# AdaSTORM: Scaling LLM Reasoning on Dynamic Graphs via Adaptive Spatio-Temporal Multi-Agent Collaboration

**Anonymous ACL submission**


## Abstract

Large Language Models (LLMs) demonstrate remarkable potential in dynamic graph reasoning, but suffer from a scaling bottleneck: current models can only handle graphs with tens of nodes, constrained by exponential reasoning overhead and finite context windows. While multi-agent systems (MAS) offer collective reasoning and topology-aware orchestration, capabilities naturally suited for graph-structured tasks, their application to dynamic graphs remains unexplored. This paper presents Scaling LLM Reasoning on Dynamic Graphs via Adaptive Spatio-Temporal Multi-Agent Collaboration (AdaSTORM), a framework that reformulates large-scale dynamic graph reasoning into two stages: (i) **Adaptive Partitioning**, partitioning large-scale dynamic graphs into subregions that match the model's reasoning capacity while minimizing inference cost; and (ii) **Collaborative Reasoning**, aligning graph partition topologies with a spatio-temporal decoupled multi-agent architecture. AdaSTORM is the first multi-agent framework tailored for dynamic graph reasoning. Extensive experiments show that AdaSTORM successfully breaks through the scaling bottleneck, scaling reasoning to thousand-node graphs with over 90% accuracy across several large-scale dynamic graph settings without external tools, significantly outperforms seven competitive baselines. Furthermore, it achieves state-of-the-art accuracy on existing benchmarks and generalizes robustly to real-world datasets. The source code is available at: `https://anonymous.4open.science/r/AdaSTORM-BB6D/`.


## 1 Introduction

Large Language Models (LLMs) show promise in reasoning over graph-structured data (Mao et al., 2024; Jin et al., 2024), with dynamic graph reasoning standing out as a critical yet challenging frontier (Zhang et al., 2024c), as it captures the temporal evolution of relational data in applications ranging from social networks to traffic forecasting (Zhang et al., 2024b; Qian et al., 2024c; Chang et al., 2024).

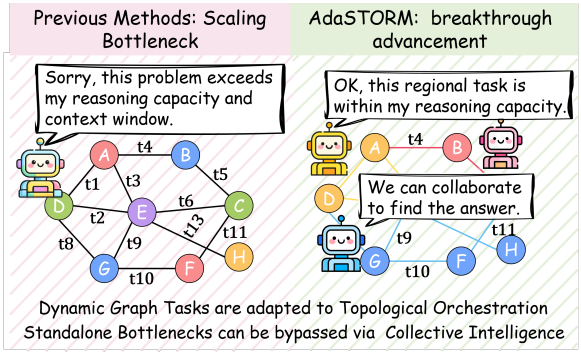



Figure 1: The core motivation of AdaSTORM. Left: Standalone LLMs suffer from severe reasoning capacity and contextual window constraints on large-scale dynamic graphs. Right: AdaSTORM bypasses these bottlenecks through collective reasoning.

Recent literature has explored two primary trajectories: benchmark evaluation (Hao et al., 2026; Dai et al., 2025b; Xu et al., 2026; Tang et al., 2025) and methodological development (Chai et al., 2026; Chen et al., 2024a; Luo et al., 2025). Despite these advancements, current methods remain confined to small-to-medium scale graphs (typically tens of nodes), as illustrated in Fig. 2. This limitation stems from a fundamental scaling bottleneck: as graph size increases, structural complexity rapidly exceeds the reasoning capacity and context window of a standalone model, inducing severe hallucinations and catastrophic performance collapse (Tang et al., 2025).

Multi-Agent Systems (MAS) (Chen et al., 2025; Guo et al., 2024a) overcome the reasoning capacity and contextual window limits of standalone LLMs by decomposing complex problems into collaborative sub-tasks (Wu et al., 2026; Hong et al., 2024; Wu et al., 2023). By orchestrating agents through structured topologies (Qian et al., 2024b;

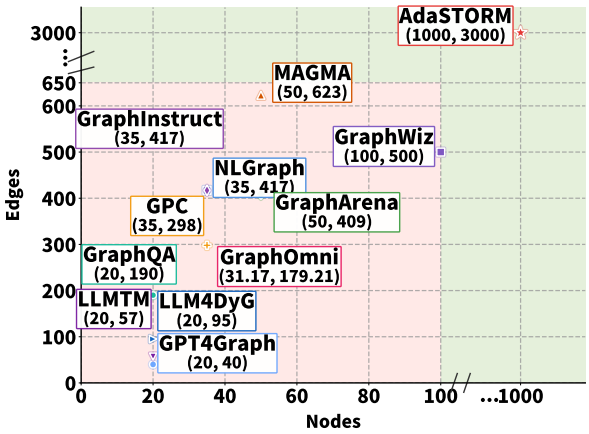



Figure 2: Comparison of graph scales handled by existing benchmarks and methods versus AdaSTORM. ($\#nodes$, $\#edges$) reflects the maximum node/edge counts reported in the respective papers.

Du et al., 2023), the communication architecture of MAS inherently mirrors the structural characteristics of graphs, rendering MAS naturally compatible with dynamic graph reasoning (Fig. 1). However, standard multi-agent paradigms (e.g., chain- and debate- based) fail to handle the combinatorial explosion triggered by graph scale (Lu and Wang, 2025; Fourney et al., 2024; Xu et al., 2025), as they typically operate globally on the entire graph topology while lacking mechanisms for large-scale graph partitioning. To bridge this gap, we present **AdaSTORM**, a multi-agent framework that reformulates large-scale dynamic graph reasoning by exploiting the inherent partitionability (Patwary et al., 2021; Li et al., 2019) and spatio-temporal decoupling (Zhang et al., 2024c) of dynamic graphs.

Specifically, AdaSTORM operates in two sequential stages: (i) **Adaptive Partitioning** tailors graph decomposition to model capacity. A capacity estimator assesses model feasibility by fusing learnable latent vectors with multimodal features, including LM-encoded queries, model profile, and GNN-based structural encodings. Guided by this capacity estimator and a cost simulator, a reinforcement learning (RL)-based adaptive partitioner learns a policy network to recursively split subregions and migrate nodes, deriving an optimal graph partition that ensures each subregion match the model's reasoning capacity while minimizing inference cost; (ii) **Collaborative Reasoning** aligns graph partition topologies with a spatio-temporal decoupled multi-agent architecture. By deploying specialized spatial and temporal agents for parallel localized inference, and leveraging cooperative message passing across boundary cut-edges for collective reasoning, the framework substantially boosts both the accuracy and scale of dynamic graph tasks for LLMs without external tools.

Our contributions are summarized as follows:

**Framework**. AdaSTORM is the first multi-agent framework tailored for dynamic graph reasoning. By exploiting the inherent partitionability, spatio-temporal decoupling and topological affinity of dynamic graphs with multi-agent orchestration, our framework effectively bypasses the scaling bottleneck of standalone LLMs, substantially boosting both the accuracy and scale of dynamic graph tasks for LLMs without external tools.

**Partitioning**. We design a reasoning capacity-guided reinforcement learning scheme that adaptively partitions large-scale graphs. Driven by flexible reward functions and custom adaptation criteria, this objective-agnostic framework seamlessly generalizes to diverse partitioning goals.

**Experiments**. Extensive evaluation validates that AdaSTORM effectively breaks througt the scaling bottleneck, consistently outperforms single-agent and multi-agent baselines across graph scales. Specifically, it achieves state-of-the-art (SOTA) performance against existing small-to-medium scale baselines (Fig. 5). Notably, it maintains over 90% accuracy on several large-scale dynamic graph reasoning settings (Table 1) and generalizes effectively to real-world datasets.

## 2 Preliminaries and Problem Formulation

**Dynamic Graph**. A dynamic graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the node set and $\mathcal{E}$ is a time-ordered sequence of events. To explicitly capture richer structural dynamics, we represent each event as a lifecycle quadruplet $(u, v, s, t) \in \mathcal{E}$, where $s$ and $t$ denote the emergence and deletion timestamps, respectively. The static projection of $\mathcal{G}$ is denoted as $G = (\mathcal{V}, E_s)$ with $E_s = \{\{u, v\} \mid \exists (u, v, s, t) \in \mathcal{E}\}$. More Preliminaries are deferred to Appendix A.

**Reasoning Feasibility**. Given an LLM $\mathcal{M}$ and a reasoning task $\mathcal{T}$, we define the reasoning feasibility of $\mathcal{M}$ on a graph region $P_i$ as the probability of successfully outputting the ground-truth answer $y^\star$:

$$\phi_i \equiv \phi_{\mathcal{M}}(P_i, \mathcal{T}) = \Pr[\mathcal{M}(P_i, \mathcal{T}) = y^\star]. \quad (1)$$

A subregion $P_i$ is reasoning-feasible if $\phi_i \geq \tau$, where $\tau$ is a predefined performance threshold.

This metric quantifies the model's inherent reasoning boundaries on structured data.

**Inference Cost**. Given a partitioning $\mathcal{P}$ and a task $\mathcal{T}$, the inference cost $C(\mathcal{P})$ is the total number of agent invocations required to solve the task.

**Problem**. Given a dynamic graph $\mathcal{G}$, an LLM $\mathcal{M}$, and a reasoning task $\mathcal{T}$, our goal is to perform inference via a multi-agent architecture, formulated as a two-stage sequential process:

(i) Adaptive Partitioning: Derive an optimal partitioning $\mathcal{P}$ that minimizes inference cost $C(\mathcal{P})$ while bounded by the model's reasoning capacity:

$$\min_{\mathcal{P}} C(\mathcal{P}) \quad \text{s.t.} \quad \phi_i \geq \tau, \quad \forall P_i \in \mathcal{P}. \tag{2}$$

(ii) Collaborative Reasoning: Given $\mathcal{P}$, resolve complex dynamic graph reasoning tasks via multi-agent collaboration, maximizing both accuracy and efficiency.

# 3 Methodology

Figure 3 illustrates the overall architecture of AdaSTORM, which consists of two sequential stages: (i) Adaptive Partitioning tailors graph decomposition to model capacities, where a capacity constrainer and a cost simulator co-guide an RL-based adaptive partitioner, thereby partitioning large-scale graphs into localized subregions, ensuring each subregion matches the model's reasoning capacity while minimize inference cost; (ii) Collaborative Reasoning aligns partition topologies with multi-agent orchestration, deploying specialized spatial and temporal agents for parallel localized inference and coordinating across cut-edges to achieve accurate and cost-effective responses without external tools.

## 3.1 Adaptive Partitioning Stage

### 3.1.1 Capacity Estimator

To guide adaptive partitioner, the Capacity Estimator predicts the feasibility metric $\phi_i$, which quantifies the probability that a target model successfully resolves a given query over subregion $P_i$.

This module bridges semantic reasoning with graph topology by constructing a multimodal joint representation $\mathbf{z}_i$. Specifically, it fuses four orthogonal modalities: LM-encoded task queries; LM-encoded model profiles synthesized via an auxiliary LLM to characterize high-level semantic attributes, including model capabilities and usage scenarios (see Appendix B); GNN-based topological encodings that uncover intrinsic structural properties; and a learnable latent vector designed to capture fine-grained behavioral nuances and adaptive task signals that transcend explicit descriptors.

Based on the joint embedding $\mathbf{z}_i$, a Multi-Layer Perceptron ($\text{MLP}_{\text{est}}$) estimates the regional feasibility. The network is optimized via a binary cross-entropy criterion:

$$\begin{aligned} \hat{\phi}_i &= \text{MLP}_{\text{est}}(\mathbf{z}_i), \\ \mathcal{L}_{\text{est}} &= -y_i \log(\hat{\phi}_i) - (1 - y_i) \log(1 - \hat{\phi}_i), \end{aligned} \tag{3}$$

where $y_i \in \{0, 1\}$ is the ground-truth verification label indicating whether the target model correctly resolves the task over subregion $P_i$.

### 3.1.2 Cost Estimator

To guide adaptive partitioning, we employ a Cost Estimator to evaluate the inference cost of a given partitioning strategy. Specifically, given a partitioning $\mathcal{P}$ and a task $\mathcal{T}$, the estimator programmatically traces the multi-agent execution process and returns the inference cost $C(\mathcal{P})$, which is defined as the total number of agent invocations required to resolve the task.

### 3.1.3 RL-based Adaptive Partitioner

Existing graph partitioning paradigms typically rely on a pre-defined number of partitions $k$ (Nazi et al., 2019; Petroni et al., 2015; Xie et al., 2014; Tsourakakis et al., 2014; Zhang et al., 2017). Yet, a model's reasoning capacity for a given task cannot be easily quantified via fixed node or edge counts. Moreover, as evidenced in Figure 4, the reasoning performance of the identical model varies drastically across diverse tasks, rendering a universal $k$ sub-optimal. To address this, we propose the Adaptive Partitioner, which dynamically decomposes the graph into $k$ optimal partitions, where for the given task, each partition perfectly matches the model's reasoning capacity while minimizing inference cost. Given an input dynamic graph $\mathcal{G}$, we first derive its static projection $G$ and initialize the partitioning of $G$ into $k_0$ regions via a warm-start graph partitioner, providing a structured starting point for subsequent refinement.

For the current partitioning $\mathcal{P}$, The pre-trained Capacity Estimator evaluates the feasibility $\phi_i$ of each region $P_i$ under the target model–query pair. To quantify the global violation of reasoning feasibility, we define the global infeasibility gap as:

$$\Phi(\mathcal{P}) = \sum_{i=1}^{k} \frac{|P_i|}{N} \max(0, \tau - \phi_i), \tag{4}$$

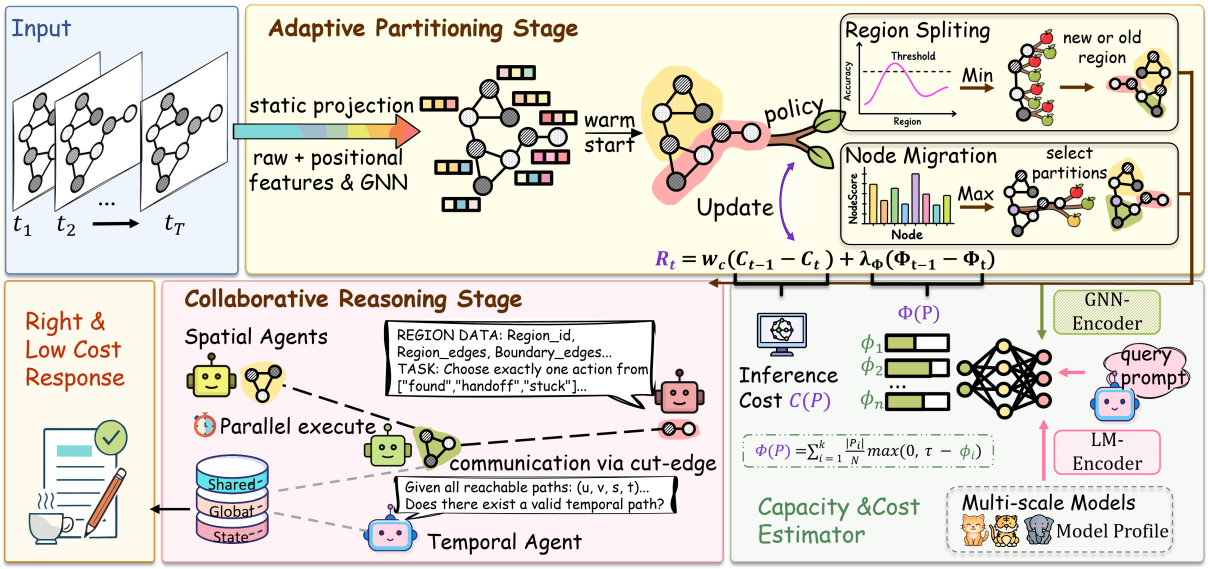
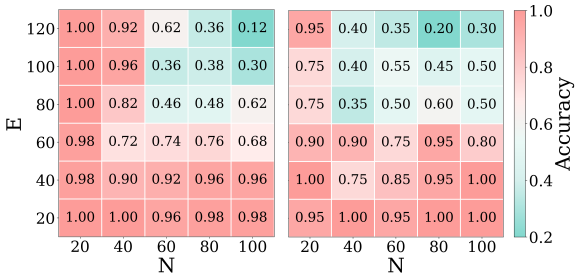



Figure 3: **Overall framework of AdaSTORM.** In Adaptive Partitioning Stage, we partition large-scale dynamic graphs into capacity-matched, cost-minimizing subregions. In Collaborative Reasoning Stage, we align partition topologies with spatio-temporally decoupled multi-agent orchestration for parallel localized inference, coordinating across cut-edges to achieve accurate and cost-effective responses without external tools.




(a) Connected Components (b) Community Detection

Figure 4: Reasoning performance of the identical model across distinct topological tasks under varying node ($N$) and edge ($E$) scales. Extended results on remaining tasks in C.

where $N$ is the total number of nodes, $|P_i|$ denotes the size of the $i$-th partition and $\tau$ is the feasibility threshold. Here, a higher $\tau$ imposes a stricter feasibility constraint, while a smaller $\Phi(\mathcal{P})$ indicates that more regions are within the model's reasoning capacity. Concurrently, the Cost Estimator computes the inference cost $C(\mathcal{P})$ by programmatically tracing the multi-agent execution process.

The Adaptive Partitioner refines $\mathcal{P}$ with two complementary actions. *Region splitting* subdivides infeasible regions to reduce the global infeasibility gap, where each node within the target region is either assigned to the new subregion or kept in the old one. *node migration* moves misplaced boundary nodes across partitions to reduce the inference cost. At each step, the policy selects one refinement action conditioned on the current feasibility and inference cost. The reward is defined by the improvement across these dual objectives:

$$r_t = w_c(C_{t-1} - C_t) + \lambda_\Phi(\Phi_{t-1} - \Phi_t) - \lambda_{\text{split}}\mathbb{1}_{\text{split}} - \lambda_{\text{reject}}\mathbb{1}_{\text{reject}}, \quad (5)$$

where $C_t$ is the inference cost, $\Phi_t$ is the global infeasibility gap (Eq. (4)), $\mathbb{1}_{\text{split}}$ and $\mathbb{1}_{\text{reject}}$ penalize unnecessary over-segmentation and ineffective refinements, respectively.

We train the policy with policy-gradient optimization. At inference time, the learned policy progressively refines the warm-start partitioning into feasibility-satisfying and inference-efficient subregions. The full MDP formulation and training details in Appendix E.

The adoption of reinforcement learning is motivated by three core insights. First, graph partitioning is NP-hard, making optimal supervision computationally intractable. Second, Our optimization objective is non-differentiable, as it depends on discrete reasoning feasibility and inference cost. Third, Partition refinement is inherently sequential: each split or migration affects later decisions. We therefore train the partitioner to maximize the cumulative reward over the refinement trajectory.

### 3.2 Collaborative Reasoning Stage

Existing literature indicates that prompt-level spatio-temporal decoupling effectively enhances

dynamic graphs reasoning in single LLMs (Zhang et al., 2024c). Therefore, after adaptive partitioning, we aligns the resulting partition topologies with a spatio-temporal decoupled multi-agent architecture for collaborative reasoning. Specifically, Spatial Agents are assigned to subregions to handle static local topology reasoning and cross-region communication, while Temporal Agents operate on dynamic events to enforce chronological constraints. LangGraph (Wang and Duan, 2024) maintains a shared global state, enabling independent agents to execute in parallel, exchange intermediate results, and update their outputs to achieve accurate and efficient responses without external tools.

**Spatial Agent**. A Spatial Agent is assigned to each subregion to handle static local topology reasoning. Each agent ingests a local subgraph defined by its nodes, edges, and boundary metadata (Appendix F). Collaboration is managed via the LangGraph-maintained shared global state: updates and messages are committed to this central repository, allowing a dispatcher to route subsequent subregion tasks based on target identifiers.

**Temporal Agent**. The Temporal Agent operates on dynamic events mapped from static edges (Appendix A) or the original dynamic graph to enforce chronological constraints, focusing on the occurrence, recurrence, and persistence of events (Appendix F). Driven by query requirements rather than a rigid pipeline, the Temporal Agent can acts as either a pre-filter for search-space pruning or a post-validator to refine spatial reasoning outcomes.

The LangGraph shared global state enables parallel execution of independent agents. By jointly updating outputs from both Spatial and Temporal Agents, this orchestration achieve accurate and efficient responses without external tools.

# 4 Experiments

We evaluate AdaSTORM through three research questions:

**RQ1:** How does AdaSTORM perform on large-scale dynamic graph tasks, particularly in comparison to leading closed-source models such as DeepSeek-V4-Flash?

**RQ2:** How does AdaSTORM perform on existing small-to-medium scale graph reasoning benchmarks?

**RQ3:** How well does AdaSTORM generalize to diverse real-world datasets?

In addition, we conduct comprehensive ablation studies to isolate and verify the contributions of each component, including the Capacity Estimator, the Adaptive Partitioner, the dynamic graph quadruple representation, spatio-temporal decoupling, and multi-agent coordination.

## 4.1 Experimental Setup

**Models and Baselines**. We backbone AdaSTORM with DeepSeek-R1-Distill-Qwen-7B, 14B, and 32B (DeepSeek-AI, 2025), denoted as Ours-DQwen-7B, Ours-DQwen-14B, and Ours-DQwen-32B, respectively. Baseline comprise: (i) standalone LLMs, including DeepSeek-V4-Flash (DeepSeek-AI, 2026), GPT-4o mini (OpenAI et al., 2024), and vanilla DeepSeek-R1-Distill-Qwen backbones; (ii) chain-based (Qian et al., 2024b) and debate-based (Du et al., 2023) multi-agent architectures; and (iii) graph-reasoning specialized frameworks, including GraphWiz (Chen et al., 2024a), GraphArena (Tang et al., 2025), GraphInstruct (Luo et al., 2025), NLGraph (Wang et al., 2024), GPC (Dai et al., 2025a), and LLM4DyG (Zhang et al., 2024c).

**Tasks and Metrics**. We evaluate AdaSTORM across four core dynamic graph tasks:

Propagation (Reachability): Evaluates whether a node $v$ is temporally reachable from $u$ via a time-respecting path $\mathcal{P}_{u\to v} = \{(u_i, u_{i+1}, s_i, t_i)\}_{i=1}^{k}$, where $u_1 = u$, $u_{k+1} = v$, and $t_i \leq s_{i+1}$.

Statistical (Temporal Motif Counting): Counts the total occurrences of a target temporal motif configuration $M_t$, defined as the cardinality $|\{\mathcal{G}' \subseteq \mathcal{G} \mid \mathcal{G}' \cong M_t\}|$.

Topological Analysis (Connected Components): Computes the number of connected components $|\mathcal{C}_c(\mathcal{G}_{\Delta T})|$ within a specified temporal window $\Delta T = [t_{\text{start}}, t_{\text{end}}]$.

Structural Mining (Community Detection): Discovers the latent community structures $|\mathcal{C}_d(\mathcal{G}_{\Delta T})|$ within the temporal window $\Delta T$.

We employ accuracy as the primary evaluation metric. For community detection, the output of the Label Propagation algorithm (Raghavan et al., 2007) serves as the ground truth.

**Datasets**. Synthetic dynamic graphs are generated by first initializing a static topology via the Stochastic Block Model (SBM) (Abbe, 2023), $G_0 = \text{SBM}(N, C, p_{\text{in}}, p_{\text{out}})$. Each edge is assigned a random timestamp uniformly distributed across the total time span, defined as $\alpha \times |E_0|$ (defaulting to $\alpha = 1.5$). To simulate edge dynamics, all initial edges are designated as add operations , with

Table 1: Performance comparison under various graph scales ($N$). We compare AdaSTORM with leading LLMs, chain- and debate-based multi-agent baselines. "Acc" denotes accuracy, "$k$" denotes the average number of regions generated by AdaSTORM's adaptive partitioning stage, and "-" indicates failure due to exceeding context window.

| Models | Community Detection | | | | | | Connected Components | | | | | | Reachability | | | | | | Temporal Motif Counting | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 500 | | 800 | | 1000 | | 500 | | 800 | | 1000 | | 500 | | 800 | | 1000 | | 500 | | 800 | | 1000 | |
| | Acc. | $k$ | Acc. | $k$ | Acc. | $k$ | Acc. | $k$ | Acc. | $k$ | Acc. | $k$ | Acc. | $k$ | Acc. | $k$ | Acc. | $k$ | Acc. | $k$ | Acc. | $k$ | Acc. | $k$ |
| DeepSeek-Distill-7B | 0% | 1.0 | – | – | – | – | 0% | 1.0 | – | – | – | – | 47% | 1.0 | – | – | – | – | 0% | 1.0 | – | – | – | – |
| DeepSeek-Distill-14B | 0% | 1.0 | – | – | – | – | 0% | 1.0 | – | – | – | – | 63% | 1.0 | – | – | – | – | 0% | 1.0 | – | – | – | – |
| DeepSeek-Distill-32B | 0% | 1.0 | – | – | – | – | 0% | 1.0 | – | – | – | – | 66% | 1.0 | – | – | – | – | 0% | 1.0 | – | – | – | – |
| GPT-4o mini | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 52% | 1.0 | 48% | 1.0 | 46% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 |
| DeepSeek-V4-Flash | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 58% | 1.0 | 65% | 1.0 | 53% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 |
| Chain-GPT-4o mini | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 66% | 1.0 | 68% | 1.0 | 63% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 |
| Debate-GPT-4o mini | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 | 62% | 1.0 | 56% | 1.0 | 53% | 1.0 | 0% | 1.0 | 0% | 1.0 | 0% | 1.0 |
| Ours-DeepSeek-Distill-7B | 23% | 20.9 | 16% | 32.4 | 10% | 46.1 | 58% | 48.0 | 56% | 79.4 | 42% | 106.6 | 57% | 14.6 | 51% | 20.5 | 52% | 27.5 | **100%** | 5.1 | 82% | 9.7 | 56% | 18.8 |
| Ours-DeepSeek-Distill-14B | 45% | 16.1 | 30% | 33.3 | 19% | 47.4 | 62% | 20.4 | 43% | 35.9 | 45% | 41.9 | 76% | 12.0 | 70% | 18.7 | 66% | 25.4 | **100%** | 5.0 | 86% | 9.1 | 60% | 17.3 |
| Ours-DeepSeek-Distill-32B | **71%** | 7.5 | **52%** | 95.4 | **26%** | 162.0 | **90%** | 18.0 | **77%** | 27.2 | **63%** | 34.9 | **90%** | 9.4 | **88%** | 15.1 | **72%** | 22.5 | **100%** | 5.0 | **91%** | 7.0 | **83%** | 12.1 |

delete operations subsequently introduced to a random subset, yielding the dynamic graph quadruple representation $(u, v, s, t)$. For reachability tasks, datasets are generated to ensure a balanced distribution of positive and negative samples. Full reproducibility is secured via a fixed random seed.

To evaluate generalization, we further employ five real-world dynamic graph datasets with richer structural and temporal diversity than synthetic benchmarks: *Wikipedia* (Kumar et al., 2019), an interaction network with 9,227 nodes (editors and Wikipedia pages) and 157,474 timestamped edit edges with 172-dimensional features; *Reddit* (Poursafaei et al., 2022), a social network consisting of 10,984 nodes (users and subreddits) and 672,447 timestamped posting edges with 172-dimensional features; *Enron* (Shetty and Adibi, 2004), a communication network with 184 nodes (employees) and 125,235 timestamped email edges; *Flights* (Schäfer et al., 2014), an airport network consisting of 13,169 nodes (airports) and 1,927,145 timestamped flight edges; and *UNTrade* (MacDonald et al., 2015), a trade network consisting of 255 nodes (countries) and 507,497 timestamped trade edges. We sample dense local subgraphs with random walk with restart (RWR) (Ham et al., 2013), selecting the top-$k$ most frequently visited nodes from random sources to preserve local topology.

**Parameter Settings**. All evaluations are executed via official DeepSeek APIs (Guo et al., 2025) under default configurations. The warm-start graph partitioner defaults to METIS (Karypis and Kumar, 1999) with $k_0 = 5$. Detailed parameter configurations are provided in Appendix D.

### 4.2 Performance on Large-Scale Dynamic Graphs (RQ1)

To evaluate AdaSTORM under substantial structural complexity, we generate 100 synthetic dynamic graphs for each scale ($N \in \{500, 800, 1000\}$). All evaluations are repeated five times, with the average metrics reported in Table 1. Notably, for Reachability task, AdaSTORM provides correct temporal paths to justify its answers, while baselines likely rely on guesswork. We summarize the core empirical findings below:

**Observation 1: Breaking through the Scaling Bottleneck**. Facing massive dynamic graph topologies, even leading closed-source models and multi-agent baselines suffer catastrophic failure, yielding near-zero accuracy except for Reachability tasks. This demonstrates that such complexity **far exceeds** the reasoning capabilities of a standalone agent. Furthermore, standard multi-agent paradigms (e.g., chain- or debate-based) fail to handle the combinatorial explosion triggered by graph scale (Lu and Wang, 2025; Fourney et al., 2024; Xu et al., 2025), as they typically operate globally on the entire graph topology while **lacking explicit mechanisms** for large-scale graph partitioning. In contrast, AdaSTORM effectively **breaks through** the scaling bottleneck, successfully scaling reasoning capabilities to thousand-node networks, while others can only handle graphs with fewer than a hundred nodes. Meanwhile, it significantly **outperforms** all five single-agent and two multi-agent baselines, notably achieving 100% accuracy on the Temporal Motif Counting task at $N = 500$, marking a fundamental advancement in LLM-based graph reasoning.

**Observation 2: Consistent Performance Gains across Model Scales**. AdaSTORM delivers robust performance improvements across varying backbone scales, consistently outperforming leading closed-source models, effectively unlocking the latent capabilities of smaller models in graph reasoning tasks. This advantage holds firmly across all evaluated tasks and graph sizes, validating the

Table 2: Dataset statistics and generalization evaluation on real-world networks.

| Dataset | #Nodes | #Edges | Community Detection | | Connected Components | | Reachability | | Temporal Motif Counting | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Acc. | $k$ | Acc. | $k$ | Acc. | $k$ | Acc. | $k$ |
| Reddit | 500 | 3,025 | 35% | 32.0 | 73% | 8.1 | 65% | 8.8 | 100% | 9.7 |
| Enron | 184 | 125,235 | 54% | 79.0 | 95% | 5.0 | 70% | 28.0 | 10% | 28.0 |
| Flight | 300 | 15,306 | 20% | 8.0 | 91% | 5.0 | 65% | 30.6 | 15% | 5.2 |
| UNtrade | 255 | 507,497 | 21% | 8.0 | 93% | 5.0 | 50% | 45.4 | 5% | 5.0 |
| Wiki | 500 | 2,136 | 15% | 53.7 | 15% | 11.4 | 76% | 14.0 | 100% | 14.1 |

Table 3: Ablation results of the capacity estimator under different graph scales. "w/o CE" denotes replacing the Capacity Estimator with a random feasibility score.

| Method | 500 | | | 800 | | | 1000 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Acc. | $k$ | Token | Acc. | $k$ | Token | Acc. | $k$ | Token |
| AdaSTORM w/o CE | 61% | 34.0 | 33,756 | 54% | 47.5 | 61,154 | 42% | 51.2 | 114,580 |
| **AdaSTORM** | **90%** | **18.0** | **16,723** | **77%** | **27.2** | **34,146** | **63%** | **34.9** | **50,754** |

structural robustness of AdaSTORM. Notably, the larger 32B backbone yields superior performance owing to its greater inherent capacity.

**Observation 3: Inherent Scalability via Scale-Agnostic Partitioning**. AdaSTORM demonstrates seamless scalability to larger graph topologies. Since the adaptive partitioner is scale-agnostic by design, the multi-agent orchestration naturally bypasses conventional context window constraints and scale bottlenecks, endowing the architecture with inherent scaling capabilities.

### 4.3 Performance on Existing Benchmarks (RQ2)

AdaSTORM is evaluated on recent graph reasoning benchmarks under their native task and graph configurations. Performance metrics are compared against state-of-the-art (SOTA) results reported in their original papers, as plotted in Fig. 5. The empirical findings show that AdaSTORM efficiently resolves a diverse array of small-to-medium scale graph tasks across all established benchmarks.

### 4.4 Generalization on Real-World Datasets (RQ3)

Evaluations are extended to real-world networks to test AdaSTORM against richer structural diversity than synthetic datasets. Specifically, we employ Ours-DeepSeek-Distill-32B as the backbone model to evaluate our framework across all datasets. The experimental results and dataset statistics are summarized in Table. 2. The outcomes demonstrate that AdaSTORM generalizes robustly to real-world environments, underscoring its practical viability.

### 4.5 Ablation Study

**Capacity Estimator**. To verify its efficacy, we substitute the Capacity Estimator with a random feasibility score while keeping the rest of the pipeline unchanged. Based on the connected components task and employing Ours-DeepSeek-Distill-32B as the backbone model, The results in Table 3 demonstrate that the Capacity Estimator provides effective guidance for AdaSTORM.

**Adaptive Partitioner**. To isolate its specific contribution, we replace the adaptive partitioner with alternative partitioning algorithms: METIS (Karypis and Kumar, 1999), GAP (Nazi et al., 2019), KaHIP (Akhremtsev et al., 2014), and Random partitioning, while the subregion count to the value determined by our method. This yields four evaluation variants: METIS-STORM, GAP-STORM, KaHIP-STORM, and Random-STORM. Their performance metrics and token overhead are compared in Fig. 6. The results demonstrate that our adaptive partitioner not only automates region count determination but also yields superior topological quality, delivering higher end-to-end performance and lower token overhead under an identical partition count.

**Dynamic Graph Quadruple Representation** $(u, v, s, t)$. The dynamic graph is formalized within the prompt using the quadruple $(u, v, s, t)$. Compared to existing $(u, v, t, a/d)$ (Hao et al., 2026), it maximizes information density with minimal token overhead, facilitating LLM comprehension. To isolate the specific contribution of this design, we evaluate two standalone agent variants on the Reachability task: SAgent-v1 employing $(u, v, t, a/d)$ and SAgent-v2 utilizing $(u, v, s, t)$. The experimental results, illustrated in Fig. 7, validate the necessity of this representation optimization.

**Spatio-Temporal Decoupling**. Prior studies (Zhang et al., 2024c) have validated the efficacy of prompt-level spatio-temporal decoupling in single LLMs. To evaluate this mechanism under a multi-agent setting, we introduce a variant STAgent, which orchestrates decoupled spatial and temporal agents, and compare it against a monolithic single-agent baseline. As shown in Fig. 7, STAgent consistently outperforms the single-agent baseline, fully validating the architectural superiority of explicit spatio-temporal decoupling.

**Multi-Agent Orchestration**. Incorporating the adaptive partitioning mechanism into the spatial agent of STAgent yields our complete framework.

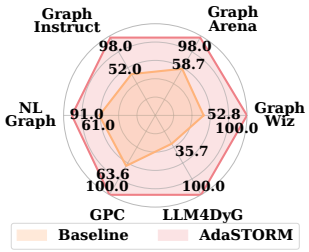



Figure 5: Performance comparison on existing graph reasoning benchmarks (Ours-DeepSeek-Distill-32B).

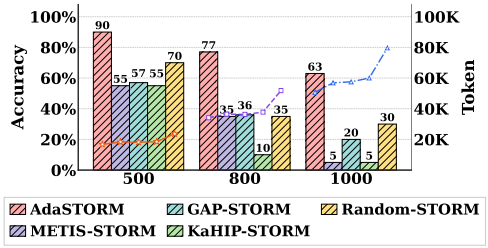



Figure 6: Ablation results (performance and token cost) of the adaptive partitioner across graph scales. Evaluated on the connected components task under same partition counts.

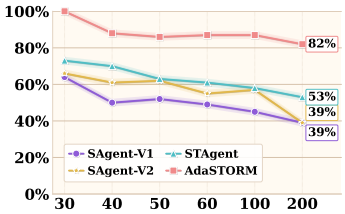



Figure 7: Ablation results of quadruple representation, spatio-temporal decoupling, and multi-agent orchestration on the reachability task.

As shown in Fig. 7, comparing AdaSTORM against the STAgent variant on the reachability task, validates the efficacy of multi-agent orchestration.

## 5 Related Work

**LLMs for Graphs**. LLM-based graph analytics generally bifurcates into benchmarking and methodological development. Benchmarks span static environments (e.g., GraphInstruct (Luo et al., 2025), GraphArena (Tang et al., 2025), GraphOmni (Xu et al., 2026)) and temporal dynamics (e.g., LLM4DyG (Zhang et al., 2024c), LLMTM (Hao et al., 2026)). Methodologically, existing efforts fall into two paradigms: (i) GNN-enhanced LLMs (Tang et al., 2024; Zhang et al., 2024a), injecting structural embeddings into token spaces yet limiting cross-task generalization; and (ii) Text-flattened LLMs (Ouyang et al., 2024; Chai et al., 2026; Fatemi et al., 2023; Perozzi et al., 2024), serializing graphs into text descriptions for direct reasoning. While monolithic approaches scale poorly due to exponential reasoning overhead and finite context windows, AdaSTORM bypasses these architectural limits via adaptive multi-agent orchestration, scaling dynamic graph reasoning to unprecedented network dimensions.

**Multi-Agent Systems**. Multi-agent systems (MAS) (Hong et al., 2024; Wu et al., 2026) orchestrate specialized LLM-based agents to achieve autonomous problem-solving across diverse domains, spaning from software development to embodied control (Qian et al., 2024a; Guo et al., 2024b; Chen et al., 2024b; Mandi et al., 2024). Technically, these frameworks orchestrate specialized agents through structured topologies, such as chains, trees, or graphs (Qian et al., 2025). This topological alignment renders MAS inherently compatible with dynamic graph reasoning.

**Graph Partitioning**. Conventional graph partitioning methodologies treat the problem as NP-hard discrete optimization, typically resolved via heuristics or approximations (Karypis et al., 1999; Ayall et al., 2022; Stanton and Kliot, 2012). However, these approaches rely on handcrafted objectives, suboptimally utilizing node attributes. While neural partitioning methods incorporate rich features and clustering constraints via continuous graph cuts (Nazi et al., 2019; Duong et al., 2023), modularity maximization (Wang et al., 2017; Tsitsulin et al., 2023; Bhowmick et al., 2023), or combinatorial reformulations (Jung and Keuper, 2022; Shah et al., 2024), their reliance on static partition counts fails to accommodate dynamic multi-agent orchestration. To break this bottleneck, we introduce an adaptive partitioner that dynamically determining the optimal region count on-the-fly while preserving superior structural partition quality.

## 6 Conclusion

This work introduces AdaSTORM, a pioneering multi-agent framework tailored for large-scale dynamic graph reasoning. By coupling adaptive partitioning with collective reasoning, the architecture orchestrates multi-agent collaboration for dynamic graph reasoning. Empirical evaluations demonstrate that AdaSTORM successfully breaks through the scaling bottlenecks, scaling reasoning to thousand-node networks and significantly outperforms seven competitive baselines. Furthermore, it achieves state-of-the-art accuracy on existing benchmarks and generalizes robustly to real-world datasets, underscoring its practical viability.

As a modular framework, AdaSTORM inherently enhances the scalability of alternative graph-reasoning methods while adapting to diverse partitioning objectives. Future work will explore instruction tuning within this orchestration to unlock further performance gains.

## 7 Limitations

Although AdaSTORM demonstrates strong scalability and effectiveness on large-scale dynamic graph reasoning, its current formulation is most naturally suited to tasks where global graph reasoning can be decomposed into capacity-constrained subregions and recomposed through inter-region collaboration. This setting aligns well with the topological and temporal reasoning tasks considered in this work, but may be less directly applicable to scenarios whose solutions depend heavily on rich semantic attributes, external domain knowledge, or highly ambiguous natural-language objectives beyond the structural and temporal information encoded in the graph. In addition, AdaSTORM treats the backbone LLM as a black-box reasoner and improves its reasoning capability through adaptive partitioning and multi-agent orchestration rather than model-level adaptation. Therefore, the overall performance can still be affected by local reasoning errors, unstable intermediate outputs, or imperfect capacity estimation.

## 8 Impact Statement

This work aims to improve the ability of LLMs to reason over large-scale dynamic graphs, a setting that appears in many real-world domains such as social interaction analysis, communication networks, transportation systems, and temporal knowledge discovery. By decomposing complex graph structures into capacity-matched subregions and coordinating multiple agents for spatio-temporal reasoning, AdaSTORM provides a scalable framework for applying LLM-based reasoning to graph scenarios that are difficult to handle with standalone models. This may benefit research on graph analytics, multi-agent systems, and LLM-based decision support by enabling more accessible reasoning over structurally complex and temporally evolving data. At the same time, dynamic graphs may contain sensitive relational or behavioral information, and automated reasoning over such data should be conducted with appropriate privacy protection, data governance, and human oversight. The proposed framework is intended as a general reasoning methodology, and its deployment in high-stakes domains should be accompanied by careful validation and responsible use.

## References


Emmanuel Abbe. 2023. Community detection and stochastic block models. *Preprint*, arXiv:1703.10146.

Yaroslav Akhremtsev, Peter Sanders, Sebastian Schlag, and Christian Schulz. 2014. Kahip - karlsruhe high quality partitioning.

Tewodros Alemu Ayall, Huawen Liu, Changjun Zhou, Abegaz Mohammed Seid, Fantahun Bogale Gereme, Hayla Nahom Abishu, and Yasin Habtamu Yacob. 2022. Graph computing systems and partitioning techniques: A survey. *IEEE Access*, 10:118523–118550.

Aritra Bhowmick, Mert Kosan, Zexi Huang, Ambuj Singh, and Sourav Medya. 2023. Dgcluster: A neural framework for attributed graph clustering via modularity maximization. *Preprint*, arXiv:2312.12697.

Monica Bianchini, Marco Gori, and Franco Scarselli. 2005. Inside pagerank. *ACM Trans. Internet Technol.*, 5(1):92–128.

J. Bourgain. 1985. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1):46–52.

Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiao Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. 2026. Graphllm: Boosting graph reasoning ability of large language model. *IEEE Transactions on Big Data*, 12(2):475–483.

Ya-Ting Chang, Zhibo Hu, Xiaoyu Li, Shuiqiao Yang, Jiaojiao Jiang, and Nan Sun. 2024. Dihan: A novel dynamic hierarchical graph attention network for fake news detection. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, page 197–206, New York, NY, USA. Association for Computing Machinery.

Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. 2024a. Graphwiz: An instruction-following language model for graph problems. *Preprint*, arXiv:2402.16029.

Shuaihang Chen, Yuanxing Liu, Wei Han, Weinan Zhang, and Ting Liu. 2025. A survey on llm-based multi-agent system: Recent advances and new frontiers in application. *Preprint*, arXiv:2412.17481.

Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. 2024b. Scalable multi-robot collaboration with large language models: Centralized or decentralized systems? *Preprint*, arXiv:2309.15943.

Xinnan Dai, Haohao QU, Yifei Shen, Bohang Zhang, Qihao Wen, Wenqi Fan, Dongsheng Li, Jiliang Tang, and Caihua Shan. 2025a. How do large language models understand graph patterns? a benchmark for graph pattern comprehension. In *International Conference on Learning Representations*, volume 2025, pages 58405–58435.

Xinnan Dai, Haohao Qu, Yifen Shen, Bohang Zhang, Qihao Wen, Wenqi Fan, Dongsheng Li, Jiliang Tang, and Caihua Shan. 2025b. How do large language models understand graph patterns? a benchmark for graph pattern comprehension. *Preprint*, arXiv:2410.05298.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

DeepSeek-AI. 2026. Deepseek-v4: Towards highly efficient million-token context intelligence.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *Preprint*, arXiv:2305.14325.

Chi Thang Duong, Thanh Tam Nguyen, Trung-Dung Hoang, Hongzhi Yin, Matthias Weidlich, and Quoc Viet Hung Nguyen. 2023. Deep mincut: Learning node embeddings by detecting communities. *Pattern Recogn.*, 134(C).

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2023. Talk like a graph: Encoding graphs for large language models. *Preprint*, arXiv:2310.04560.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang, Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. 2024. Magentic-one: A generalist multi-agent system for solving complex tasks. *Preprint*, arXiv:2411.04468.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024a. Large language model based multi-agents: A survey of progress and challenges. *Preprint*, arXiv:2402.01680.

Xudong Guo, Kaixuan Huang, Jiale Liu, Wenhui Fan, Natalia Vélez, Qingyun Wu, Huazheng Wang, Thomas L. Griffiths, and Mengdi Wang. 2024b. Embodied llm agents learn to cooperate in organized teams. *Preprint*, arXiv:2403.12482.

Bumsub Ham, Dongbo Min, and Kwanghoon Sohn. 2013. A generalized random walk with restart and its application in depth up-sampling and interactive segmentation. *IEEE Transactions on Image Processing*, 22(7):2574–2588.

William L. Hamilton, Rex Ying, and Jure Leskovec. 2018. Inductive representation learning on large graphs. *Preprint*, arXiv:1706.02216.

Bing Hao, Minglai Shao, Zengyi Wo, Yunlong Chu, Yuhang Liu, and Ruijie Wang. 2026. Llmtm: Benchmarking and optimizing llms for temporal motif analysis in dynamic graphs. *Preprint*, arXiv:2512.22266.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. *Preprint*, arXiv:2308.00352.

Bowen Jin, Gang Liu, Chi Han, Meng Jiang, Heng Ji, and Jiawei Han. 2024. Large language models on graphs: A comprehensive survey. *Preprint*, arXiv:2312.02783.

Steffen Jung and Margret Keuper. 2022. Learning to solve minimum cost multicuts efficiently using edge-weighted graph convolutional neural networks. *Preprint*, arXiv:2204.01366.

G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. 1999. Multilevel hypergraph partitioning: applications in vlsi domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(1):69–79.

G. Karypis and V. Kumar. 1999. Multilevel k-way hypergraph partitioning. In *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*, pages 343–348.

Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, KDD '19, page 1269–1278. ACM.

He Li, Hang Yuan, and Jianbin Huang. 2019. Real-time edge repartitioning for dynamic graph. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 2125–2128, New York, NY, USA. Association for Computing Machinery.

Yuxing Lu and Jinzhuo Wang. 2025. Karma: Leveraging multi-agent llms for automated knowledge graph enrichment. *ArXiv*, abs/2502.06472.

Zihan Luo, Xiran Song, Hong Huang, Jianxun Lian, Chenhao Zhang, Jinqi Jiang, Xing Xie, and Hai Jin. 2025. Graphinstruct: Empowering large language models with graph understanding and reasoning capability. *Preprint*, arXiv:2403.04483.

Graham K. MacDonald, Kate A. Brauman, Shipeng Sun, Kimberly M. Carlson, Emily S. Cassidy, James S. Gerber, and Paul C. West. 2015. Rethinking agricultural trade relationships in an era of globalization. *BioScience*, 65:275–289.

Zhao Mandi, Shreeya Jain, and Shuran Song. 2024. Roco: Dialectic multi-robot collaboration with large

language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 286–299.

Qiheng Mao, Zemin Liu, Chenghao Liu, Zhuo Li, and Jianling Sun. 2024. Advancing graph representation learning with large language models: A comprehensive survey of techniques. *Preprint*, arXiv:2402.05952.

Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, and Azalia Mirhoseini. 2019. Gap: Generalizable approximate graph partitioning framework. *Preprint*, arXiv:1903.00614.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, and 401 others. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

Sheng Ouyang, Yulan Hu, Ge Chen, and Yong Liu. 2024. Gundam: Aligning large language models with graph understanding. *Preprint*, arXiv:2409.20053.

Md Anwarul Kaium Patwary, Saurabh Garg, Sudheer Kumar Battula, and Byeong Kang. 2021. Sdp: Scalable real-time dynamic graph partitioner. *Preprint*, arXiv:2110.15669.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. 2024. Let your graph do the talking: Encoding structured data for llms. *Preprint*, arXiv:2402.05862.

Fabio Petroni, Leonardo Querzoni, Khuzaima Daudjee, Shahin Kamali, and Giorgio Iacoboni. 2015. Hdrf: Stream-based partitioning for power-law graphs. pages 243–252.

Farimah Poursafaei, Shenyang Huang, Kellin Pelrine, and Reihaneh Rabbany. 2022. Towards better evaluation for dynamic link prediction. *Preprint*, arXiv:2207.10128.

Chen Qian, Yufan Dang, Jiahao Li, Wei Liu, Zihao Xie, Yifei Wang, Weize Chen, Cheng Yang, Xin Cong, Xiaoyin Che, Zhiyuan Liu, and Maosong Sun. 2024a. Experiential co-learning of software-developing agents. *Preprint*, arXiv:2312.17025.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024b. Chatdev: Communicative agents for software development. *Preprint*, arXiv:2307.07924.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2025. Scaling large language model-based multi-agent collaboration. *Preprint*, arXiv:2406.07155.

Hao Qian, Hongting Zhou, Qian Zhao, Hao Chen, Hongxiang Yao, Jingwei Wang, Ziqi Liu, Fei Yu, Zhiqiang Zhang, and Jun Zhou. 2024c. Mdgnn: multi-relational dynamic graph neural network for comprehensive and dynamic stock investment prediction. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press.

Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76:036106.

Matthias Schäfer, Martin Strohmeier, Vincent Lenders, Ivan Martinovic, and Matthias Wilhelm. 2014. Bringing up opensky: A large-scale ads-b sensor network for research. In *IPSN-14 Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, pages 83–94.

Rishi Shah, Krishnanshu Jain, Sahil Manchanda, Sourav Medya, and Sayan Ranu. 2024. Neurocut: A neural approach for robust graph partitioning. *Preprint*, arXiv:2310.11787.

Jitesh Shetty and Jafar Adibi. 2004. The enron email dataset database schema and brief statistical report.

Isabelle Stanton and Gabriel Kliot. 2012. Streaming graph partitioning for large distributed graphs. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, page 1222–1230, New York, NY, USA. Association for Computing Machinery.

R.S. Sutton and A.G. Barto. 1998. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. 2024. Graphgpt: Graph instruction tuning for large language models. *Preprint*, arXiv:2310.13023.

Jianheng Tang, Qifan Zhang, Yuhan Li, Nuo Chen, and Jia Li. 2025. Grapharena: Evaluating and exploring large language models on graph computation. *Preprint*, arXiv:2407.00379.

Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. 2023. Graph clustering with graph neural networks. *Preprint*, arXiv:2006.16904.

Charalampos E. Tsourakakis, Christos Gkantsidis, Bozidar Radunovic, and Milan Vojnović. 2014. Fennel: streaming graph partitioning for massive scale graphs. *Proceedings of the 7th ACM international conference on Web search and data mining*.

Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. 2017. Mgae: marginalized graph

autoencoder for graph clustering. In *CIKM'17 - Proceedings of the 2017 ACM Conference on Information and Knowledge Management*, pages 889–898, United States of America. Association for Computing Machinery (ACM). ACM International Conference on Information and Knowledge Management 2017, CIKM 2017 ; Conference date: 06-11-2017 Through 10-11-2017.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2024. Can language models solve graph problems in natural language? *Preprint*, arXiv:2305.10037.

Jialin Wang and Zhihua Duan. 2024. Agent ai with langgraph: A modular framework for enhancing machine translation using large language models. *Preprint*, arXiv:2412.03801.

Feijie Wu, Zitao Li, Fei Wei, Yaliang Li, Bolin Ding, and Jing Gao. 2026. Talk to right specialists: Iterative routing in multi-agent systems for question answering. *Preprint*, arXiv:2501.07813.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *Preprint*, arXiv:2308.08155.

Cong Xie, Ling Yan, Wu-Jun Li, and Zhihua Zhang. 2014. Distributed power-law graph computing: theoretical and empirical analysis. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, page 1673–1681, Cambridge, MA, USA. MIT Press.

Hao Xu, Xiangru Jian, Xinjian Zhao, Wei Pang, Chao Zhang, Suyuchen Wang, Qixin Zhang, Zhengyuan Dong, Joao Monteiro, Bang Liu, Qiuzhuang Sun, and Tianshu Yu. 2026. Graphomni: A comprehensive and extensible benchmark framework for large language models on graph-theoretic tasks. *Preprint*, arXiv:2504.12764.

Zelai Xu, Chao Yu, Fei Fang, Yu Wang, and Yi Wu. 2025. Language agents with reinforcement learning for strategic play in the werewolf game. *Preprint*, arXiv:2310.18940.

Chenzi Zhang, Fan Wei, Qin Liu, Zhihao Gavin Tang, and Zhenguo Li. 2017. Graph edge partitioning via neighborhood heuristic. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 605–614, New York, NY, USA. Association for Computing Machinery.

Mengmei Zhang, Mingwei Sun, Peng Wang, Shen Fan, Yanhu Mo, Xiaoxiao Xu, Hong Liu, Cheng Yang, and Chuan Shi. 2024a. Graphtranslator: Aligning graph model to large language model for open-ended tasks. *Proceedings of the ACM Web Conference 2024*.

Tianli Zhang, Tongya Zheng, Zhenbang Xiao, Zulong Chen, Liangyue Li, Zunlei Feng, Dongxiang Zhang, and Mingli Song. 2024b. Language models-enhanced semantic topology representation learning for temporal knowledge graph extrapolation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, page 3227–3236, New York, NY, USA. Association for Computing Machinery.

Zeyang Zhang, Xin Wang, Ziwei Zhang, Haoyang Li, Yijian Qin, and Wenwu Zhu. 2024c. Llm4dyg: Can large language models solve spatial-temporal problems on dynamic graphs? *Preprint*, arXiv:2310.17110.

## A Additional Preliminaries

**Graph Partitioning**. Given a graph $G = (\mathcal{V}, E_s)$, a graph partitioning $\mathcal{P}$ is defined as a collection of $k$ mutually disjoint and localized subgraphs:

$$\mathcal{P} = \{P_1, P_2, \ldots, P_k\}. \tag{6}$$

Each partition is formalized as a subgraph $P_i = (V_i, E_i)$. The vertex sets $\{V_1, V_2, \ldots, V_k\}$ form a valid partition over the global node set $\mathcal{V}$, satisfying:

$$V_i \cap V_j = \emptyset, \quad \forall i \neq j, \qquad \bigcup_{i=1}^{k} V_i = \mathcal{V}. \tag{7}$$

The corresponding edge set $E_i$ is induced directly from the global edge set $E_s$, retaining all links strictly confined within $V_i$:

$$E_i = \big\{\{u, v\} \in E_s \;\big|\; u \in V_i,\ v \in V_i\big\}. \tag{8}$$

Unlike existing partitioning methods, the number of subregions $k$ in AdaSTORM is not pre-fixed, but adaptively determined according to the reasoning task and the target model's capacity.

**Cut Edges**. Given a graph $G = (\mathcal{V}, E_s)$ and a partitioning $\mathcal{P} = \{P_1, \ldots, P_k\}$, the pairwise cut-edge set between two distinct partitions $P_i$ and $P_j$ is defined as:

$$E_{\text{cut}}(P_i, P_j) = \big\{\{u, v\} \in E_s \;\big|\; u \in V_i,\ v \in V_j\big\}, \tag{9}$$

where each edge in the undirected graph is represented as an unordered pair $\{u, v\}$. By aggregating these crossings, the global cut-edge set of the partitioning $\mathcal{P}$ is formulated as:

$$E_{\text{cut}}(\mathcal{P}) = \bigcup_{1 \leq i < j \leq k} E_{\text{cut}}(P_i, P_j). \tag{10}$$

**Dynamic Mapping**. When temporal information is required, the corresponding dynamic event set $\mathcal{E}_{\text{rec}}(E')$ is defined by recovering all lifecycle quadruplets from the global sequence $\mathcal{E}$:

$$\mathcal{E}_{\text{rec}}(E') = \big\{(u, v, s, t) \in \mathcal{E} \;\big|\; \{u, v\} \in E'\big\}. \tag{11}$$

## B Model Profiles

To construct the explicit semantic attributes required by the Cognitive Estimator, we employ an auxiliary LLM to synthesize structured model profiles. Specifically, the auxiliary model is prompted to extracts inherent architectural histories, capacities, and empirical graph-task performance directly from runtime execution logs, thereby characterizing high-level semantic attributes, including model capabilities and operational scenarios, as detailed below:

- **DeepSeek-R1-Distill-Qwen-7B**: This variant is an official DeepSeek-R1 distilled checkpoint built upon the Qwen2.5-Math-7B backbone. The R1 distilled family is fine-tuned on an 800k-sample dataset curated directly from DeepSeek-R1. Empirical evaluations indicate that it demonstrates robust proficiency in local reasoning, achieving approximately 79% accuracy on Reachability and 71% on Temporal Motif Counting, whereas it exhibits prominent bottlenecks on global topological tasks, dropping to 55% on Connected Components and 43% on Community Detection.

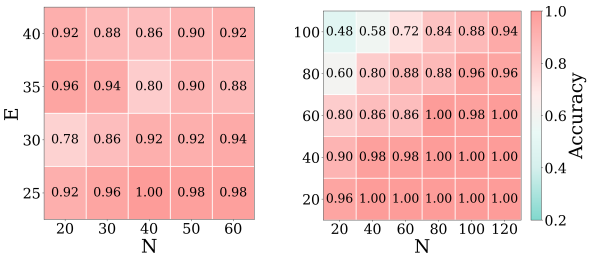
- **DeepSeek-R1-Distill-Qwen-14B**: This checkpoint is an official DeepSeek-R1 distilled model grounded on the Qwen2.5-14B architecture, leveraging the identical R1 distillation pipeline and the 800k-sample training configuration. According to the execution logs, it delivers a more balanced topological reasoning profile across the evaluation suite, elevating performance to approximately 82% on Reachability, 82% on Temporal Motif Counting, 76% on Connected Components, and 55% on Community Detection.

- **DeepSeek-R1-Distill-Qwen-32B**: This architecture represents an official DeepSeek-R1 distilled checkpoint based on the Qwen2.5-32B backbone. Official release notes highlight that this 32B distilled variant performs on par with or surpasses OpenAI o1-mini on selected reasoning benchmarks. Empirical logs demonstrate that it establishes the upper accuracy frontier among the three models, securing approximately 89% on Reachability, 90% on Temporal Motif Counting, 80% on Connected Components, and 71% on Community Detection.

## C Extended Task-Specific Performance Landscapes

Figure 8a and Figure 8b provide extended performance evaluations for graph reachability and temporal motif detection. Under the identical model

configuration, the resulting accuracy landscapes exhibit fundamentally distinct degradation trajectories as topological scales change. These evaluations directly corroborate our main claim that reasoning performance varies drastically across diverse tasks, rendering a universal, static partition count $k$ sub-optimal.




(a) Graph reachability. (b) Temporal motif.

Figure 8: Reasoning performance of the identical target model under varying node ($N$) and edge ($E$) scales for extended tasks.

# D Detailed Parameter Configurations

The GNN backbone is configured as a 2-layer GraphSAGE (Hamilton et al., 2018) with a hidden dimension of 32. Positional embeddings employ a default of 35 anchor nodes. In Eq. (5), the weight $w_c$ adapts dynamically, scaling from 0.05 when any sub-region is infeasible to 2.0 when all sub-regions achieve feasibility; the remaining coefficients are set to $\lambda_\Phi = 100$ (for numerical scale alignment), $\lambda_{\text{split}} = 0.05$, and $\lambda_{\text{reject}} = 0.03$. The feasibility threshold for Eq. (4) is set to $\tau = 0.63$. For Eq. (12), the coefficient is $c = 0.85$. For Eq. (35), the terminal parameters are configured as $\lambda_{\text{final}} = 150$, $B_{\text{final}} = 150$, and $\lambda_{\text{call}}^{\text{final}} = 0.1$. The policy network $\pi_\theta$ is optimized using Adam with a learning rate of 0.0001, a discount factor of $\gamma = 0.99$, and a trajectory length of $T = 2$.

# E Details of the Adaptive Partitioner

## E.1 MDP Formulation

We formulate iterative partition refinement as a Markov Decision Process (MDP) defined by $(S, A, \rho, R, \gamma)$. The objective is to learn a policy that produces a partitioning $\mathcal{P}$ satisfying the cognitive feasibility threshold across all regions while minimizing communication overhead. Here, $S$ is the state space, $A$ is the hybrid action space covering node migration and region splitting, $\rho : S \times A \times S \to [0, 1]$ denotes the state transition probability, $R : S \times A \to \mathbb{R}$ is the reward function, and $\gamma \in (0, 1)$ is the discount factor.

## E.2 Initialization and Positional Embeddings

Rather than starting from empty partitions, we initialize the process with a warm start. Specifically, we employ METIS (Stanton and Kliot, 2012) to decompose the graph into $k_0$ initial clusters based on raw node features and positional embeddings. This pre-partitioning provides a structured starting point for subsequent optimization.

To preserve structural proximity, we adopt Lipschitz Embedding (Bourgain, 1985) and compute positional features through Random Walk with Restart (RWR) (Bianchini et al., 2005). Let $\mathcal{V}_{anc} = \{v_1, \ldots, v_\alpha\} \subseteq \mathcal{V}$ be a set of $\alpha$ anchor nodes. For each anchor $v_i$, the steady-state probability vector $\vec{r}_i$ is defined as

$$\vec{r}_i = c\tilde{\mathbf{W}}\vec{r}_i + (1 - c)\vec{e}_i, \tag{12}$$

where $\tilde{\mathbf{W}}$ is the transition matrix, $\vec{e}_i$ is the one-hot indicator for anchor $v_i$, and $c \in (0, 1)$ controls the balance between walking to neighbors and restarting at the anchor. Each node $u$ is represented by

$$\text{pos}(u) = [r_{1u}, \ldots, r_{\alpha u}], \tag{13}$$

where $r_{iu}$ is the $u$-th element of $\vec{r}_i$. We concatenate raw node features with positional encodings to obtain the initial embedding:

$$\text{emb}_{init}(u) = \mathbf{X}[u] \parallel \text{pos}(u), \tag{14}$$

where $\parallel$ denotes concatenation.

## E.3 Graph Encoders

Building upon Eq. (14), we employ Graph Neural Networks (GNNs) (Hamilton et al., 2018) to capture topology-aware node representations for partition refinement. The node migration policy and the region splitting policy use the same message-passing architecture but maintain separate trainable parameters. For each branch $b \in \{\text{mig}, \text{split}\}$, the input representation of node $u$ is initialized as

$$\mathbf{h}_{u,b}^0 = \text{emb}_{init}(u). \tag{15}$$

At layer $l + 1$, the node representation is updated by

$$\mathbf{h}_{u,b}^{l+1} = \mathbf{W}_{1,b}^l \mathbf{h}_{u,b}^l + \mathbf{W}_{2,b}^l \cdot \frac{1}{|\mathcal{N}_u|} \sum_{u' \in \mathcal{N}_u} \mathbf{h}_{u',b}^l, \tag{16}$$

where $\mathcal{N}_u$ denotes the neighborhood of $u$, and $\mathbf{W}_{1,b}^l, \mathbf{W}_{2,b}^l$ are trainable parameters.

### E.4 State and High-level Action Selection

At step $t$, the partition status is denoted as $\mathcal{P}^t = \{P_1^t, \ldots, P_k^t\}$. The state is defined as

$$S^t = \{S_1^t, \ldots, S_k^t, \Phi(\mathcal{P}^t), C_t\}, \tag{17}$$

where

$$S_i^t = \{\mathbf{h}_v^L \mid v \in P_i^t\}, \tag{18}$$

$\Phi(\mathcal{P}^t)$ is the global infeasibility gap, and $C_t$ is the current inter-region communication cost.

The action space $\mathcal{A}$ contains two refinement types: node migration and region splitting. We define the policy input as

$$\mathbf{q}^t = \left[\min_{i \in \mathcal{P}^t} \phi_i^t,\ \Phi(\mathcal{P}^t)\right]. \tag{19}$$

A lightweight policy head first selects the refinement operation $o^t \in \mathcal{O}^t$, where $\mathcal{O}^t$ contains node migration and region splitting:

$$\pi_\theta^{\text{type}}(o^t \mid S^t) = \mathrm{Softmax}_{o^t \in \mathcal{O}^t}\left(\mathrm{MLP}_{\theta_a}(\mathbf{q}^t)\right). \tag{20}$$

To reduce the combinatorial explosion of the action space, we employ heuristic candidate selection to identify candidate nodes and bottleneck regions, while the RL policy focuses on high-level decision optimization.

### E.5 Node Migration

When the node migration action is selected, we identify a structurally misplaced node and reassign it using the migration policy. Let $P(v)$ be the current partition assignment of node $v$, and let

$$\mathcal{N}_p(v) = \{u \in \mathcal{N}(v) \mid P(u) = p\} \tag{21}$$

be the neighbors of $v$ in partition $p$. The structural misalignment of $v$ is quantified by

$$\mathrm{NodeScore}^t(v) = \frac{\max_{p \neq P(v)} |\mathcal{N}_p(v)|}{|\mathcal{N}_{P(v)}(v)|} \cdot \frac{1}{d_v + \epsilon}, \tag{22}$$

where $d_v$ is the node degree and $\epsilon$ is a stability constant. This score highlights fragile boundary nodes that dominate the total edge cut.

Conditioned on the selected node $v$, the compatibility between $v$ and each candidate partition $p \in \mathcal{P}^t$ is computed by aggregating interactions with neighbors in $p$:

$$\mathrm{PartScore}_\theta(p, v) = \mathrm{AGG}_{u \in \mathcal{N}_p(v)}\Big(\mathrm{MLP}_{\theta_m}\left(\mathbf{h}_{v,\text{mig}}^L \parallel \mathbf{h}_{u,\text{mig}}^L\right)\Big). \tag{23}$$

The migration action is sampled from

$$\pi_\theta^{\text{mig}}(a^t = p \mid S^t, v) = \frac{\exp(\mathrm{PartScore}_\theta(p, v))}{\sum_{p' \in \mathcal{P}_{\text{act}}^t} \exp(\mathrm{PartScore}_\theta(p', v))}. \tag{24}$$

If node $v$ is migrated from partition $i$ to partition $j$, the transition is

$$P_i^{t+1} \leftarrow P_i^t \setminus \{v\}, \quad P_j^{t+1} \leftarrow P_j^t \cup \{v\}. \tag{25}$$

### E.6 Region Splitting

When the splitting action is selected, we identify the region that contributes most to the global infeasibility gap. For each partition $P_i \in \mathcal{P}$, its contribution is defined as

$$c_i = |P_i| \cdot \max(0, \tau - \phi_i). \tag{26}$$

The target region is selected as

$$r = \arg\max_{P_i \in \mathcal{P}} c_i. \tag{27}$$

Given the target region $r$, the splitting policy uses the topology-aware representations from the splitting encoder(Eq. (16)). For each node $u \in r$, we construct

$$\mathbf{z}_u = \mathrm{MLP}\left(\mathbf{h}_{u,\text{split}}^L \parallel \mathrm{AGG}_{v \in r}(\mathbf{h}_{v,\text{split}}^L) \parallel \mathbf{s}_r\right), \tag{28}$$

where $\mathbf{s}_r$ encodes region-specific structural attributes, including partition size, internal node degree, boundary degree, boundary ratio, and the estimated feasibility rate $\phi_r$. The binary assignment policy is

$$\pi_\theta^{\text{split}}(b_u^t = 1 \mid S^t, r, u) = \frac{\exp(g_{u,1})}{\exp(g_{u,0}) + \exp(g_{u,1})}, \tag{29}$$

$$\mathbf{g}_u = \mathrm{MLP}_{\theta_s}(\mathbf{z}_u). \tag{30}$$

Node-wise assignments execute a bipartition of $r$. The partition set is updated as

$$\mathcal{P}^{t+1} \leftarrow (\mathcal{P}^t \setminus \{r\}) \cup \{r_1, r_2\}, \tag{31}$$

where

$$r_1 = \{u \in r \mid \hat{y}_u = 0\}, \quad r_2 = \{u \in r \mid \hat{y}_u = 1\}. \tag{32}$$

This operation increases the number of partitions by one and refines the global partitioning granularity.

### E.7 Reward Design

The immediate reward is defined as

$$r_t = w_c(C_{t-1} - C_t) + \lambda_\Phi(\Phi_{t-1} - \Phi_t) - \lambda_{\text{split}}\mathbb{1}_{\text{split}} - \lambda_{\text{reject}}\mathbb{1}_{\text{reject}}, \tag{33}$$

where $\mathbb{1}_{\text{split}}$ penalizes unnecessary split actions to prevent over-segmentation, and $\mathbb{1}_{\text{reject}}$ penalizes ineffective maneuvers, including splitting attempts with no feasibility gain and node migrations that degrade feasibility or fail to reduce communication cost.

To account for long-term partitioning quality, we use discounted returns:

$$D_t = \sum_{j=0}^{T-t} \gamma^j r_{t+j}, \tag{34}$$

where $T$ is the horizon length and $\gamma \in (0, 1]$ is the discount factor.

At the end of an episode, we add a terminal reward:

$$r_T^{\text{final}} = -\lambda_{\text{final}}\Phi_T + B_{\text{final}}\mathbb{1}_{|\mathcal{I}_T|=0} - \lambda_{\text{call}}^{\text{final}} \max(0, C_T - C_0), \tag{35}$$

where $\mathcal{I}_T = \{P_i \in \mathcal{P} \mid \phi_i < \tau\}$ denotes the set of infeasible partitions. The first term penalizes residual infeasibility, the second term rewards global feasibility, and the third term discourages communication degradation.

### E.8 Policy Optimization and Inference

We optimize the policy network by maximizing the expected return. The policy parameters $\theta$ are updated via the REINFORCE gradient estimator (Sutton and Barto, 1998). For an episode with horizon $T$, the policy objective

$$J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} D_t\right] \tag{36}$$

is optimized through

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} D_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]. \tag{37}$$

Here, $a_t$ includes both node migration and region splitting. For splitting actions, $\log \pi_\theta(a_t \mid s_t)$ incorporates the joint probability of branch selection and node-wise Bernoulli sampling. For migration actions, it accounts for the probability of branch selection and target partition selection.

During training, the policy iteratively refines the warm-start partitioning within a bounded horizon $T$, guided by the Cognitive Estimator and the communication simulator. During inference, the learned policy deterministically refines the initial partitioning to obtain subregions that are both feasibility-satisfying and communication-efficient.

## F Prompt Design

This section presents the prompt templates used by the spatial and temporal agents. The spatial agent performs local reasoning and cross-region communication, while the temporal agent ensures temporal validity. The prompt templates for the spatial and temporal agents across the four tasks are shown in the following figures.

## Spatial Agent Prompt for Reachability

**System Prompt.**
You are an expert in region undirected-graph reachability. You will receive `REGION DATA`. `Region_edges` is an undirected graph within your region: each pair `[u,v]` means an undirected edge between nodes $u$ and $v$. `Boundary_edges` lists boundary connections: each pair `[b,h]` means a boundary link from your region's boundary node $b$ to a neighboring region's handoff node $h$. You MUST follow the TASK and RULES to decide your action.

**REGION DATA.**

```
Region_id:        {region_id}
Region_nodes:     {region_nodes}
Boundary_nodes:   {boundary_nodes}
Region_edges:     {edges_str}
Boundary_edges:   {boundary_str}
Previous_path:    {path_history}
Current_node:     {current_node}
Target_node:      {target_node}
```

**TASK.**
Choose exactly one action from `["found", "handoff", "stuck"]`.

**CASE A.** If `target_in_region == true`, find a valid path from `Current_node` to `Target_node`. If a valid path exists, output `action="found"` with `path_segment`. The field `path_segment` may contain multiple paths when multiple valid paths exist. Otherwise, go to CASE B.

**CASE B.** If `target_in_region == false` OR no valid path to the target exists, evaluate boundary handoff candidates. Let `degree(x)` be the number of `Region_edges` containing $x$. If `degree(Current_node) == 0` AND `Current_node` is in `Boundary_nodes`, set `Boundary_nodes = [Current_node]`; otherwise keep `Boundary_nodes` unchanged. For each boundary node $b$, the agent MUST find a valid path from `Current_node` to $b$. If such a path exists, then for each boundary edge $e$ in `Boundary_edges` where `e[0] == b`, let `h = e[1]`. If both $b$ and $h$ are not in `Previous_path`, create a candidate with `boundary_node=b`, `handoff_node=h`, and `path_to_node=path`. The field `path_to_node` may contain multiple paths when multiple valid paths exist. Keep up to `{max_candidates}` candidates, preferring shorter `path_to_node` and diverse `handoff_node`. If candidates are not empty, output `action="handoff"` with candidates.

**OUTPUT JSON.**

```
{"action":"found", "target_in_region":true,  "path_segment":[...], ...}
{"action":"handoff", "target_in_region":true/false,  "candidates":[{"boundary_node":int,
  "handoff_node":int, "path_to_node":[...], ...}]}
{"action":"stuck", "target_in_region":true/false}
```

Figure 9: Prompt template for the spatial agent. The agent performs local reachability reasoning and decides whether to return a found path, hand off to another region, or report failure.

## Temporal Agent Prompt for Reachability

**System Prompt.**
You are an expert agent specialized in temporal reasoning tasks. You will receive a Temporal Definition, Task Instruction, Answer Instruction, and a Question. You must solve the temporal task and output the necessary reasoning and the final answer as "Yes" or "No".

**Temporal Definition.**
A temporal path is a sequence of one or more quadruples $(u_i, v_i, s_i, t_i)$, where the undirected edge $(u_i, v_i)$ is added at time $s_i$, and $t_i$ is the last valid time. Equivalently, the deletion happens at time $t_i + 1$. The path is valid iff there exist traversal times $m_1 \leq \cdots \leq m_k$ where equality is allowed, such that for each $i$, $\max(m_{i-1}, s_i) \leq m_i \leq t_i$, with $m_0 = -1$.

**Validity Check.**
To check validity, greedily assign the earliest possible traversal time for each edge: $m_i = \max(m_{i-1}, s_i)$. If $m_i \leq t_i$ holds for all edges, the path is valid. Remember: any single-quadruple path is valid.

**Task Instruction.**
Your task is to determine whether at least one of the given temporal paths is valid.

**Answer Instruction.**
After `Answer:`, output only "Yes" or "No".

**Question.**
Given all reachable paths between node `{u_query}` and node `{v_query}`:`{paths_repr}`. Does there exist a valid temporal path between node `{u_query}` and node `{v_query}`?

`Answer:`

Figure 10: Prompt template for the temporal agent. The agent verifies whether candidate paths satisfy temporal validity constraints.

## Spatial Agent Prompt for Temporal Motif Counting

**System Prompt.**
You are an expert in undirected graph motif counting. `Region_edges` is an undirected graph: each pair `[u, v]` means an undirected edge between nodes $u$ and $v$. Do NOT output code, pseudocode, or programming-language snippets. Do NOT describe hypothetical execution, e.g., "after executing the code". Execute the requested steps directly in plain text, following the user's rules exactly. Do not replace step-by-step membership checks with shorthand expressions.

**REGION DATA.**

```
Region_id:                  {region_id}
Region_nodes:               {region_nodes}
Region_edges (undirected):  {edges_str}
```

**MOTIF DEFINITION.**
`{motif_type}`: A sequence of `{l}` edges, $M =$ `{motif_edge}`, involving `{k}` unique nodes.

**TASK.**
Find all `{motif_type}` using ONLY `Region_edges`. There is no need to avoid duplicate triangles. Output all triangles as they are found.

**OUTPUT JSON.**
`{"triangles": [[[u0,u1],[u1,u2],[u2,u0]], ...]}`.

Figure 11: Prompt template for the spatial agent in the motif recognition task. The agent constructs adjacency sets within a region and enumerates motif instances using only local region edges.

## Temporal Agent Prompt for Temporal Motif Counting

**System Prompt.**
You are an expert agent specialized in temporal reasoning tasks. Do NOT output code, pseudocode, or programming-language snippets. Do NOT describe hypothetical execution, e.g., "after executing the code". Execute the requested steps directly in plain text, following the user's rules exactly.

**Temporal Definition.**
A temporal motif instance is defined as a sequence of edges $[[u_0, v_0, t_0], \ldots, [u_l, v_l, t_l]]$, where timestamps are strictly distinct and `time_span` $<$ `{delta}`.

**Task.**
Given a list of `{motif_type}` motif instances, count how many are valid.

**Input Data.**
`Motif instances: {instances_str}`.

**OUTPUT JSON.**
`{"Count": count}`.

Figure 12: Prompt template for the temporal agent in the motif recognition task. The agent verifies temporal validity by checking timestamp distinctness and the temporal span of each candidate motif instance.

## Spatial Agent Prompt for Community Detection

**System Prompt.**
You are an expert in community detection using label propagation. Perform exactly ONE asynchronous label-update round for `Region_nodes`. Neighbor relationships come ONLY from `Edges`. `Edges` is undirected: each pair `[u, v]` means an undirected edge between nodes $u$ and $v$. ASYNCHRONOUS means that nodes are updated in the given `Region_nodes` order, and the latest available labels within the same iteration are immediately used.

**REGION DATA.**

```
Region_id:                  {region_id}
Iteration:                  {iteration}
Region_nodes:               {local_nodes}
Neighbor_nodes:             {neighbor_nodes_set}
Edges (Undirected):         {all_edges}
```

**INITIAL LABELS.**
`All_node_labels_prev ({node: label}): {all_label_map}`

**TASK.**
Do exactly ONE asynchronous update round for `Region_nodes` only, and output `updated_labels`.

**IMPORTANT.**
When computing `neighbors(x)`, both endpoints $u$ and $v$ of each edge must be checked to see whether either equals $x$. The agent updates `L[x]` immediately after each node. Therefore, when processing the next node in `Region_nodes`, if it is a neighbor of a previously updated node, it must use the new label of that neighbor. The keys of `updated_labels` must match `Region_nodes` exactly, with no missing or extra nodes. Do NOT output labels for `Neighbor_nodes`. `neighbors(x)` MUST contain all and only the neighbors of $x$, with no missing neighbors and no extra nodes. If `current_label` is in `tied_labels`, the agent MUST keep it and must NOT choose the smallest label. The output must be a dictionary mapping node to label.

**OUTPUT JSON.**
`{"updated_labels": {"Region_nodes": label}}`

Figure 13: Prompt template for the spatial agent in the community detection task. The agent performs one asynchronous label-propagation update round within a region using only undirected edge relationships.

## Temporal Agent Prompt for Community Detection

**System Prompt.**
You are an expert agent specialized in temporal filtering.

**Dynamic Graph Definition.**
A dynamic graph is given as a list of quadruples. Each quadruple `[u, v, t0, t1]` means an undirected edge $(u, v)$ is active from time $t_0$ inclusive to time $t_1$ inclusive. That is, the edge exists for all $t$ such that $t_0 \leq t \leq t_1$.

**Task.**
Given a time window $[\texttt{W_start}, \texttt{W_end}] = [\{\texttt{window_start}\}, \{\texttt{window_end}\}]$, select all edges that exist for the ENTIRE window. An edge satisfies this condition iff $t_0 \leq \texttt{W_start}$ AND $t_1 \geq \texttt{W_end}$.

**Input.**
`Dynamic Graph: {interval_str}`

**Output JSON.**
`{"edges": [[u, v], ...]}`.

Figure 14: Prompt template for the temporal agent in the Community Detection task. The agent selects edges that remain active throughout the given time window.

## Spatial Agent Prompt for Connected Components

**System Prompt.**
You are an expert in undirected graph connected-components counting.

**REGION DATA.**

```
Region_id:                    {region_id}
Region_nodes:                 {region_nodes}
Boundary_nodes:               {boundary_str}
Region_edges (undirected):    {edges_str}
```

**Connected Component Definition.**
In an undirected graph, a connected component is a maximal subgraph in which every pair of nodes is connected by some path.

**TASK.**
Compute all connected components within the region using ONLY `Region_edges`, then output: 1) the number of connected components; 2) for each boundary node, the `component_id` it belongs to.

**Critical.**
Components must cover all `Region_nodes` exactly once, with full cover and no overlap. For nodes that form singleton components, verify that they indeed have no incident edges; otherwise, union them with the corresponding component. `local_cc_count == len(components)`. Keys of `boundary_cc_map` must exactly match `Boundary_nodes`. And for each boundary node, `boundary_cc_map[node]` must be the minimum node id of its component.

**OUTPUT JSON.**
`{"local_cc_count": int, "boundary_cc_map": {"boundary_node": component_id}}`

Figure 15: Prompt template for the spatial agent in the connected components task. The agent computes local connected components within a region and maps each boundary node to its component identifier.

## Temporal Agent Prompt for Connected Components

**System Prompt.**
You are an expert agent specialized in temporal filtering.

**Dynamic Graph Definition.**
A dynamic graph is given as a list of quadruples. Each quadruple $[u, v, t_0, t_1]$ means that an undirected edge $(u, v)$ is active from time $t_0$ inclusive to time $t_1$ inclusive. That is, the edge exists for all $t$ such that $t_0 \leq t \leq t_1$.

**Task.**
Given a time window $[\texttt{W_start}, \texttt{W_end}] = [\{\texttt{window_start}\}, \{\texttt{window_end}\}]$, select all edges that exist for the ENTIRE window. An edge should be selected iff it satisfies: $(t_0 \leq \texttt{W_start})$ AND $(t_1 \geq \texttt{W_end})$.

**Input.**
`Dynamic Graph: {interval_str}`

**Output JSON.**
`{"edges": [[u, v], ...]}`

Figure 16: Prompt template for the temporal agent in the connected components task. The agent filters dynamic edges and returns the edges that remain active throughout the given time window.