LLM2Prune: Using LLMs as Domain Experts for Search Space Reduction

Anonymous authors
Paper under double-blind review

Abstract

Combinatorial optimization problems defined over graphs involve large discrete search spaces where many candidates contribute little due to redundancy or low value. Pruning the ground set to a smaller pool of promising candidates makes heuristics and exact solvers practical for large real-world instances. Classical submodularity-based pruning algorithms do not scale efficiently, while learning-based approaches depend on handcrafted features that require domain expertise and limit generalization. We propose LLM2Prune, a framework that uses large language models (LLMs) to generate features from a task description, which are then used by a downstream classifier to prune the search space. We guide the feature discovery process with feature-importance scores and performance metrics. Across diverse graph optimization tasks, LLM2Prune prunes over 90% of the ground set while retaining near-optimal solutions, achieving orders-of-magnitude speedups over existing approaches. Code, data, and pre-trained models are available at: `https://anonymous.4open.science/r/LLM_Pruning-798E`.

1 Introduction

Combinatorial optimization problems on graphs arise in many application domains, including social networks, transportation, telecommunications, and scheduling, and are typically NP-hard. For instance, in facility location problems such as opening retail outlets under a budget constraint, many candidate sites are unpromising due to high costs, low demand, or redundancy with nearby locations. In fact, among the 21 NP-complete problems introduced by Karp (Karp, 2009), 10 are decision versions of graph optimization problems, while most of the remaining problems, such as set covering, can be naturally formulated on graphs. Several heuristics have been developed to obtain near-optimal solutions, but they do not typically scale well with problem size. Pruning the ground set to a smaller pool of promising candidates enables heuristics and exact methods to scale more efficiently without sacrificing solution quality.

In this work, we study the problem of identifying a smaller subset of the ground set for graph optimization problems, where the goal is to maximize or minimize an objective function over a large discrete configuration space subject to constraints. Formally, we have the following problem definition for the general CO problem.

Problem Definition (Pruning). A CO problem can be expressed as the tuple (S, Ω, f), where S denotes the search space over discrete decision variables, Ω represents the set of constraints that define feasibility, and $f : S \to \mathbb{R}$ is the objective function to be optimized. Given a CO problem with ground set $\mathcal{U}$ and an exact solver $\mathcal{OPT}$, the optimal solution is

$$X^\star = \mathcal{OPT}(\mathcal{U}) = \arg \max_{X \subseteq \mathcal{U},\ X \text{ satisfies } \Omega} f(X)$$

The pruning problem seeks to construct a much smaller ground set $\mathcal{U}'$ with $|\mathcal{U}'| \ll |\mathcal{U}|$, such that solving the reduced instance yields nearly the same objective value:

$$\frac{f(\mathcal{OPT}(\mathcal{U}'))}{f(\mathcal{OPT}(\mathcal{U}))} = 1 - \epsilon$$

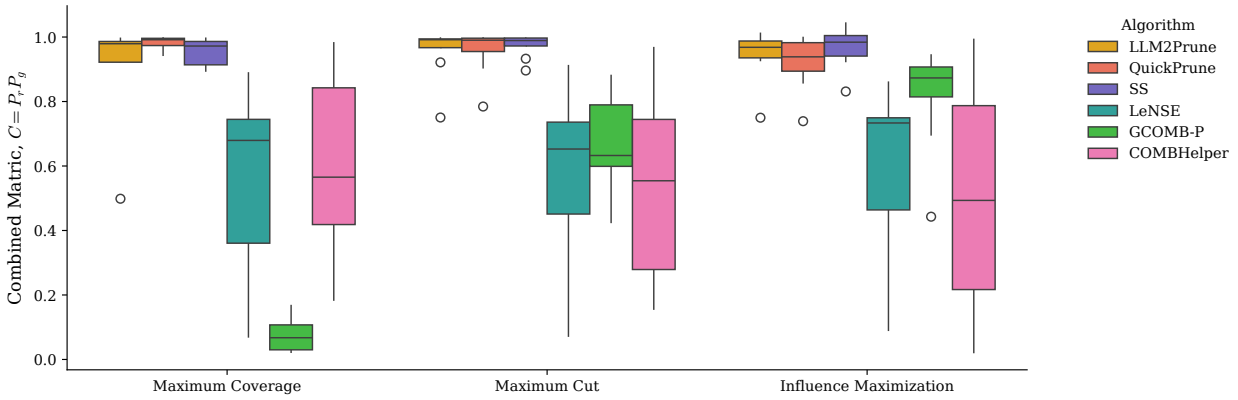

Figure 1: Comparison of pruning algorithms across three CO problems. The boxplots report the combined metric $C = P_r P_g$ (defined below), which balances solution quality and ground set reduction. LLM2Prune (ours) is competitive with classical approaches (QuickPrune, SS) and outperforms learning-based approaches (GCOMB, LeNSE, COMBHelper) by a large margin, without relying on domain knowledge.

for a small approximation factor $\epsilon \geq 0$.

Existing pruning approaches are limited by scalability issues (Zhou et al., 2017; Nath & Kuhnle, 2025), reliance on handcrafted problem-specific features (Ireland & Montana, 2022; Tian et al., 2024), lack of problem-agnostic pipelines or weak performance on general graph optimization problems (Lauri & Dutta, 2019; Lauri et al., 2023; Manchanda et al., 2020), and poor adaptability to constraints. Thus, in this work, we are motivated by the following question:

Is it possible to design a scalable pruning framework that avoids handcrafted features, delivers strong performance across general CO problems over graphs, and adapts easily to constraints?

To address these limitations, we propose LLM2Prune, a framework that replaces prior supervised methods relying on problem-specific, hand-crafted features that limit generality across graph optimization problems and constraints In addition, it overcomes classical methods that use sequential pruning, which struggle with scalability. Our approach uses LLMs to automatically generate features for a downstream classifier, which is then applied to prune the ground set of a graph optimization problem. Given a problem definition, we employ an LLM-guided beam search over the feature space, directed by both performance evaluation and feature importance scores. These signals act as surrogate gradients, steering the search toward high-quality features that improve the performance of the downstream classifier. In contrast, prior works that employ LLMs for feature generation either rely on manually specified rules, derive new features only from existing ones, or use simple feedback loops that allow error propagation and lack feature-importance guidance. By comparison, our approach generates features in a guided search framework that balances exploration with structured signals, making it robust to noisy feedback and enabling consistent feature discovery.

Contributions. Our work makes the following contributions:

- We propose LLM2Prune, a framework that leverages the reasoning capabilities of LLMs to automatically generate features for a lightweight classifier that prunes the search space of CO problems over graphs. Unlike prior approaches, it avoids hand-crafted, problem-specific engineering and can easily adapt to constraints.
- We introduce an LLM-guided beam search strategy for feature generation that incorporates feature-importance score and performance evaluation. This approach overcomes the limitations of a simple feedback loop by maintaining diversity and stability in feature exploration.
- We evaluate LLM2Prune on diverse graph optimization problems under both size and knapsack constraints. Under size constraints, LLM2Prune achieves solution quality comparable to classical

methods while enabling substantially faster pruning. It consistently outperforms state-of-the-art learning-based pruning approaches in solution quality and runtime. The ability of LLM2Prune to adapt naturally to additional constraints is demonstrated under knapsack constraints, where it achieves superior performance compared to existing methods.

Organization. The rest of this paper is organized as follows. In Section 2, we describe the relationship between our contribution and existing work. In Section 3, we present our algorithm, LLM2Prune, and conduct an empirical evaluation, comparing it to existing learned and classical heuristics for the pruning problem, followed by a discussion of limitations and future directions. Finally, in Section 5, we conclude the paper.

2 Related Work

Our work closely relates to two lines of research: pruning the search space for CO problems and automated feature engineering with LLMs. We emphasize that our approach is orthogonal to methods that directly generate algorithms using LLMs, which often exhibit degraded performance as graph size increases (Yang et al., 2023; Liu et al., 2024). Instead, we leverage the reasoning capabilities of LLMs to generate features for a learned algorithm for pruning.

Automated Feature Engineering via LLMs. Automated feature engineering generates useful features from raw data without human effort. Prior work with LLMs has focused primarily on tabular data, where new features are derived from existing ones. These approaches either rely on manually designed feature-generation rules (Zhang et al., 2023; 2024) or few-shot prompts (Han et al., 2024), and thus depend on hand-crafted design choices. Moreover, existing methods (Nam et al., 2024; Hollmann et al., 2023) suffer from error propagation caused by simple feedback loops.

In contrast, our method does not rely on existing features, a predefined search space, or few-shot examples. Instead, it uses feature-importance scores to identify useful features rather than depending solely on performance evaluation. To reduce error propagation and ensure stable feature generation, we incorporate beam search, which produces more robust and consistent results.

Pruning Search Space for CO Problems. To improve the scalability of heuristics, several works have proposed to prune the search space of CO problems and focus only on the reduced ground set. These approaches fall into two categories: classical and learning-based algorithms.

Classical Approaches. Zhou et al. (2017) propose a pruning algorithm based on submodularity to reduce the cost of submodular maximization under size constraints. Similarly, Nath & Kuhnle (2025) introduce a submodularity-based pruning algorithm that provides theoretical guarantees on both the fraction of the optimal value retained and the size of the resulting pruned ground set. The main limitation of these approaches is that they prune sequentially, which makes the process time-consuming. In contrast, our algorithm prunes in a single step. While this can produce a slightly larger pruned ground set in some cases, we demonstrate that our algorithm is orders of magnitude faster, particularly on large datasets.

Learning-Based Approaches. These algorithms can be divided into supervised learning methods and hybrid approaches that combine supervised and reinforcement learning. In the supervised setting, small instances are solved with exact or heuristic solvers, and the solutions are used to train a binary classifier that predicts whether a vertex or edge can be safely removed without compromising solution quality. These methods rely on hand-crafted, problem-specific features. For Maximum Clique Enumeration, Lauri & Dutta (2019) apply logistic regression with hand-crafted features to prune vertices, while Grassia et al. (2019); Lauri et al. (2023) extends this line of work with a multi-stage pruning strategy in which successive classifiers, trained on graph-theoretic and statistical features, iteratively reduce the search space. Similarly, Fitzpatrick et al. (2021); Sun et al. (2021) train classifiers to prune edges unlikely to belong to the optimal tour in Traveling Salesman Problem instance.

The main limitation of these methods is their heavy reliance on problem-specific manual feature engineering and trial-and-error design, which restricts generalization. To address this, Tian et al. (2024) proposes COMBHelper, a GNN-based framework with knowledge distillation and a problem-specific boosting mod-

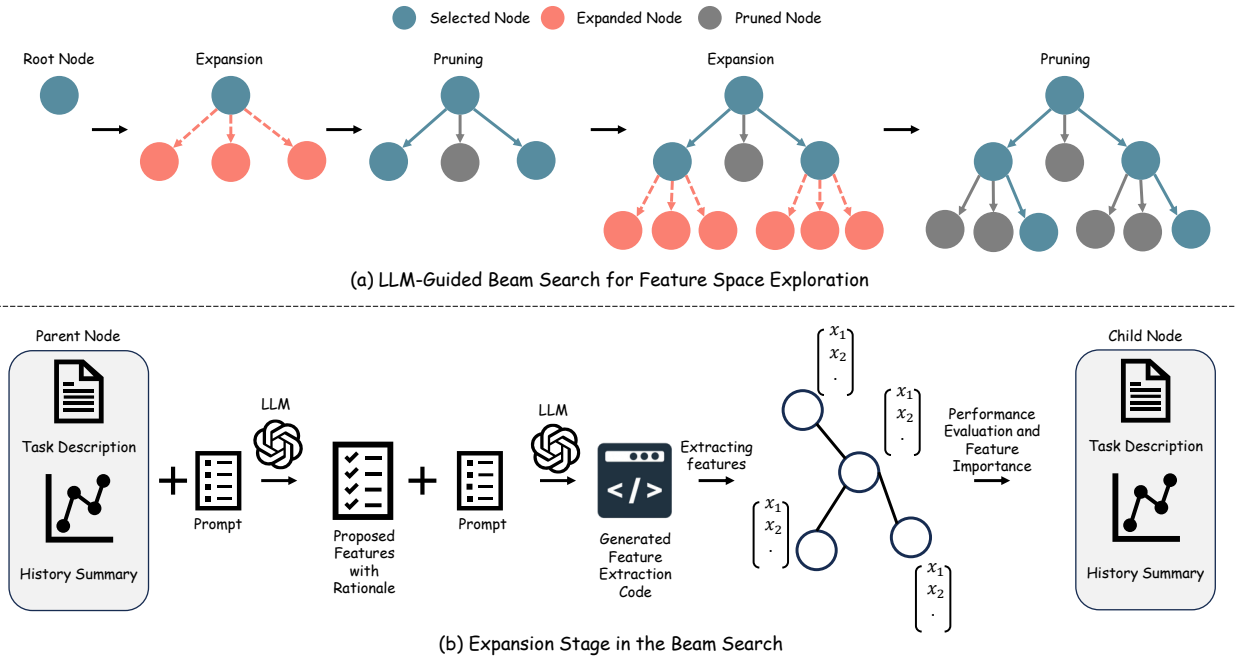

Figure 2: (a) Two phases of LL2Prune, illustrated with beam width $\beta = 2$ and expansion factor $\gamma = 3$. In the expansion stage, each node expands into γ child nodes and in the pruning stage the beam grows by another depth, keeping its width size to β after pruning out the candidate nodes with poor performances. (b) An overview of the expansion stage. Each parent node, consisting of a task description and a history summary, is used to prompt the LLM to propose new features. A second prompt generates executable feature extraction code, which is applied to produce features. These are evaluated for performance and importance, and the resulting history is carried forward to form child nodes.

ule designed to generalize across CO problems over graphs. While effective, it still depends on computationally expensive Fourier Feature Mapping and problem-specific boosting.

Hybrid methods that integrate supervised and reinforcement learning are generally more problem-agnostic. For instance, GCOMB (Manchanda et al., 2020) uses a probabilistic greedy algorithm to generate training data, followed by a linear regression model that applies a weighted-degree heuristic to prune the set. A Q-learning network is then trained on this reduced search space to sequentially predict the solution. A key limitation is that the initial pruning depends on a handcrafted degree heuristic, which does not generalize well across CO problems (Ireland & Montana, 2022).

Similarly, LeNSE (Ireland & Montana, 2022) employs a two-stage learning process. First, a GNN is trained in a supervised manner to generate embeddings for subgraphs sampled from training graphs. Then, a policy network uses these embeddings to guide a local search procedure, moving from an initial subgraph to one more likely to contain the optimal solution. A major drawback, however, is its reliance on domain expertise to categorize subgraphs, set the number of samples per class, and tune critical hyperparameters such as subgraph size and embedding dimensionality separately for each dataset and problem.

As opposed to prior learning-based approaches, LLM2Prune makes the pruning process fully automated, requiring no domain knowledge, and can easily adapt to constraints.

3 LLM2Prune

The main goal of our work is to design a learning-based framework that prunes the search space through automated reasoning, making the pruning process generalize across graph optimization problems while accommodating additional constraints. We formalize the feature space search with LLMs as a decision tree,

denoted by $\mathcal{T}$. Each node in the tree will have its own proposed feature set and classifier, along with a running log of everything tried so far, including good features, bad features, performance results, and importance scores, so the LLM can propose new features knowing both the current state and the full path of past attempts. Formally, each node is denoted by $n_{d,i}$, where $d \geq 0$ indicates the depth of the node in the tree and $i \in \{1, \ldots, N_d\}$ indexes the node among all N_d nodes at depth d.

Each node $n_{d,i}$ is represented as a tuple

$$n_{d,i} = \big(\mathcal{F}_{d,i}, \mathcal{M}_{d,i},\ g_{\theta_{d,i}},\ \mathcal{H}_{d,i}\big),$$

where $\mathcal{F}_{d,i}$ is the proposed feature set, $g_{\theta_{d,i}}$ is the trained classifier on $\mathcal{F}_{d,i}$, and $\mathcal{M}_{d,i}$ denotes the performance metric of the classifier, defined as the combined metric $C = P_r \cdot P_g$ (see Section 4) evaluated on a held-out synthetic validation graph. The final component, $\mathcal{H}_{d,i}$, represents the history, defined as the accumulation of all feature sets, performance metrics, feature-importance vectors, and discarded features (if execution exceeds the timeout τ) along the unique path from the root node $n_{0,1}$ to the node $n_{d,i}$:

$$\mathcal{H}_{d,i} = \bigcup_{\ell=0}^{d} \bigcup_{j \in \pi(\ell,i)} \Big(\mathcal{F}_{\ell,j},\ M_{\ell,j},\ \mathrm{Imp}_{\ell,j},\ \mathcal{F}_{\ell,j}^{\mathrm{fail}}\Big),$$

where $\pi(\ell, i)$ denotes the ancestor index at depth ℓ along the path from the root to $n_{d,i}$, $M_{\ell,j}$ is the performance metric of the classifier, $\mathrm{Imp}_{\ell,j}$ is the feature-importance vector, and $\mathcal{F}_{\ell,j}^{\mathrm{fail}}$ is the set of features discarded at $n_{\ell,j}$ due to exceeding cut-off time.

The cumulative history $\mathcal{H}_{d,i}$ that records all past feature proposals, performance evaluations, and importance scores along the path from the root. This formulation ensures that the LLM generates new feature proposals with full awareness of both the local information at the node and the trajectory that led to it. We condition on the full path rather than only the immediate parent because the LLM may otherwise re-propose features explored and discarded at earlier depths; providing the complete history of past attempts avoids this redundancy and reduces the number of repeated proposals during search.

Our algorithm begins at the root node and expands the search tree one depth level at a time in a breadth-first manner. At each depth d, the procedure consists of two phases: expansion and pruning (as shown in Figure 2A).

Expansion. The expansion phase of our algorithm follows the expansion step of standard beam search. Given an expansion factor γ, each node $n_{d,i}$ expands into γ child nodes. These child nodes are produced directly by the LLM, which takes as input the task description with constraints and the accumulated history $\mathcal{H}_{d,i}$, together with a feature-generation meta-prompt Z_{feature}.

This meta-prompt does not simply request new features but provides higher-level guidance on how the LLM should generate new features. Specifically, Z_{feature} (see Appendix A.1) instructs the LLM to enumerate and retain important features, remove redundant or uninformative ones, and propose new features derived from the current set of high-performing features. In addition, we include meta-instructions requiring the LLM to provide both a formal definition and a rationale for each proposed feature.

A key property of this step is that the LLM is stochastic: even under identical inputs, it can generate diverse outputs due to sampling mechanisms such as temperature, thereby broadening exploration of the feature space. Formally, the child feature sets are sampled as

$$\big\{\mathcal{F}_{d+1,j}\big\}_{j=1}^{\gamma} \sim \mathrm{LLM}\big(Z_{\mathrm{feature}}, \mathrm{Task}, \mathrm{Summary}(\mathcal{H}_{d,i})\big),$$

where $\mathrm{Summary}(\mathcal{H}_{d,i})$ denotes the LLM-generated summary of the history at node $n_{d,i}$. We provide a meta-prompt Z_{summary} (see Appendix A.2) that summarizes the history feedback into key points and insights. Instead of providing the entire feedback history from the root to a node, this approach keeps the feedback compact and makes the reasoning process easier for the LLM to follow.

Given a child node with its generated feature set $\mathcal{F}_{d+1,j}$, we prompt the LLM with meta-prompt Z_{code} (see Appendix A.3) to produce executable code for each proposed feature $f \in \mathcal{F}_{d+1,j}$. Executing this code yields

the feature values for the training data. To ensure computational efficiency, we impose a timeout threshold τ. If execution exceeds τ, the failure is recorded in the history $\mathcal{H}_{d+1,j}$ and the corresponding feature is discarded. This mechanism (i) keeps the pruning process lightweight by excluding costly features and (ii) prevents further exploration of features correlated with discarded ones. If the execution of a code block raises an error, this error is passed to the LLM for the next code generation iteration.

Traditional supervised approaches (Lauri et al., 2023; Tian et al., 2024) often assume that all nodes not included in the solution obtained from an exact or heuristic solver are equally unimportant, an assumption known to introduce label noise (Manchanda et al., 2020). While Manchanda et al. (2020) addresses this issue using a probabilistic greedy approach, where the loss of classifying each node is proportional to how frequently it is selected by solvers, we instead integrate semi-supervised learning to mitigate label noise. We then train a classifier g_θ on the features in $\mathcal{F}_{d+1,j}$. For each element $v \in \mathcal{U}$, the model outputs a probability distribution $p_\theta(y \mid v)$ over labels $y \in \{0, 1\}$. After a warm-up period, we retain the high-confidence nodes

$$\mathcal{V}_{\text{conf}} = \{v \in \mathcal{V} : \max_y p_\theta(y \mid v) \geq \delta\},$$

where δ is a confidence threshold. For each $v \in \mathcal{V}_{\text{conf}}$, we assign a soft pseudo-label

$$\tilde{y}_v = p_\theta(y \mid v).$$

These soft pseudo-labels, together with the ground-truth labels on the supervised subset $\mathcal{V}_{\text{labeled}}$, guide further training and improve robustness by incorporating uncertainty rather than enforcing hard assignments.

The downstream classifier g_θ is implemented as a two-layer Graph Convolutional Network (GCN) (Kipf & Welling, 2016), operating on the LLM-proposed node-level features. For each node $v \in \mathcal{U}$, it outputs a probability $p_\theta(y = 1 \mid v)$ indicating how likely the node belongs in the pruned ground set.

We then validate the trained network and extract feature importance scores using an explainability algorithm, which provides insights into how different features influence the model's predictions. This information constitutes new feedback for the corresponding child node and is appended to its history. To extract feature-importance vectors, we apply GNNExplainer (Ying et al., 2019), which provides attribution scores indicating each feature's contribution to the classifier's predictions. These scores form the feedback signal $\text{Imp}_{d,i}$ used to guide subsequent feature generation.

Pruning. Let the set of nodes at depth d be denoted by

$$\mathcal{N}_d = \{n_{d,1}, \ldots, n_{d,N_d}\}.$$

From this set, the algorithm maintains a beam

$$B_d \subseteq \mathcal{N}_d, \quad |B_d| = \beta,$$

where β is the beam width. During pruning, the algorithm evaluates each candidate node $n_{d+1,j} \in \mathcal{C}_{d+1}$ using its performance metric $M_{d+1,j}$, and constructs the next beam B_{d+1} by selecting top-β nodes.

The process continues until the search reaches a maximum depth $D_{\max}$, which we treat as a hyperparameter. At this point, the algorithm terminates, and the best node overall is returned as

$$n^{\text{best}} = \arg\max_{n_{d,i} \in \mathcal{T}} M_{d,i}.$$

Finally, when beam search finishes, we keep only the best node found so far based on the performance metric. In Appendix F, we compare the beam-search variant against alternative configurations, e.g., a simple feedback loop and a no-feedback baseline, to isolate the effects of iterative feedback and feature-importance guidance. We emphasize that LLM2Prune is a general framework that can work with any classifier and any explainability algorithms applied to that classifier.

The beam search is performed once on synthetic graphs that match the structural distribution of the target domain. The metric M serves as a selection criterion among candidate feature sets during search. Once the best feature set is identified, the classifier is fine-tuned on the actual target training graph and applied to prune the ground set. Algorithm 1 describes the full LLM2Prune pipeline.

Algorithm 1 LLM2PRUNE

Require: Task description, ground set $\mathcal{U}$, beam width β, expansion factor γ, max depth $D_{\max}$, timeout τ, confidence threshold δ

Ensure: Pruned ground set $\mathcal{U}' \subseteq \mathcal{U}$

Initialize root node $n_{0,1}$ with empty feature set and history $\mathcal{H}_{0,1} = \emptyset$
$B_0 \leftarrow \{n_{0,1}\}$
for $d = 0, 1, \ldots, D_{\max} - 1$ **do**
  $\mathcal{C}_{d+1} \leftarrow \emptyset$
  for each $n_{d,i} \in B_d$ **do** ▷ Expansion
    summary $\leftarrow$ LLM(Z_{summary}, $\mathcal{H}_{d,i}$)
    for $j = 1, \ldots, \gamma$ **do**
      $\mathcal{F}_{d+1,j} \sim$ LLM(Z_{feature}, Task, summary)
      Generate and execute code for each $f \in \mathcal{F}_{d+1,j}$; discard features exceeding τ
      Train $g_{\theta_{d+1,j}}$ on $\mathcal{F}_{d+1,j}$ with semi-supervised learning (threshold δ)
      Compute $M_{d+1,j}$ and $\text{Imp}_{d+1,j}$; update $\mathcal{H}_{d+1,j}$
      $\mathcal{C}_{d+1} \leftarrow \mathcal{C}_{d+1} \cup \{n_{d+1,j}\}$
    end for
  end for
  $B_{d+1} \leftarrow$ top-β nodes in $\mathcal{C}_{d+1}$ by M ▷ Beam pruning
end for
$n^{\text{best}} \leftarrow \arg\max_{n_{d,i} \in \mathcal{T}} M_{d,i}$
Fine-tune $g_{\theta^{\text{best}}}$ on the target training graph
return top-ranked nodes of $\mathcal{U}$ under $g_{\theta^{\text{best}}}$ as $\mathcal{U}'$

Table 1: Comparison of pruning algorithms for size-constraint experiments. Highlighted are the results ranked **first**, **second** , and **third**. *Values are reported from Ireland & Montana (2022); Nath & Kuhnle (2025). "–" indicates that the corresponding algorithm obtained no reasonable result. Due to space constraints, we report the results for Maximum Cover in Appendix G, and the results are qualitatively similar.

Dataset	Classical Approach						Learning-Based Approach											
	QuickPrune			SS			**LLM2Prune**			GCOMB-P*			LeNSE*			COMBHelper*		
	P_r	P_g	C	P_r	P_g	C	P_r	P_g	C	P_r	P_g	C	P_r	P_g	C	P_r	P_g	C
Maximum Cut																		
Facebook	0.9886	0.7935	**0.7845**	0.9928	0.9027	**0.8962**	1.0004	0.7501	0.7504	0.8130	0.9500	**0.7723**	1.0000	0.0700	0.0700	1.0000	0.1538	0.1538
DBLP	1.0000	0.9954	**0.9954**	1.0000	0.9963	**0.9963**	0.9981	0.9841	**0.9822**	0.6460	0.9900	0.6395	0.9930	0.9200	0.9136	1.0000	0.1825	0.1825
Deezer	0.9996	0.9730	**0.9726**	0.9999	0.9855	**0.9854**	0.9999	0.9907	**0.9906**	0.8500	0.9900	0.8415	0.9750	0.7400	0.7215	1.0000	0.3754	0.3754
Slashdot	1.0000	0.9904	**0.9904**	1.0000	0.9889	**0.9889**	1.0000	0.9926	**0.9926**	0.6320	0.9900	0.6257	0.9900	0.6200	0.6138	0.7935	0.9929	0.7879
Wiki	0.9985	0.9037	**0.9023**	0.9981	0.9346	**0.9328**	1.0000	0.9214	**0.9214**	0.9200	0.9600	0.8832	0.9810	0.3900	0.3826	1.0000	0.7011	0.7011
Twitter	1.0000	0.9900	**0.9900**	1.0000	0.9893	**0.9893**	1.0000	0.9938	**0.9938**	0.6280	0.9900	0.6217	0.9870	0.4800	0.4738	1.0000	0.5542	0.5542
YouTube	1.0000	0.9994	**0.9994**	1.0000	0.9987	**0.9987**	1.0000	0.9991	**0.9991**	0.5360	0.9900	0.5306	0.9870	0.7900	0.7797	0.9990	0.9705	0.9695
Skitter	1.0000	0.9995	**0.9995**	1.0000	0.9991	**0.9991**	0.9966	0.9988	**0.9954**	0.4270	0.9900	0.4227	0.9740	0.7100	0.6915	–	–	–
Influence Maximization																		
Facebook	0.9919	0.7450	**0.7390**	0.9208	0.9027	**0.8312**	0.9995	0.7501	**0.7498**	0.9510	0.7300	0.6942	0.9790	0.0900	0.0881	1.0039	0.3248	0.3261
DBLP	0.8603	0.9946	0.8557	1.0495	0.9963	**1.0456**	0.9953	0.9984	**0.9937**	0.8630	0.9900	0.8544	0.9690	0.8900	**0.8624**	0.9873	0.0194	0.0192
Deezer	0.9384	0.9698	**0.9101**	1.0262	0.9855	**1.0113**	0.9759	0.9907	**0.9668**	0.8050	0.5500	0.4428	0.9720	0.7600	0.7387	0.9837	0.1093	0.1075
Slashdot	0.9984	0.9881	**0.9865**	0.9959	0.9889	0.9848	1.0215	0.9926	**1.0139**	0.9660	0.9800	0.9467	0.9660	0.7700	0.7438	1.0076	0.9875	**0.9950**
Wiki	1.0269	0.8831	**0.9069**	0.9863	0.9346	**0.9218**	1.0039	0.9214	**0.9249**	0.9690	0.9000	0.8721	0.9600	0.5100	0.4896	0.9969	0.6066	0.6047
Twitter	1.0045	0.9965	**1.0010**	0.9575	0.9893	**0.9473**	0.9817	0.9876	**0.9695**	0.9200	0.9800	0.9016	0.9660	0.4000	0.3864	0.9972	0.4947	0.4933
YouTube	0.9677	0.9994	0.9671	0.9846	0.9987	**0.9833**	1.0005	0.9991	**0.9996**	0.9330	0.9900	0.9237	0.9710	0.7500	0.7282	0.9992	0.9705	**0.9697**
Skitter	0.9813	0.9999	**0.9812**	1.0032	0.9991	**1.0023**	1.0035	0.9970	**1.0006**	0.8830	0.9900	0.8742	0.9830	0.7800	0.7667	–	–	–

4 Empirical Evaluation

In this section, we compare the empirical performance of LLM2PRUNE with existing methods for the pruning problem across three applications, following (Ireland & Montana, 2022; Manchanda et al., 2020; Nath & Kuhnle, 2025).

Summary of Results. For all experiments, we show that LLM2PRUNE typically prunes over 90% of the ground set while sacrificing very little objective value, achieving competitive performance or outperforming

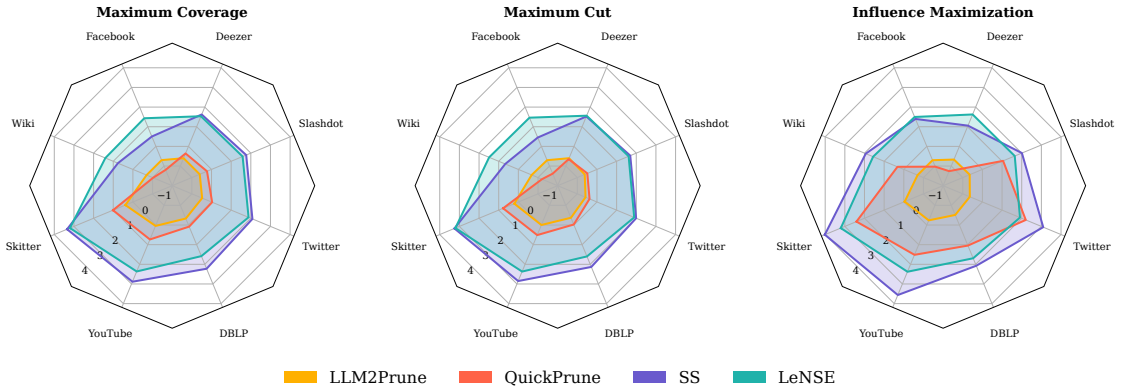

Figure 3: Runtime comparison (log scale) of pruning algorithms on Maximum Coverage, Maximum Cut, and Influence Maximization across eight datasets.

the competing methods. Under the size constraint, although classical approaches such as QuickPrune (Nath & Kuhnle, 2025) and Submodular Sparsification (SS) (Zhou et al., 2017) outperform our method on the combined metric (defined below) in some cases, LLM2Prune can be orders of magnitude faster while remaining competitive, since classical approaches sequentially prune the ground set, as shown in Figure 3. Among learning-based pruning methods, LLM2Prune consistently outperforms GCOMB-P (Manchanda et al., 2020), LeNSE (Ireland & Montana, 2022), and COMBHelper (Tian et al., 2024) across nearly all evaluated instances. Furthermore, under the knapsack constraint, LLM2Prune outperforms TOP-k and QuickPrune while retaining the flexibility and efficiency observed in the size-constrained setting, demonstrating that our framework generalizes naturally to different constraints.

Evaluation Metrics. Following Ireland & Montana (2022); Nath & Kuhnle (2025), we evaluate the algorithms using three performance metrics, where higher values indicate better performance. The first is the pruning approximation ratio P_r, defined as the ratio of the objective value obtained from the pruned ground set $\mathcal{U}'$ to that from the original ground set $\mathcal{U}$, both computed by a heuristic $\mathcal{H}$, i.e., $P_r = f(\mathcal{H}(\mathcal{U}'))/f(\mathcal{H}(\mathcal{U}))$. The second is the pruned fraction P_g, which measures the proportion of the original ground set that is removed. Finally, to avoid trivial solutions that maximize one metric while disregarding the other (e.g., pruning nothing or pruning everything), we use the combined metric $C = P_r P_g$. We want to highlight that C can exceed one than one in cases where the heuristic can find a better solution on the pruned ground set than on the full ground set. This is possible because the heuristic does not guarantee an exact solution, and with less noise, it may find a better solution on the reduced set. Similar findings are reported in Ireland & Montana (2022).

Applications. We evaluate our algorithm on three problems: Maximum Coverage (MaxCover), Maximum Cut (MaxCut), and Influence Maximization (IM), under both size and knapsack constraints, following Manchanda et al. (2020); Ireland & Montana (2022). Detailed specifications of these applications are provided in Appendix B, along with the optimal features identified by our framework.

Discovered Features. A key strength of LLM2Prune is that it generates features that are not generic graph statistics but are grounded in the problem objective and constraints. For MaxCut, it discovers *random cut expectation* (the expected marginal contribution of a node to the cut under a random partition), a non-trivial descriptor directly derived from the cut objective. For IM, it identifies *average incoming activation probability* and *outgoing activation probability sum*, which require reasoning about the independent cascade model and capture a node's susceptibility and direct influence spread respectively. Under the knapsack constraint, LLM2Prune automatically encodes the cost-benefit trade-off imposed by the constraint: for MaxCov it selects *coverage per cost*, and for IM it selects *degree-to-weight ratio* and *in-out degree difference*, capturing both cost-efficiency and directional propagation asymmetry. These features would require significant domain expertise to design manually, yet LLM2Prune discovers them automatically from the task description alone. Full feature lists are provided in Appendix B.

Baselines and Prior Methods. We compare our algorithm with methods that accelerate heuristics by pruning the ground set rather than directly optimizing the objective function. Our evaluation includes the following classical and learning-based approaches: QuickPrune, Submodular Sparsification (SS), GCOMB-P (Manchanda et al., 2020), LeNSE (Ireland & Montana, 2022), and COMBHelper (Tian et al., 2024). Details of each baseline are provided in Appendix C. For the general knapsack constraint, prior methods do not generalize, except for QuickPrune, so we compare our algorithm only with QuickPrune.

Heuristics. For both size and knapsack constraints, we follow (Nath & Kuhnle, 2025): for MaxCover and MaxCut, we use Standard Greedy (Nemhauser et al., 1978); for IM, we adopt IMM (Tang et al., 2015); and under the knapsack constraint, we employ Khuller et al. (1999) for IM and MaxCover, and Pham et al. (2023) for MaxCut.

Datasets. We evaluate our approach using real-world datasets from the Stanford Large Network Dataset Collection (Leskovec & Sosič, 2016) for the CO problems on graphs, following the experimental setup of Ireland & Montana (2022). A summary of all datasets used can be found in Appendix D.

Common Setup. In all experiments, our algorithm and the baselines are trained to create a pruned ground set for CO problems with a budget of $k = 100$, following Ireland & Montana (2022). The downstream classifier, a two-layer GCN (Kipf & Welling, 2016), operates on the LLM-proposed node-level features. To extract feature-importance vectors, we employ GNNExplainer (Kipf & Welling, 2016). For each dataset, we adopt the train–test splits from Ireland & Montana (2022). We experiment with two LLMs, GPT-5-nano and LLaMA-3.3-70B-Instruct, to propose features, generate code, and summarize feedback. We report results using GPT-5-nano in the main paper. The two models achieve effectively comparable performance (as shown in Figure 6).

Since running beam search for features on each instance would be costly, and our main motivation is to reduce computation, we instead perform feature-space search for each problem using the Holme–Kim random graph model (Holme & Kim, 2002). We select this model because it closely resembles large-scale social networks: it exhibits both the scale-free degree distribution and the high clustering coefficient that are empirically observed in real-world social graphs (Holme & Kim, 2002). After identifying the feature set, we fine-tune the downstream classifier for each instance. Appendix E provides details on the network architecture, training and data generation hyperparameters, LLM settings, beam search, and baseline configurations.

4.1 Evaluation on Size Constraint

In this subsection, we first present our experiments under the size-constraint setting. Under this setting, each element in the ground set has a unit cost. All our baselines except QuickPrune, which works for both size and knapsack constraints, are designed specifically for the size constraint.

Result Analysis. From Table 1, we observe that LLM2Prune performs competitively with classical methods such as QuickPrune and SS. However, both approaches rely on sequential submodularity-based pruning, which is orders of magnitude slower than our method on large datasets such as YouTube and Skitter (as shown in Figure 3). In contrast, LLM2Prune achieves comparable or better performance without relying on hand-crafted features, demonstrating that learning-based pruning can be both effective and efficient.

Within the learning-based category, LLM2Prune outperforms prior methods in both combined metric and runtime. GCOMB-P and COMBHelper frequently underperform, while LeNSE is more stable but generally results in lower combined-metric values. Unlike other learning-based methods that depend on handcrafted features, LLM2Prune balances efficiency with strong performance, effectively narrowing the gap with classical methods.

Runtime Analysis. Figure 3 reports the runtime of the four strongest algorithms across all applications. Classical methods are generally slower: SS is consistently the slowest, while QuickPrune is faster but scales poorly on large graphs. Among learning-based methods, LLM2Prune is orders of magnitude faster than LeNSE while maintaining higher solution quality, striking a strong balance between scalability and accuracy. Compared to other learning-based baselines, it also achieves the most consistent performance across applications. We report the absolute runtime analysis, including an inference-time breakdown by component (see Appendix G.3).

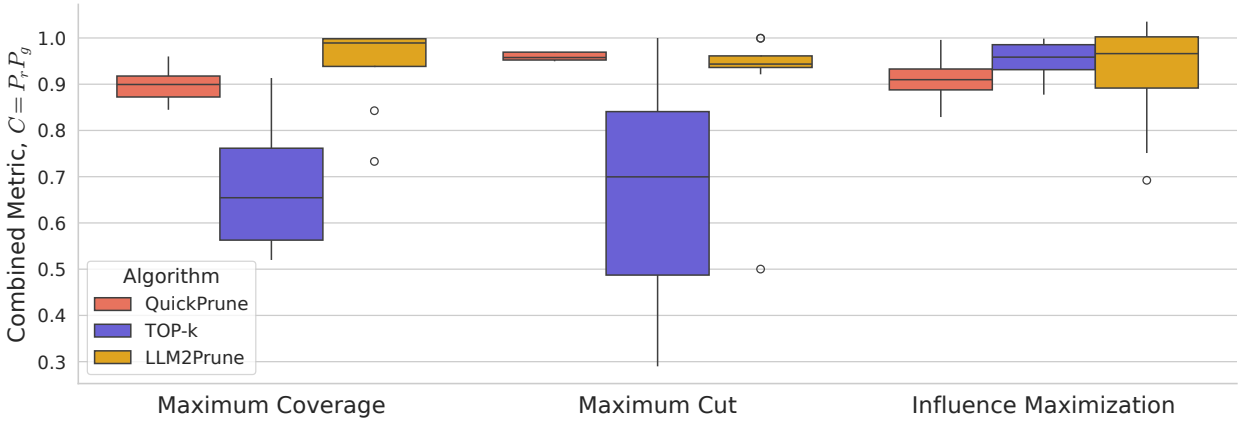

Figure 4: Combined metric for LLM2Prune, QuickPrune, and TOP-k under the knapsack constraint across MaxCov, MaxCut, and IM. LLM2Prune consistently outperforms TOP-k, matches or slightly exceeds QuickPrune on MaxCov and IM, and is comparable to QuickPrune on MaxCut.

Multi-budget Scores. Following Ireland & Montana (2022), we evaluate whether heuristics can also achieve close-to-optimal results for budgets lower than the one used to train the classifier. In our experiments, the classifier is trained to identify a pruned ground set for a budget of 100. We then test whether the pruned ground set returned by the classifier can also be used for budgets below 100. We observe that the ground set provided by LLM2Prune remains optimal for budgets lower than the training budget in nearly all instances. We report the multi-budget scores (see Appendix G.4). The results are qualitatively similar to our experiments under the knapsack constraint.

4.2 Evaluation on Knapsack Constraint

For the knapsack constraint, we adopt the cost model introduced by Nath & Kuhnle (2025); Yaroslavtsev et al. (2020). Let $G = (V, E)$ be a graph with vertex set V and edge set E. For each node $v \in V$, let $N(v) = \{u \in V : (u, v) \in E\}$ denote its neighborhood. We define the cost of node v as $c(v) = \frac{\beta}{|V|}\big(|N(v)| - \alpha\big)$, where $\alpha = \frac{1}{20}$ is a fixed offset parameter, and $\beta > 0$ is a normalization factor chosen such that $c(v) \geq 1$ for all $v \in V$. This cost model is more realistic for real-world graphs, where some nodes are inherently more expensive, for example, hubs in social networks or highly connected routers in communication networks. In contrast, a uniform cost unrealistically treats them as cheaply as sparsely connected nodes, which biases solutions toward hubs and can trivialize the optimization.

Result Analysis. As stated above, only QuickPrune can be adopted to the knapsack constraint, so we compare our algorithm with QuickPrune. We also compare it with another heuristic, TOP-k, following Nath & Kuhnle (2025).

Figure 4 shows that LLM2Prune outperforms TOP-k across all applications, and outperforms QuickPrune for MaxCov and IM, while slightly lagging behind in MaxCut. We highlight that, unlike other learning-based methods, LLM2Prune can easily adapt to the knapsack constraint by simply modifying the task description provided to the model.

Limitations and Future Directions. A main limitation of LLM2Prune is the reliance on beam search, which is more computationally expensive than a simple feedback loop. As shown in Appendix F, simple feedback loops often fail to recover when early iterations contain reasoning errors, whereas beam search mitigates this issue by maintaining multiple candidate paths, at the cost of additional runtime. An important future direction is to reduce this overhead by leveraging reasoning tokens or similar mechanisms to improve the reliability of single-path feedback loops. Such approaches could capture richer intermediate reasoning without requiring multiple branches, thereby achieving the stability of beam search with significantly lower computational cost. Beyond computational cost, LLM2Prune is also sensitive to the choice of synthetic

proxy used during feature discovery. When this proxy is structurally mismatched to the target graph family (for instance, Erdős–Rényi graphs, which have homogeneous degree distributions and lack clustering or community structure), the discovered features do not transfer well to real-world graphs. LLM2Prune therefore generalizes within a structural family but degrades under cross-family distribution shift.

5 Conclusion

We present LLM2Prune, a general framework for accelerating heuristics on graph optimization problems without problem-specific domain knowledge. We prompt the LLM to propose candidate feature sets, and, guided by performance evaluation and feature importance from an explainer algorithm, a downstream classifier learns to identify nodes that can be safely pruned without degrading solution quality. Standard heuristics are then applied to the reduced search space to obtain the final solution. Across real-world datasets for diverse CO problems over graphs under both size and knapsack constraints, LLM2Prune consistently outperforms prior learning-based methods and remains competitive with classical approaches. Moreover, the framework adapts easily to new constraints, unlike other learning-based algorithms, and can even surpass classical methods tailored to those constraints. In addition, LLM2Prune achieves substantial runtime reductions on large-scale graphs. It preserves near-optimal solution quality even when pruning over 90% of the ground set. These results highlight both the efficiency and reliability of the approach.

References

James Fitzpatrick, Deepak Ajwani, and Paula Carroll. Learning to sparsify travelling salesman problem instances. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 410–426. Springer, 2021.

Marco Grassia, Juho Lauri, Sourav Dutta, and Deepak Ajwani. Learning multi-stage sparsification for maximum clique enumeration. *arXiv preprint arXiv:1910.00517*, 2019.

Sungwon Han, Jinsung Yoon, Sercan O Arik, and Tomas Pfister. Large language models can automatically engineer features for few-shot tabular learning. *arXiv preprint arXiv:2404.09491*, 2024.

Noah Hollmann, Samuel Müller, and Frank Hutter. Large language models for automated data science: Introducing caafe for context-aware automated feature engineering. *Advances in Neural Information Processing Systems*, 36:44753–44775, 2023.

Petter Holme and Beom Jun Kim. Growing scale-free networks with tunable clustering. *Physical review E*, 65(2):026107, 2002.

David Ireland and Giovanni Montana. Lense: Learning to navigate subgraph embeddings for large-scale combinatorial optimisation. In *International conference on machine learning*, pp. 9622–9638. PMLR, 2022.

Richard M Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*, pp. 219–241. Springer, 2009.

David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 137–146, 2003.

Samir Khuller, Anna Moss, and Joseph Seffi Naor. The budgeted maximum coverage problem. *Information processing letters*, 70(1):39–45, 1999.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Juho Lauri and Sourav Dutta. Fine-grained search space classification for hard enumeration variants of subset problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 2314–2321, 2019.

Juho Lauri, Sourav Dutta, Marco Grassia, and Deepak Ajwani. Learning fine-grained search space pruning and heuristics for combinatorial optimization. *Journal of Heuristics*, 29(2):313–347, 2023.

Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1–20, 2016.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024.

Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. *Advances in Neural Information Processing Systems*, 33:20000–20011, 2020.

Jaehyun Nam, Kyuyoung Kim, Seunghyuk Oh, Jihoon Tack, Jaehyung Kim, and Jinwoo Shin. Optimized feature generation for tabular data via llms with decision tree reasoning. *Advances in Neural Information Processing Systems*, 37:92352–92380, 2024.

Ankur Nath and Alan Kuhnle. Theoretically grounded pruning of large ground sets for constrained, discrete optimization. In Yingzhen Li, Stephan Mandt, Shipra Agrawal, and Emtiyaz Khan (eds.), *Proceedings of The 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pp. 3664–3672. PMLR, 03–05 May 2025. URL `https://proceedings.mlr.press/v258/nath25a.html`.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14:265–294, 1978.

Canh V Pham, Tan D Tran, Dung TK Ha, and My T Thai. Linear query approximation algorithms for non-monotone submodular maximization under knapsack constraint. *arXiv preprint arXiv:2305.10292*, 2023.

Yuan Sun, Andreas Ernst, Xiaodong Li, and Jake Weiner. Generalization of machine learning for problem reduction: a case study on travelling salesman problems. *Or Spectrum*, 43(3):607–633, 2021.

Youze Tang, Yanchen Shi, and Xiaokui Xiao. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pp. 1539–1554, 2015.

Hao Tian, Sourav Medya, and Wei Ye. Combhelper: A neural approach to reduce search space for graph combinatorial problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 20812–20820, 2024.

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

Grigory Yaroslavtsev, Samson Zhou, and Dmitrii Avdiukhin. "bring your own greedy"+max: Near-optimal 1/2-approximations for submodular knapsack. In Silvia Chiappa and Roberto Calandra (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 3263–3274. PMLR, 26–28 Aug 2020. URL `https://proceedings.mlr.press/v108/yaroslavtsev20a.html`.

Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019.

Tianping Zhang, Zheyu Aqa Zhang, Zhiyuan Fan, Haoyan Luo, Fengyuan Liu, Qian Liu, Wei Cao, and Li Jian. Openfe: Automated feature generation with expert-level performance. In *International Conference on Machine Learning*, pp. 41880–41901. PMLR, 2023.

Xinhao Zhang, Jinghan Zhang, Banafsheh Rekabdar, Yuanchun Zhou, Pengfei Wang, and Kunpeng Liu. Dynamic and adaptive feature generation with llm. *arXiv preprint arXiv:2406.03505*, 2024.

Tianyi Zhou, Hua Ouyang, Jeff Bilmes, Yi Chang, and Carlos Guestrin. Scaling submodular maximization via pruned submodularity graphs. In *Artificial Intelligence and Statistics*, pp. 316–324. PMLR, 2017.