary mapping each problem node to its list of hardware nodes (e.g., `{0: [1, 2], 1: [3], ...}`). If the problem is infeasible, return `None`.
- Ensure the code strictly follows your mathematical model.
- Comment the code to show which block of constraints corresponds to which mathematical equation.
- Include a small, reproducible `if __name__ == "__main__":` block with dummy data to test the function.