Towards Faster Graph Partitioning via Pre-training and Inductive Inference

Meng Qin[§†], Chaorui Zhang[†], Yu Gao[†], Yibin Ding[†], Weipeng Jiang[†], Weixi Zhang[†], Wei Han[†], Bo Bai[†]
[§]Department of Computer Science & Engineering, The Hong Kong University of Science & Technology
[†]Theory Lab, Central Research Institute, 2012 Labs, Huawei Technologies Co., Ltd.

***Abstract*—Graph partitioning (GP) is a classic problem that divides the node set of a graph into densely-connected blocks. Following the IEEE HPEC Graph Challenge and recent advances in pre-training techniques (e.g., large-language models), we propose PR-GPT (Pre-trained & Refined Graph ParTitioning) based on a novel pre-training & refinement paradigm. We first conduct the *offline pre-training* of a deep graph learning (DGL) model on small synthetic graphs with various topology properties. By using the inductive inference of DGL, one can directly *generalize* the pre-trained model (with frozen model parameters) to large graphs and derive feasible GP results. We also use the derived partition as a good initialization of an efficient GP method (e.g., InfoMap) to further *refine* the quality of partitioning. In this setting, the *online generalization* and *refinement* of PR-GPT can not only benefit from the transfer ability regarding quality but also ensure high inference efficiency without re-training. Based on a mechanism of reducing the scale of a graph to be processed by the refinement method, PR-GPT also has the potential to support streaming GP. Experiments on the Graph Challenge benchmark demonstrate that PR-GPT can ensure faster GP on large-scale graphs without significant quality degradation, compared with running a refinement method from scratch. We will make our code public at https://github.com/KuroginQin/PRGPT.**

***Index Terms*—Graph Partitioning, Community Detection, Inductive Graph Inference, Pre-training & Refinement**

I. Introduction

Graph partitioning (GP), a.k.a. graph clustering [2] or disjoint community detection [3], is a classic problem that divides the node set of a graph into disjoint blocks (a.k.a. clusters or communities) with dense linkages distinct from other blocks. Since the extracted blocks may correspond to some substructures of real-world complex systems (e.g., functional groups in protein-protein interactions), many network applications (e.g., parallel task assignment [4], Internet traffic classification [5], and protein complex detection [6]) are formulated as GP.

GP on large-scale graphs is difficult but essential as it is usually formulated as several NP-hard combinatorial optimization problems (e.g., modularity maximization [7]). The IEEE HPEC Graph Challenge [8] provides a competitive benchmark to evaluate both quality and efficiency of a GP method and has attracted a series of solutions (e.g., incremental LOBPCG for spectral clustering [9], Kalman filter [10] and data batching [11] for stochastic block partitioning, as well as fast randomized graph embedding [12]).

This paper serves as an extension of our prior work [1]. It introduces a modified method and reports new results evaluated on the IEEE HPEC Graph Challenge benchmark.

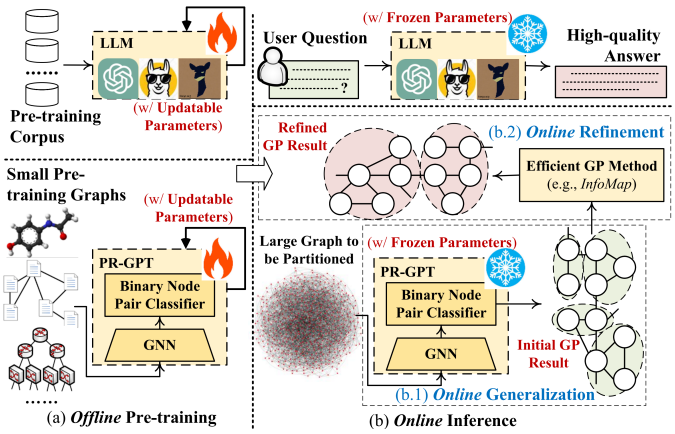


Fig. 1: An overview about the applications of (i) foundation models (e.g., LLMs) and (ii) our PR-GPT method, including the (a) *offline pre-training* and (b) *online inference*. The inference of PR-GPT includes the (b.1) *online generalization* and (b.2) *online refinement*.

In this study, we explore the potential of deep graph learning (DGL) to obtain a better trade-off between the quality and efficiency of GP. Inspired by recent advances in foundation models (e.g., LLMs [13]) and pre-training techniques [14], [15], we propose PR-GPT (Pre-trained & Refined Graph ParTitioning), a modification of our prior method [1], with an overview shown in Fig. 1. It follows a pre-training & refinement paradigm including the (i) *offline pre-training*, (ii) *online generalization*, and (iii) *online refinement*.

Assume that one has enough time to prepare a well-trained DGL model in an offline way. We first pre-train the PR-GPT model (i.e., *offline pre-training*) on a set of small graphs $\{G_t\}$ (e.g., less than 5K nodes) that cover various topology properties (e.g., node degrees and block sizes). After that, we directly generalize the pre-trained model (with frozen model parameters) to large graphs $\{G'\}$ (e.g., more than 1M nodes) via inductive inference [16] and derive feasible GP results $\{C'\}$ without re-training (i.e., *online generalization*). In existing pre-training techniques [14], [17], the online generalization after pre-training usually provides an initialization of model parameters, which are further fine-tuned w.r.t. different tasks. Inspired by this motivation, we treat C' as a good initialization of an efficient GP method (e.g., *InfoMap* [18]) and adopt its output $\bar{C}'$ as a refined version of C' (i.e., *online refinement*).

Note that the application of PR-GPT is analogous to that of LLMs. For instance, users benefit from the online inference of ChatGPT, which can generate high-quality answers in just a

few seconds but do not need to train an LLM from scratch using a great amount of resources. The *online generalization* and *refinement* of PR-GPT can benefit from inductive inference, which transfers the ability to derive high-quality GP results from pre-training data to new unseen graphs while ensuring high inference efficiency without re-training. Experiments on the Graph Challenge benchmark demonstrate that PR-GPT can achieve faster GP without significant quality degradation, compared with running a refinement method from scratch.

The major contributions of this paper beyond our prior work [1] are summarized as follows.

- We introduce PR-GPT, an advanced modification of our prior method, with fewer model parameters and better scalability. In contrast, directly applying our prior method to some large graphs in our experiments (e.g., more than 40M edges) results in the out-of-memory exception.
- To the best of our knowledge, we are the first to submit a solution based on graph pre-training and inductive inference to the Graph Challenge benchmark, which involves the GP on synthetic graphs with various scales, topology properties, and ground-truth. Whereas, our prior method was only evaluated on static graphs without ground-truth.
- We also explore the ability of PR-GPT to support streaming GP while our prior work only considers static GP.

II. Problem Statement & Preliminaries

In general, a graph G can be represented as a tuple (V, E), where $V := \{v_1, v_2, \cdots, v_N\}$ and $E := \{(v_i, v_j) | v_i, v_j \in V\}$ are the sets of nodes and edges. One can use an adjacency matrix $\mathbf{A} \in \{0,1\}^{N \times N}$ to describe the topology structure of G, where $\mathbf{A}_{ij} = \mathbf{A}_{ji} = 1$ if $(v_i, v_j) \in E$ and $\mathbf{A}_{ij} = \mathbf{A}_{ji} = 0$ otherwise. We adopt the problem statement of IEEE HPEC Graph Challenge [19] and study the following GP problem.

Graph Partitioning (GP). Given a graph G, (static) GP aims to partition the node set V into K disjoint subsets $C := (C_1, \cdots, C_K)$ (i.e., blocks or communities) s.t. (i) the linkage within each block is dense but (ii) that between blocks is loose. We consider the challenging K-agnostic GP, where the number of blocks K is unknown. Namely, one should simultaneously determine K and the corresponding block partition C.

Streaming GP. Graph Challenge provides two models to simulate streaming GP. We consider the more challenging snowball model and leave the extension to emerging edges in future work. Given a graph G, the snowball model divides V into T disjoint subsets $(V_1, \cdots, V_T)$, with V_t as the set of newly added nodes in the t-th step. Let $\bar{V}_t := \cup_{s=1}^{t} V_s$ and $\bar{E}_t$ be the set of cumulative edges induced by $\bar{V}_t$. For each step t, streaming GP requires to derive a K-agnostic block partition based on the cumulative topology $(\bar{V}_t, \bar{E}_t)$.

Modularity Maximization. GP can be formulated as the combinatorial optimization objective of modularity maximization [7]. Given G and K, it aims to obtain a partition C that maximizes the modularity metric:

$$\max_{C} \ \mathrm{Mod}(G, K) := \frac{1}{2|E|} \sum_{r=1}^{K} \sum_{v_i, v_j \in C_r} [\mathbf{A}_{ij} - \frac{d_i d_j}{2|E|}], \quad (1)$$

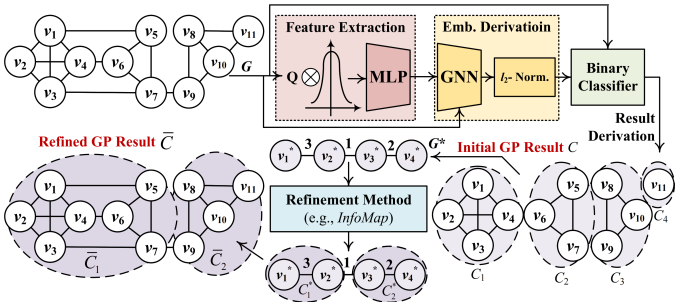


Fig. 2: Model architecture and inference procedure of PR-GPT.

where $d_i := \sum_i \mathbf{A}_{ij}$ is the degree of node v_i; $|E|$ is the number of edges. One can rewrite (1) into the matrix form:

$$\min_{\mathbf{H}} -\mathrm{tr}(\mathbf{H}^T \mathbf{Q} \mathbf{H}) \text{ s.t. } \mathbf{H}_{ir} = \begin{cases} 1, v_i \in C_r \\ 0, \text{otherwise} \end{cases}, \quad (2)$$

where $\mathbf{Q} \in \mathbb{R}^{N \times N}$ is defined as the modularity matrix with $\mathbf{Q}_{ij} := [\mathbf{A}_{ij} - d_i d_j / (2|E|)]$; $\mathbf{H} \in \{0,1\}^{N \times K}$ indicates the block membership C.

Pre-training & Refinement. As shown in Fig. 1, the pre-training & refinement paradigm includes the (i) *offline pre-training*, (ii) *online generalization*, and (iii) *online refinement*. For simplicity, we denote a DGL model as $C = f(G; \theta)$, which derives a block partition C given a graph G, with θ as the set of model parameters to be learned.

In *offline pre-training*, we generate a set of small graphs $\Gamma = \{G_1, \cdots, G_M\}$ (e.g., less than 5K nodes) using the generator of Graph Challenge. The generation of each $G_s \in \Gamma$ includes its topology $(V^{(s)}, E^{(s)})$ and corresponding block partition $C^{(s)}$. We then pre-train f (e.g., iteratively updating θ) based on $\{(V^{(s)}, E^{(s)})\}$ and $\{C^{(s)}\}$ in an offline way.

After that, we generalize f to new large graphs $\{G'\}$ (e.g., more than 1M nodes) with frozen θ in *online generalization*. Based on the inductive inference of f, one can directly derive a feasible block partition C' for G' (e.g., via only one feed-forward propagation (FFP) of f) without re-training. Inspired by existing pre-training and fine-tuning paradigm, we introduce *online refinement*, where the derived partition C' is used as a good initialization of an efficient K-agnostic GP method (e.g., *InfoMap*) to further refine the quality of C'.

Evaluation Protocol. Our evaluation is consistent with the application of foundation models as illustrated in Fig. 1. Concretely, one can get high-quality answers from an LLM in just a few seconds during its *online inference* phase and does not need to train it from scratch. Assume that we have enough time to prepare a well-trained f during the *offline pre-training*, which is usually a one-time effort. Our evaluation focuses on the *online generalization* and *refinement* w.r.t. the GP on large graphs $\{G'\}$. In contrast, we have to run most existing methods on $\{G'\}$ from scratch, as they are inapplicable to inductive graph inference and thus cannot benefit from pre-training [16].

III. Methodology

In this section, we detail our PR-GPT method. Fig. 2 gives an overview of the model architecture and inference procedure.

A. Model Architecture

Similar to our prior method [1], PR-GPT reformulates GP as the binary node pair classification and follows a GNN-based end-to-end architecture. An auxiliary variable $\mathbf{S} \in \{0,1\}^{N\times N}$ is introduced to represent the binary classification result, where $\mathbf{S}_{ij} = \mathbf{S}_{ji} = 1$ if nodes (v_i, v_j) are in the same block and $\mathbf{S}_{ij} = \mathbf{S}_{ji} = 0$ otherwise. The inference of PR-GPT only considers $\{\mathbf{S}_{ij}\}$ w.r.t a small set of node pairs $P = \{(v_i, v_j)\}$ ($|P| \ll N^2$), which are rearranged as a vector $\mathbf{y} \in \{0,1\}^{|P|}$. We let $\mathbf{y}_s = \mathbf{S}_{ij} = \mathbf{S}_{ji}$ for each $p_s = (v_i, v_j) \in P$.

*1) **Feature Extraction**:* Let k be the dimensionality of node embedding or features with $k \ll N$. PR-GPT first uses the following feature extraction module to extract community-preserving features, arranged as a matrix $\tilde{\mathbf{X}} \in \mathbb{R}^{N\times k}$, from the modularity maximization objective (2):

$$\tilde{\mathbf{X}} = \mathrm{MLP}(\mathbf{X}) \text{ and } \mathbf{X} = \mathbf{Q}\mathbf{\Omega} = \mathbf{A}\mathbf{\Omega} - \mathbf{d}(\mathbf{d}^T\mathbf{\Omega}), \quad (3)$$

where $\mathbf{\Omega} \in \mathbb{R}^{N\times k}$ is a random matrix with $\mathbf{\Omega}_{ir} \sim \mathcal{N}(0, 1/k)$; $\mathbf{d} := [d_1, \cdots, d_N]^T \in \mathbb{Z}^N$ is a vector about node degrees. It applies the Gaussian random projection [20], an efficient dimension reduction technique with rigorous bounds w.r.t. information loss, to the modularity matrix $\mathbf{Q}$ in (2) that encodes key characteristics regarding implicit community structures.

Note that $\mathbf{Q} \in \mathbb{R}^{N\times N}$ is usually a dense matrix. Directly applying the random projection causes a complexity of $O(N^2k)$ intractable for large graphs. To reduce the complexity, our prior method introduces a sparsified matrix $\tilde{\mathbf{Q}} \in \mathbb{R}^{N\times N}$, where $\tilde{\mathbf{Q}}_{ij} = \mathbf{Q}$ if $(v_i, v_j) \in E$ and $\tilde{\mathbf{Q}}_{ij} = 0$ otherwise, and applies the random projection to $\tilde{\mathbf{Q}}$. However, $\tilde{\mathbf{Q}}$ may lose some information encoded in $\mathbf{Q}$. Instead of using $\tilde{\mathbf{Q}}$, PR-GPT still applies random projection to $\mathbf{Q}$ but follows the multiplication order described in (3), which can reduce the complexity from $O(N^2k)$ to $O(|E|k + Nk)$.

We also apply a multi-layer perceptron (MLP) to the reduced features $\mathbf{X} \in \mathbb{R}^{N\times k}$, which leverages additional non-linearity for feature extraction.

*2) **Embedding Derivation**:* The extracted features $\tilde{\mathbf{Z}}$ are fed into a multi-layer GNN, which further derives community-preserving embeddings. Inspired by SGC [21], we remove all the learnable model parameters and non-linearity of GCN [22] in our prior method, which improves the scalability. Assume that there are L GNN layers. PR-GPT derives embedding representations, arranged as $\tilde{\mathbf{Z}} \in \mathbb{R}^{N\times k}$, via

$$\tilde{\mathbf{Z}} = \mathrm{LN}(\mathbf{Z}) \text{ and } \mathbf{Z} = \tilde{\mathbf{A}}^L\tilde{\mathbf{X}}, \quad (4)$$

where $\mathrm{LN}(\mathbf{Z})$ is the row-wise l_2-normalization of the embedding matrix $\mathbf{Z} \in \mathbb{R}^{N\times k}$ (i.e., $\mathbf{Z}_{i,:} \leftarrow \mathbf{Z}_{i,:}/|\mathbf{Z}_{i,:}|_2$); $\tilde{\mathbf{A}} := \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$; $\hat{\mathbf{A}} := \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix with self-edges; $\hat{\mathbf{D}}$ is the degree diagonal matrix of $\hat{\mathbf{A}}$.

One can formulate the l-th GNN layer as $\mathbf{Z}^{(l)} = \tilde{\mathbf{A}}\mathbf{Z}^{(l-1)}$, with $\mathbf{Z}^{(0)} := \tilde{\mathbf{X}}$. $\mathbf{Z}^{(l)}_{i,:}$ is the intermediate representation of node v_i. It is also the weighted mean over features w.r.t. $\{v_i\} \cup n(v_i)$, with $n(v_i)$ as the neighbor set of v_i. This mechanism further enhances the ability of $\{\mathbf{Z}^{(l)}\}$ to capture community structures, since it forces nodes (v_i, v_j) with similar neighbors $(n(v_i), n(v_j))$ (i.e., dense local linkage) to have similar representations $(\mathbf{Z}^{(l)}_{i,:}, \mathbf{Z}^{(l)}_{j,:})$.

Let $\tilde{\mathbf{z}}_i := \tilde{\mathbf{Z}}_{i,:}$ be the embedding of v_i. The l_2-normalization ensures that $|\tilde{\mathbf{z}}_i| = 1$ and thus $|\tilde{\mathbf{z}}_i - \tilde{\mathbf{z}}_j|^2 = 2 - 2\mathbf{z}_i\mathbf{z}_j^T$.

*3) **Binary Node Pair Classification**:* Given a node pair (v_i, v_j), PR-GPT adopts the following binary classifier same as our prior method to estimate the classification result $\mathbf{S}_{ij}$ using corresponding embeddings $(\tilde{\mathbf{z}}_i, \tilde{\mathbf{z}}_j)$:

$$\begin{aligned}&\hat{\mathbf{S}}_{ij} = \exp(-|\tilde{\mathbf{z}}_i - \tilde{\mathbf{z}}_j|^2\tau_{ij}) = \exp(2\tau_{ij}\cdot(\tilde{\mathbf{z}}_i\tilde{\mathbf{z}}_j^T - 1)),\\ &\text{with } \tau_{ij} = g_s(\tilde{\mathbf{z}}_i)g_d(\tilde{\mathbf{z}}_j),\end{aligned} \quad (5)$$

where $\hat{\mathbf{S}}_{ij} \in [0,1]$ denotes the estimation of $\mathbf{S}_{ij}$; g_s and g_d are two MLPs with the same configurations; τ_{ij} is a pair-wise parameter determined by $(\tilde{\mathbf{z}}_i, \tilde{\mathbf{z}}_j)$. Namely, $\{\mathbf{S}_{ij}\}$ is estimated via a combination of the (i) Euclidean distance and (ii) inner product w.r.t. corresponding embeddings $\{(\tilde{\mathbf{z}}_i, \tilde{\mathbf{z}}_j)\}$.

*4) **Result Derivation**:* Given a graph $G = (V, E)$, the overall procedure to derive a feasible GP result is summarized in Algorithm 1. Fig. 2 also gives a running example.

Algorithm 1: Result Derivation Given a Graph

Input: input graph $G = (V, E)$
Output: a feasible GP result C w.r.t. G
1 derive $\hat{\mathbf{y}}$ w.r.t. E via one FFP of the model
2 $\tilde{E} \leftarrow \emptyset$ //Initialize edge set of auxiliary graph $\tilde{G}$
3 **for each** node pair (i.e., edge) $e_s = (v_i, v_j) \in E$ **do**
4 **if** $\hat{\mathbf{y}}_s > 0.5$ **then**
5 add $e_s = (v_i, v_j)$ to $\tilde{E}$
6 extract connected components of $\tilde{G}$ via DFS/BFS on $\tilde{E}$
7 treat each component as a block to form C

In line 1, we derive $\{\hat{\mathbf{S}}_{ij}\}$ for all the node pairs (i.e., edges) in the edge set E of G and rearrange them as a vector $\hat{\mathbf{y}}$. In lines 2-5, an auxiliary graph $\tilde{G} = (V, \tilde{E})$ is constructed based on $\hat{\mathbf{y}}$. $\tilde{G}$ shares the same node set V as G but has a different edge set $\tilde{E}$. Concretely, we add $e_s = (v_i, v_j)$ to $\tilde{E}$ if $\hat{\mathbf{y}}_s > 0.5$. After that, the constructed $\tilde{G}$ may contain multiple connected components (e.g., $\{v_1, v_2, v_3, v_4\}$ and $\{v_5, v_6, v_7\}$ in Fig. 2). In particular, edges $\{e_s\}$ in the same component are with high values $\{\hat{\mathbf{y}}_s\}$. It indicates that the associated nodes are more likely to be partitioned into the same block. In lines 6-7, we extract all the connected components of $\tilde{G}$ via the DFS/BFS on $\tilde{E}$ and treat each component as a feasible block.

In addition to E, our prior method also sampled other node pairs (e.g., (v_5, v_8) in Fig. 2) when constructing $\tilde{G}$. PR-GPT removes this sampling procedure to further improve efficiency.

Our experiments demonstrate that even with the aforementioned simplifications, PR-GPT can still achieve impressive GP quality on the Graph Challenge benchmark.

B. Offline Pre-training

PR-GPT adopts the same setup of *offline pre-training* as our prior method. We first generate a set of small pre-training graphs $\Gamma = \{G_1, \cdots, G_M\}$ using the standard generator of Graph Challenge. Instead of fixing generator parameters, we

Algorithm 2: *Online Generalization* and *Refinement*

Input: a large graph $G' = (V', E')$ to be partitioned
Output: a refined GP result $\bar{C}'$ w.r.t. G'
1 get input features $\tilde{\mathbf{X}}'$ w.r.t. $\mathcal{G}'$ via (3)
2 get initial GP result C' w.r.t. G' via Algorithm 1
3 construct weighted super-graph G^* based on C'
4 get refined $\bar{C}'$ via a refinement method w/ G^* as input

simulate various properties (e.g., distributions of degrees and block sizes) by sampling these parameters from certain distributions, which can increase the diversity of pre-training data. The pre-training of PR-GPT combines the (i) unsupervised modularity maximization and (ii) supervised binary cross-entropy objectives for each graph $G_t \in \Gamma$. Due to space limits, we omit details of pre-training data generation and pre-training algorithm, which can be found in our prior work [1] and code.

C. Online Inference for Static GP

As described in Algorithm 2, the *online inference* of PR-GPT includes the *online generalization* (i.e., lines 1-2) and *online refinement* (i.e., lines 3-4).

1) ***Online Generalization****:* After the *offline pre-training*, we can generalize PR-GPT to a large graph G' with frozen parameters θ and derive a feasible partition C' via Algorithm 1.

2) ***Online Refinement****:* We further treat the derived C' as a good initialization of an existing K-agnostic GP method and apply this method to refine C'. Concretely, the initialization can be in the form of a weighted super-graph G^* w.r.t. C', where we merge nodes in each block as a super-node (e.g., $v_1^* = C_1 = \{v_1, v_2, v_3, v_4\}$ in Fig. 2) and set the number of between-block edges as the weight of corresponding super-edge (e.g., 3 for (v_1^*, v_2^*) in Fig. 2). We use G^* as the input of a K-agnostic GP method that can handle weighted graphs (e.g., *InfoMap* [18] and *Locale* [23] in our experiments) and derive a GP result C^* w.r.t. G^*. C^* is then recovered to a partition $\bar{C}'$ w.r.t. G', which is a refined version of C'.

Compared with running a refinement method on G' from scratch, *online refinement* may be much more efficient, since it reduces the number of nodes to be processed (e.g., reducing 11 nodes to 4 super-nodes in Fig. 2). Therefore, PR-GPT has the potential to achieve faster GP w.r.t. the refinement method.

D. Extension to Streaming GP

The *online generalization* and *refinement* of PR-GPT shares a motivation similar to that of some streaming GP approaches [9], [19], where the partition of current step provides an **initialization** for next step. In each step, these streaming approaches incrementally update their GP results to **avoid running the base algorithm from scratch**. Similarly, PR-GPT tries to achieve faster GP compared with a refinement method by reducing the number of nodes to be processed. Due to these similar motivations, we believe that PR-GPT has the potential to support streaming GP.

As stated in Section II, we consider the snowball model, where new nodes are added in each step. The inductive inference of DGL allows RP-GPT to directly generalize the pre-trained model parameters to new topology of each step without re-training. Our analysis about inference time (see Table VII) shows that *online refinement* is the major bottleneck of PR-GPT. We adopt a naive strategy to directly run the non-bottleneck *online generalization* procedure from scratch in each step. Our experiments demonstrate that even without incremental updating for *online generalization*, PR-GPT can still obtain efficiency improvement for streaming GP, which is consistent with some previous work [19], based on its mechanism of reducing the scale of a graph to be processed.

TABLE I: Detailed Statistics of the Generated Benchmark Datasets

N	$\lvert E\rvert$	K	**Degrees**	**Density**
10K	402K-449K	25	6-194	8×10^{-3}
50K	2.02M-2.03M	44	8-246	2×10^{-3}
100K	4.05M-4.07M	56	6-242	8×8^{-4}
500K	20.3M-20.3M	98	4-230	1×10^{-4}
1M	40.6M-40.7M	125	4-264	8×10^{-5}

E. Complexity Analysis

To derive embeddings $\{\tilde{\mathbf{z}}_i\}$, the complexities of (i) feature extraction described in (3) and (ii) one FFP of GNN defined in (4) are $O(|E|k + Nk)$ and $O(|E|Lk + Nk)$, with L as the number of GNN layers. The complexities of (iii) one FFP of binary classifier (5) to construct $\tilde{G}$ and (iv) extracting connected components of $\tilde{G}$ via DFS/BFS are $O(|E|k^2)$ and $O(N + |\tilde{E}|)$, where $|\tilde{E}|$ is the number of edges in $\tilde{G}$, with $|\tilde{E}| \leq |E|$. Therefore, the overall complexity of *online generalization* is no more than $O(|E|k(L + k) + Nk)$, with $k, L \ll N, |E|$.

The complexity of *online refinement* depends on the concrete refinement method (e.g., *InfoMap*), which is usually efficient. Note that any improvement regarding the efficiency of a refinement method (e.g., better parallel implementations) can also further improve the efficiency of PR-GPT.

IV. Experiments

A. Experiment Setups

We followed standard setups of the IEEE HPEC Graph Challenge benchmark [8] to validate the effectiveness of PR-GPT for both static and streaming GP.

1) ***Datasets****:* We considered the hardest setting with the (i) ratio between the number of within-block edges and between edges and (ii) block size heterogeneity set to 2.5 and 3. The standard generator of Graph Challenge was used to generate graphs with different scales, where we respectively set the number of nodes N to 10K, 50K, 100K, 500K, and 1M. For each setting, we independently generated five graphs and reported corresponding average evaluation results. Table I summarizes statistics of the generated datasets, where $|E|$ and K denote the numbers of edges and blocks.

2) ***Baselines****:* We compared PR-GPT with seven baselines that can tackle K-agnostic GP, including (i) *MC-SBM* [24], (ii) *Par-SBM* [25], (iii) *Louvain* [26], (iv) *RaftGP-C* [12], (v) *RaftGP-M* [12], (vi) *InfoMap* [18], and (vii) *Locale* [23].

TABLE II: Evaluation Results of Static GP with N=10K

Methods	Time↓(s)	AC↑(%)	ARI↑(%)	F1↑(RCL, PCN)(%)
MC-SBM	217.06	99.32	99.51	99.53 (99.17, 99.91)
Par-SBM	2.34	99.89	99.97	99.97 (99.97, 99.98)
Louvain	3.99	94.68	95.71	95.94 (99.90, 92.38)
RaftGP-C	11.01	99.32	99.15	99.20 (99.86, 98.56)
RaftGP-M	10.34	99.27	99.09	99.14 (99.86, 98.45)
InfoMap	1.18	99.89	99.96	99.96 (99.97, 99.96)
PR-GPT(IM)	**0.80**	**99.94**	99.96	99.96 (99.98, 99.95)
Improv.	+32.2%	+0.05%	+0.0%	+0.0%
Locale	3.20	95.15	97.18	97.33 (99.96, 94.86)
PR-GPT(Lcl)	**1.56**	**96.81**	**98.08**	**98.19** (99.94, 96.50)
Improv.	+51.3%	+1.7%	+0.9%	+0.9%

TABLE III: Evaluation Results of Static GP with N=50K

Methods	Time↓(s)	AC↑(%)	ARI↑(%)	F1↑(RCL, PCN)(%)
MC-SBM	2553.67	99.33	99.19	99.22 (98.51, 99.97)
Par-SBM	19.77	99.00	99.34	99.36 (99.98, 98.76)
Louvain	29.13	83.60	72.43	73.51 (99.91, 59.00)
RaftGP-C	63.91	99.18	98.52	98.57 (99.91, 97.29)
RaftGP-M	63.16	99.00	98.16	98.22 (99.89, 96.66)
InfoMap	11.67	99.88	99.96	99.96 (99.97, 99.94)
PR-GPT(IM)	**6.61**	99.63	99.63	99.64 (99.94, 99.35)
Improv.	+43.6%	-0.3%	-0.3%	-0.3%
Locale	23.75	93.97	94.44	94.62 (99.96, 89.92)
PR-GPT(Lcl)	**13.44**	**94.49**	94.41	94.59 (99.90, 89.85)
Improv.	+43.4%	+0.6%	-0.03%	-0.03%

TABLE IV: Evaluation Results of Static GP with N=100K

Methods	Time↓(s)	AC↑(%)	ARI↑(%)	F1↑(RCL, PCN)(%)
MC-SBM	9240.27	99.00	98.93	98.95 (97.98, 99.99)
Par-SBM	52.68	98.97	99.62	99.63 (99.99, 99.26)
Louvain	68.03	74.08	59.99	61.28 (99.94, 44.40)
RaftGP-C	138.58	99.49	99.04	99.06 (99.93, 98.23)
RaftGP-M	133.14	99.75	99.58	99.59 (99.95, 99.24)
InfoMap	28.85	99.93	99.97	99.97 (99.97, 99.96)
PR-GPT(IM)	**16.92**	99.82	99.81	99.81 (99.85, 99.78)
Improv.	+41.4%	-0.1%	-0.2%	-0.2%
Locale	55.33	89.91	86.87	87.21 (99.97, 77.54)
PR-GPT(Lcl)	**31.79**	**90.84**	**89.79**	**90.05** (99.83, 82.12)
Improv.	+42.5%	+1.0%	+3.4%	+3.3%

In particular, *MC-SBM* is the standard baseline of Graph Challenge. *Locale* is an advanced modification of *Louvain*. *RaftGP-C* and *RaftGP-M* are two variants of the innovation award winner of Graph Challenge 2023, which are GNN-based embedding approaches without (pre-)training. We adopt *InfoMap* and *Locale* as two example refinement methods of PR-GPT and highlight the improvement in quality and efficiency w.r.t. running these refinement methods from scratch. Note that PR-GPT can also be easily extended to leverage other efficient GP approaches (e.g., *Louvain*) for *online refinement*.

*3) **Evaluation Metrics**:* We followed the evaluation criteria of Graph Challenge to adopt *accuracy* (AC), *adjusted random index* (ARI), *precision*, and *recall* as quality metrics. Given precision and recall, we also reported the corresponding *F1-score*. The *inference time* (sec) of a method was adopted as the efficiency metric, based on the evaluation protocol described in Section II. We defined that a method encounters the out-of-time exception if it cannot derive a feasible GP result within $10,000$ seconds.

*4) **Parameter & Environment Settings**:* Let k be the feature or embedding dimensionality. L_{F}, L_{GNN}, and L_{BC} denote the numbers of MLP layers in (3), GNN layers in (4), and MLP layers in (5), respectively. We set $(k, L_{\mathrm{F}}, L_{\mathrm{GNN}}, L_{\mathrm{BC}}) = (32, 2, 2, 4)$ for PR-GPT on all the datasets. Other parameter settings of PR-GTP can be checked in our code.

We used Python 3.8 to implement PR-GPT, where the model architecture and Algorithm 1 were implemented via PyTorch 1.10 and SciPy 1.10 (with BFS/DFS supported by the efficient 'scipy.sparse.csgraph.connected_components' function). Moreover, we adopted the official or widely-used (C/C++ or Python) implementations of all the baselines and tuned their parameters to report the best quality.

All the experiments were conducted on a server with one Intel 14-core CPU, one 24GB memory GPU, 45GB main memory, and Ubuntu 20.04 OS.

B. Evaluation of Static GP

The average evaluation results of static GP over five generated graphs w.r.t. each setting of the dataset are shown in Tables II, III, IV, V, and VI, where metrics of PR-GPT are in **bold** if they achieve improvement w.r.t. the corresponding refinement methods; OOT and OOM represent the out-of-time and out-of-memory exceptions; '(IM)' and '(Lcl)' indicate that *InfoMap* and *Locale* are used as the refinement method. We also report the corresponding runtime (sec) of each *online inference* step of PR-GPT in Table VII, where 'Feat', 'FFP', 'Init', and 'Refine' represent the runtime of (i) feature extraction described in (3), (ii) one FFP of the model, (iii) initial result derivation (i.e., Algorithm 1), and (iv) *online refinement*.

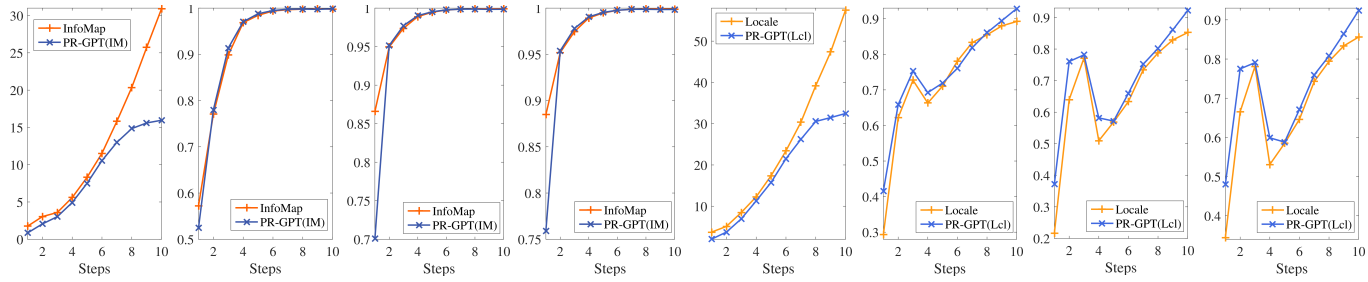
On all the datasets, two variants of PR-GPT achieve significant improvement of efficiency (e.g., more than 20%) w.r.t. the corresponding refinement methods while the quality degradation is less than 1%. In summary, PR-GPT achieves the best efficiency and is always in the top groups with the best quality. It indicates that the pre-training & refinement paradigm of PR-GPT can help ensure a better trade-off between the quality and efficiency of static GP.

Surprisingly, PR-GPT can even obtain improvement for both aspects in some cases. Compared with *InfoMap*, *Locale* is a weaker refinement method in terms of quality. PR-GPT can even ensure significant quality improvement (e.g., more than 10%) when *Locale* suffers from poor quality metrics (e.g., on datasets with $N = 500$K and 1M). The aforementioned results demonstrate that the *offline pre-training* (on historical small graphs) may also help resist noise and improve the GP quality (on new large graphs) compared with running a refinement method from scratch. In contrast, most existing GP approaches cannot benefit from pre-training and inductive inference.

According to Table VII, *online refinement* is the major bottleneck of PR-GPT, depending on the concrete refinement method. In particular, the runtime of *online refinement* is much smaller than that of running a refinement method from scratch. It verifies our motivation that the *online generalization* can derive a good initialization (i.e., a weighted super-graph G^*) with a much smaller scale (e.g., in terms of the number of nodes to be processed) and thus help achieve faster GP.

C. Evaluation of Streaming GP

As discussed in Section III-D, the *online generalization* and *refinement* of PR-GPT shares a motivation similar to some



(a) IM, Time↓ (s) (b) IM, AC↑ (c) IM, ARI↑ (d) IM, F1↑ (e) Lcl, Time↓ (s) (f) Lcl, AC↑ (g) Lcl, ARI↑ (h) Lcl, F1↑

Fig. 3: Evaluation results of streaming GP.

TABLE V: Evaluation Results of Static GP with N=500K

Methods	**Time**↓(s)	**AC**↑(%)	**ARI**↑(%)	**F1**↑(RCL, PCN)(%)
MC-SBM	OOT	OOT	OOT	OOT
Par-SBM	305.65	97.69	98.69	98.71 (99.99, 97.49)
Louvain	561.62	39.75	20.85	22.73 (99.90, 12.85)
RaftGP-C	OOM	OOM	OOM	OOM
RaftGP-M	OOM	OOM	OOM	OOM
InfoMap	245.78	99.42	99.75	99.75 (99.76, 99.74)
PR-GPT(IM)	**194.68**	99.16	98.77	98.79 (99.76, 97.91)
Improv.	+20.8%	-0.3%	-1.0%	-1.0%
Locale	464.72	61.25	38.45	39.77 (99.96, 24.96)
PR-GPT(Lcl)	**325.28**	**69.71**	**51.84**	**52.79** (99.79, 35.99)
Improv.	+30.0%	+13.8%	+34.8%	+32.7%

TABLE VI: Evaluation Results of Static GP with N=1M

Methods	**Time**↓(s)	**AC**↑(%)	**ARI**↑(%)	**F1**↑(RCL, PCN)(%)
MC-SBM	OOT	OOT	OOT	OOT
Par-SBM	761.63	98.90	99.43	99.44 (99.99, 98.89)
Louvain	1266.81	25.10	13.76	15.41 (99.87, 8.38)
RaftGP-C	OOM	OOM	OOM	OOM
RaftGP-M	OOM	OOM	OOM	OOM
InfoMap	680.75	98.64	99.35	99.36 (99.45, 99.26)
PR-GPT(IM)	**530.42**	**99.07**	**99.48**	**99.49** (99.55, 99.42)
Improv.	+22.1%	+0.4%	+0.1%	+0.1%
Locale	1216.16	49.70	24.84	26.19 (99.95, 15.13)
PR-GPT(Lcl)	**996.65**	**62.23**	**35.53**	**36.62** (99.81, 22.58)
Improv.	+18.1%	+25.2%	+43.0%	+39.8%

streaming GP approaches. We demonstrate the potential of PR-GPT to support streaming GP by comparing its quality and efficiency with that of running the corresponding refinement method from scratch in each step.

As stated in Section II, the snowball model was adopted to simulate streaming GP, where we set the number of steps to 10. The average evaluation results of streaming GP over five independent runs on the datasets with $N = 100$K are visualized in Fig. 3. In addition, Table VIII reports the corresponding variation regarding (i) the number of nodes N and (ii) the average number of nodes $\tilde{N}$ in the initialization (i.e., the weighted super-graph) given by PR-GPT.

With the increase of step, the time of running a refinement method from scratch grows linearly. For PR-GPT, the increase of inference time is slightly sub-linear, which is consistent with the evaluation of streaming GP in some previous solutions [19] submitted to Graph Challenge. PR-GPT can achieve quality close to the corresponding baselines in most steps and even obtain better quality in some cases. As shown in Table VIII, PR-GPT significantly reduces the number of nodes to be processed. In particular, it can ultimately reduce the scale of a graph by about half as the step increases. In summary, the pre-training & refinement paradigm of PR-GPT has the potential to support streaming GP.

TABLE VII: Detailed Inference Time (sec) of PR-GPT

N		**Total**	**Feat**	**FFP**	**Init**	**Refine**
10K	PR-GPT (IM)	0.80	0.01	0.03	0.51	0.25
	PR-GPT (Lcl)	1.56				1.00
50K	PR-GPT (IM)	6.61	0.04	0.16	2.91	3.49
	PR-GPT (Lcl)	13.44				10.33
100K	PR-GPT (IM)	16.92	0.08	0.33	6.45	10.06
	PR-GPT (Lcl)	31.79				24.93
500K	PR-GPT (IM)	194.68	0.41	1.69	46.64	145.94
	PR-GPT (Lcl)	325.28				276.54
1M	PR-GPT (IM)	530.42	0.80	3.34	113.14	413.13
	PR-GPT (Lcl)	996.65				879.36

TABLE VIII: Detailed Variation of N and $\tilde{N}$ in Streaming GP

Steps	1	2	3	4	5	6	7	8	9	10
N	10K	20K	30K	40K	50K	60K	70K	80K	90K	100K
$\tilde{N}$	8K	18K	27K	37K	46K	54K	60K	63K	63K	60K

V. Conclusion

In this paper, we considered K-agnostic GP and proposed PR-GPT. It follows a pre-training & refinement paradigm, including the (i) *offline pre-training* on historical small graphs as well as the (ii) *online generalization* to and (iii) *refinement* on new large graphs. We evaluated PR-GPT on the IEEE HPEC Graph Challenge benchmark, comparing its inference quality and efficiency over seven baselines. Experiments demonstrated that PR-GPT, combined with different refinement methods (e.g., *InfoMap* and *Locale*), can achieve faster GP without significant quality degradation. Surprisingly, it can even achieve improvement for both quality and efficiency in some cases. Based on a mechanism of providing a good initialization with a smaller scale (i.e., the number of nodes to be processed), PR-GPT also has the potential to support streaming GP and can obtain efficiency improvement consistent with the results of some previous work.

In this study, we only considered the snowball model of streaming GP. We plan to extend our method to another emerging edge model of Graph Challenge in our future work. Moreover, extending PR-GPT to (i) dynamic graphs [27]–[30] that has a motivation similar to streaming GP and (ii) attributed graphs with the consideration of inherent correlations between graph topology and attributes [31]–[34] are also our next research focuses.

REFERENCES

[1] M. Qin, C. Zhang, Y. Gao, W. Zhang, and D.-Y. Yeung, "Pre-train and refine: Towards higher efficiency in k-agnostic community detection without quality degradation," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 2467–2478.

[2] S. E. Schaeffer, "Graph clustering," *Computer Science Review*, vol. 1, no. 1, pp. 27–64, 2007.

[3] S. Fortunato and M. E. Newman, "20 years of network community detection," *Nature Physics*, vol. 18, no. 8, pp. 848–850, 2022.

[4] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel Computing*, vol. 26, no. 12, pp. 1519–1534, 2000.

[5] M. Qin, K. Lei, B. Bai, and G. Zhang, "Towards a profiling view for unsupervised traffic classification by exploring the statistic features and link patterns," in *Proceedings of the 2019 ACM SIGCOMM Workshop on Network Meets AI & ML*, 2019, pp. 50–56.

[6] G. Qin and L. Gao, "Spectral clustering for detecting protein complexes in protein–protein interaction (ppi) networks," *Mathematical and Computer Modelling*, vol. 52, no. 11-12, pp. 2066–2074, 2010.

[7] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences (PNAS)*, vol. 103, no. 23, pp. 8577–8582, 2006.

[8] E. Kao, V. Gadepally, M. Hurley, M. Jones, J. Kepner, S. Mohindra, P. Monticciolo, A. Reuther, S. Samsi, W. Song *et al.*, "Streaming graph challenge: Stochastic block partition," in *Proceedings of the 2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–12.

[9] D. Zhuzhunashvili and A. Knyazev, "Preconditioned spectral clustering for stochastic block partition streaming graph challenge (preliminary version at arxiv.)," in *Proceedings of 2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–6.

[10] L. J. Durbeck and P. Athanas, "Kalman filter driven estimation of community structure in time varying graphs," in *Proceedings of 2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2022, pp. 1–7.

[11] F. Wanye, V. Gleyzer, E. Kao, and W.-c. Feng, "An integrated approach for accelerating stochastic block partitioning," in *Proceedings of 2023 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2023, pp. 1–7.

[12] Y. Gao, M. Qin, Y. Ding, L. Zeng, C. Zhang, W. Zhang, W. Han, R. Zhao, and B. Bai, "Raftgp: Random fast graph partitioning," in *Proceedings of the 2023 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2023, pp. 1–7.

[13] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[14] Y. Cao, J. Xu, C. Yang, J. Wang, Y. Zhang, C. Wang, L. Chen, and Y. Yang, "When to pre-train graph neural networks? from data generation perspective!" in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 142–153.

[15] Z. Liu, X. Yu, Y. Fang, and X. Zhang, "Graphprompt: Unifying pre-training and downstream tasks for graph neural networks," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 417–428.

[16] M. Qin, C. Zhang, B. Bai, G. Zhang, and D.-Y. Yeung, "Towards a better tradeoff between quality and efficiency of community detection: An inductive embedding method across graphs," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 17, no. 9, pp. 1–34, 2023.

[17] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. S. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," in *Proceedings of the 8th International Conference on Learning Representations (ICLR)*. OpenReview.net, 2020.

[18] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the National Academy of Sciences (PNAS)*, vol. 105, no. 4, pp. 1118–1123, 2008.

[19] A. J. Uppal and H. H. Huang, "Fast stochastic block partition for streaming graphs," in *Proceedings of 2018 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–6.

[20] R. I. Arriaga and S. Vempala, "An algorithmic theory of learning: Robust concepts and random projection," *Machine learning*, vol. 63, pp. 161–182, 2006.

[21] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 6861–6871.

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the 5th International Conference on Learning Representations (ICLR)*. OpenReview.net, 2017.

[23] P.-W. Wang and J. Z. Kolter, "Community detection using fast low-cardinality semidefinite programming," *Advances in Neural Information Processing Systems (NIPS)*, vol. 33, pp. 3374–3385, 2020.

[24] T. P. Peixoto, "Efficient monte carlo and greedy heuristic for the inference of stochastic block models," *Physical Review E*, vol. 89, no. 1, p. 012804, 2014.

[25] C. Peng, Z. Zhang, K.-C. Wong, X. Zhang, and D. Keyes, "A scalable community detection algorithm for large graphs using stochastic block models," in *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[26] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory & Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[27] K. Lei, M. Qin, B. Bai, and G. Zhang, "Adaptive multiple non-negative matrix factorization for temporal link prediction in dynamic networks," in *Proceedings of ACM SIGCOMM 2018 Workshop on Network Meets AI & ML*, 2018, pp. 28–34.

[28] K. Lei, M. Qin, B. Bai, G. Zhang, and M. Yang, "Gcn-gan: A non-linear temporal link prediction model for weighted dynamic networks," in *Proceedings of 2019 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2019, pp. 388–396.

[29] M. Qin, C. Zhang, B. Bai, G. Zhang, and D.-Y. Yeung, "High-quality temporal link prediction for weighted dynamic graphs via inductive embedding aggregation," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 35, no. 9, pp. 9378–9393, 2023.

[30] M. Qin and D.-Y. Yeung, "Temporal link prediction: A unified framework, taxonomy, and review," *ACM Computing Surveys (CSUR)*, vol. 56, no. 4, pp. 1–40, 2023.

[31] M. Qin, D. Jin, K. Lei, B. Gabrys, and K. Musial-Gabrys, "Adaptive community detection incorporating topology and content in social networks," *Knowledge-Based Systems*, vol. 161, pp. 342–356, 2018.

[32] P. Chunaev, T. Gradov, and K. Bochenina, "Community detection in node-attributed social networks: How structure-attributes correlation affects clustering quality," *Procedia Computer Science*, vol. 178, pp. 355–364, 2020.

[33] M. Qin and K. Lei, "Dual-channel hybrid community detection in attributed networks," *Information Sciences*, vol. 551, pp. 146–167, 2021.

[34] Z. Zhao, Z. Ke, Z. Gou, H. Guo, K. Jiang, and R. Zhang, "The trade-off between topology and content in community detection: An adaptive encoder–decoder-based nmf approach," *Expert Systems with Applications*, vol. 209, p. 118230, 2022.