CONDENSATION ON DEMAND: ADAPTIVE BUDGET ALLOCATION FOR GRAPH CONDENSATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph condensation constructs a compact synthetic graph for efficient GNN training, yet existing methods typically prescribe class-wise synthetic budgets from label frequencies before condensation. This implicitly treats class size as a proxy for representational demand, although frequency alone does not indicate where additional synthetic capacity is most useful. We instead formulate budget allocation as a sequential intra-class refinement problem, in which, beyond a minimum class-coverage requirement, each additional synthetic node is assigned according to region-level refinement utility. We develop **AUA**, an analytic allocator that scores feasible refinements by the reduction of a decomposable representation-and-semantic residual potential, providing explicit geometric and predictive interpretations. We further introduce **LUA**, a task-grounded extension that reranks analytically promising refinements using lightweight supervision of their predictive marginal gains. Each terminal region defines one synthetic node, while mass-preserving weights keep each class's total supervised training mass independent of its allocated resolution. Experiments on six graph benchmarks show that adaptive allocation improves the use of limited synthetic capacity. AUA enables efficient allocation with low condensation overhead, while LUA builds on AUA's analytic proposals to deliver consistent gains across non-saturated settings and strong performance on large-scale benchmarks and across architectures.

1 INTRODUCTION

Graph-structured data are ubiquitous in applications such as social networks, recommender systems, biological networks, and citation networks (Reiser et al., 2022; Huang et al., 2022; Wu et al., 2022; Brody et al., 2021; Wang et al., 2022). Graph neural networks (GNNs) achieve strong predictive performance by exploiting relational dependencies and higher-order structural information (Gao et al., 2023; Zhang et al., 2024a; Liu et al., 2023b), but training them on large graphs remains costly due to repeated multi-hop message passing (Gilmer et al., 2017; Zhou et al., 2023; Hashemi et al., 2024; Wu et al., 2024). This cost is amplified when models must be trained repeatedly, as in hyperparameter tuning, architecture search, and continual refinement (Zhang et al., 2024b; Fang et al., 2023; Jiang et al., 2024).

Graph condensation (GC) addresses this problem by constructing a compact synthetic graph that enables efficient GNN training while preserving predictive utility (Jin et al., 2021; Zheng et al., 2023; Gao et al., 2025c; Xu et al., 2024). Existing methods mainly focus on how the synthetic graph should be optimized, including its node features (Jin et al., 2021; Zheng et al., 2023; Zhang et al., 2024c; Liu et al., 2024), structure and model behavior (Wang et al., 2024; Xu et al., 2023), or data distributions (Liu et al., 2022; 2023a; Xiao et al., 2024). In most graph condensation pipelines, however, class-wise synthetic allocation is prescribed before condensation rather than determined by the evolving representational demand. This leaves a complementary question unresolved: *where should each additional unit of synthetic capacity be allocated?*

A common practice is to prescribe class-wise synthetic budgets according to the label proportions of the original graph, which we refer to as *ratio-driven allocation*. This implicitly treats class frequency as a proxy for representational demand. Yet frequency measures how many samples a class contains, not how difficult those samples are to distinguish or how valuable additional representational capacity would be for them. Consequently, a frequent but already well-represented class may

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

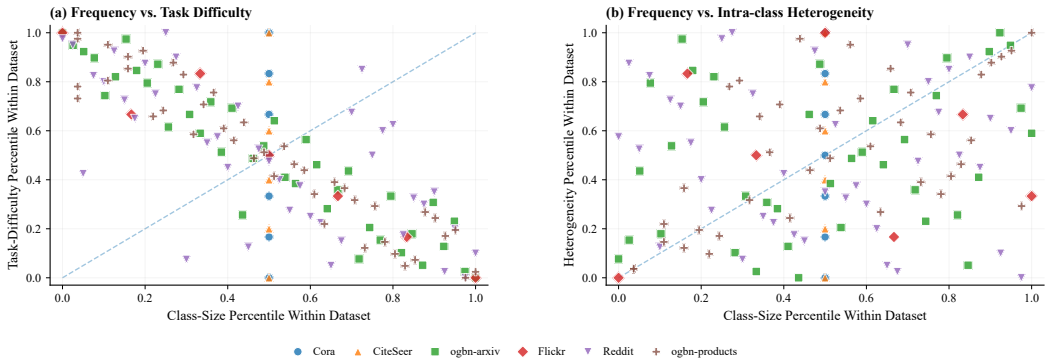


Figure 1: **Class frequency is an unreliable proxy for representational demand.** Each point denotes a class using within-dataset percentile ranks. (a) Class frequency versus cross-fitted predictive difficulty. (b) Class frequency versus intra-class heterogeneity. The dashed diagonal denotes perfect rank agreement. Classes with comparable frequencies can exhibit substantially different difficulty and internal heterogeneity.

receive excessive synthetic capacity, while a smaller class with greater refinement demand remains under-represented.

We examine this mismatch in Fig. 1 across six graph benchmarks by comparing class frequency with two demand-relevant quantities: cross-fitted predictive difficulty and intra-class heterogeneity. To remove dataset-specific scale differences, all quantities are converted to within-dataset percentile ranks. Classes with similar or identical frequencies can exhibit substantially different predictive difficulty and internal heterogeneity, indicating that class size alone does not reliably characterize where additional representational capacity is needed. Together, these observations motivate allocation based on refinement demand rather than class frequency alone.

Motivated by this mismatch, we recast synthetic budget allocation as a *sequential region refinement* problem. Rather than deciding how many synthetic nodes each class should receive beforehand, we ask: which currently represented region should receive the next unit of synthetic capacity? Starting from one coarse region per class, each refinement assigns one additional representative to a selected region by splitting it into two finer subregions. Beyond the minimum coverage floor, regional demand is expressed through the marginal utility of refinement, and the remaining class-wise budgets emerge from the resulting sequence of local decisions. As illustrated in Fig. 2, this formulation allows regions from different classes to compete directly for a globally limited budget.

We develop **AUA**, an analytic allocator that measures how much representation and semantic residual can be resolved by each feasible refinement. Its utility admits exact geometric and predictive interpretations and induces greedy descent on an explicit decomposable potential. However, analytically recoverable structure need not perfectly reflect downstream predictive value. We therefore introduce **LUA** as a task-grounded extension of AUA: AUA first proposes analytically promising refinements, while LUA learns to rerank them using lightweight signals of immediate refinement gain, remaining complexity, and regional task difficulty. This analytic-proposal/learned-reranking design preserves efficient search while accounting for the mismatch between structural refinability and predictive utility.

Finally, each terminal region is instantiated as one synthetic node. To decouple representation resolution from total training mass, we preserve the mass associated with each class regardless of how many representatives it receives. Thus, adaptive allocation increases representational granularity without mechanically increasing a class's supervised training weight.

Our contributions are summarized as follows:

- We identify synthetic budget allocation as an underexplored dimension of graph condensation and show that class frequency alone provides limited guidance about representational demand.

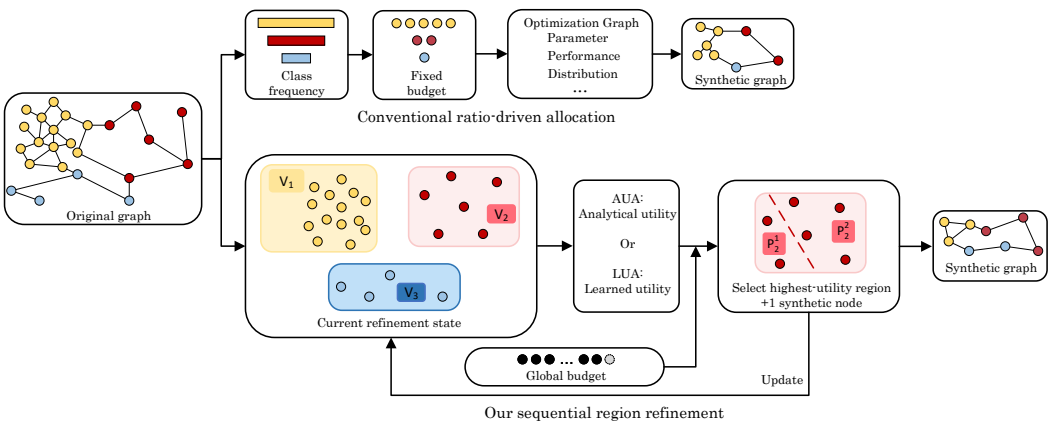


Figure 2: **Ratio-driven allocation versus sequential region refinement.** (a) Conventional methods predetermine class-wise synthetic budgets from label proportions. (b) We allocate each additional synthetic node according to region-level marginal utility, allowing class-wise budgets to emerge from sequential refinement.

- We formulate budget allocation as sequential region refinement, where each additional synthetic node is assigned according to marginal refinement utility and class-wise budgets emerge from region-level decisions.
- We develop AUA, an efficient analytic allocator based on a decomposable representation-and-semantic potential, and LUA, its task-grounded extension for reranking analytically promising refinements. Experiments on six graph benchmarks demonstrate effective and efficient allocation across diverse condensation regimes.

2 Related Work

Graph Reduction and Graph Condensation. Graph reduction improves scalability by constructing compact graphs that preserve useful information, including graph sparsification (Althöfer et al., 1993; Spielman & Srivastava, 2008; Batson et al., 2013) and graph coarsening (Loukas, 2019; Huang et al., 2021; Wickman et al., 2022). Graph condensation (GC) instead learns a small synthetic graph optimized for downstream GNN training (Jin et al., 2021; Zheng et al., 2023; Gao et al., 2025c). Existing GC methods primarily differ in their optimization objectives, including parameter or trajectory matching (Jin et al., 2021; 2022; Zheng et al., 2023), performance matching with kernel-based objectives (Wang et al., 2024; Xu et al., 2023), and distribution matching (Liu et al., 2022; 2023a; Xiao et al., 2024). Recent methods such as CGC further simplify condensation by reformulating class-level matching as a training-free class-partition problem (Gao et al., 2025a). While these studies mainly focus on how synthetic graphs are constructed or optimized, class-wise synthetic budgets are typically prescribed beforehand, often independently of local representational demand; our work instead determines synthetic allocation through sequential region-level refinement.

Adaptive and Generalized Condensation Settings. Recent studies have extended condensation beyond standard static settings. MCond considers inductive graph condensation (Gao et al., 2024a), while FGD, RobGC, and OpenGC study fairness preservation, robustness, and open-world generalization, respectively (Feng et al., 2023; Gao et al., 2025b; 2024b). These works broaden condensation along complementary dimensions, whereas our focus is the allocation of limited synthetic capacity.

A closely related recent work is C^2TC, which studies class-adaptive cardinality for tabular data condensation (Xu et al., 2026). C^2TC directly adjusts class-wise cluster counts according to a clustering objective, whereas our formulation allocates each additional synthetic representative to a specific current region according to its marginal refinement utility. Thus, class-wise budgets are not directly optimized but emerge from sequential region-level decisions. Our utilities further exploit graph-propagated representations and predictive structure, with LUA incorporating task-grounded supervision.

3 METHOD

3.1 PROGRESSIVE MARGINAL ALLOCATION

Given a node-attributed graph $G = (V, A, X, Y)$ with C classes, graph condensation constructs a synthetic graph $G' = (A', X', Y')$ under a node budget $B \ll |V|$. Let $V_{\text{obs}} \subseteq V$ denote the nodes observable during condensation and $V_{\text{tr}} \subseteq V_{\text{obs}}$ the labeled training nodes. For class c, define

$$V_c = \{i \in V_{\text{tr}} : y_i = c\}, \qquad n_c = |V_c|,$$

and let C_+ be the number of non-empty classes. We initialize one class-pure region per non-empty class,

$$\mathcal{S}_0 = \{V_c : |V_c| > 0\}. \tag{1}$$

Each splittable region P is associated with a single deterministic binary refinement $\text{Split}(P) = \{P^{(1)}, P^{(2)}\}$, constructed using the task-aware representation defined below. Let $\mathcal{A}(\mathcal{S}_t)$ denote the splittable regions and

$$\mathcal{U}(\mathcal{S}_t, P) = (\mathcal{S}_t \setminus \{P\}) \cup \{P^{(1)}, P^{(2)}\} \tag{2}$$

the corresponding state update. The splitter is shared by all allocation policies, separating *how* a region is refined from *which* region receives the next unit of capacity.

We further introduce a class-independent minimum coverage parameter $b_{\min} \geq 1$. Before every non-empty class reaches this minimum, refinements are restricted to under-covered classes; afterward, all splittable regions compete globally. We denote the resulting eligible set by $\mathcal{A}_{\text{cov}}(\mathcal{S}_t)$, with details provided in Appendix A.3. When $b_{\min} = 1$, global competition begins directly from the initial state.

Each refinement increases the number of terminal regions by one. After $T = B - C_+$ refinements, the class-wise synthetic budget is

$$b_c(B) = |\{P \in \mathcal{S}_T : P \subseteq V_c\}|, \qquad \sum_{c=1}^{C} b_c(B) = B. \tag{3}$$

Class-wise budgets emerge from region-level refinement, with $b_c(B) \leq n_c$. Since every non-singleton region remains splittable, $B = |V_{\text{tr}}|$ yields a policy-invariant singleton partition.

To score refinements efficiently, we construct a lightweight task-aware space from propagated observable features,

$$H^{(0)} = \text{Proj}(X_{\text{obs}}), \qquad H^{(h)} = \widehat{A}_{\text{obs}} H^{(h-1)}, \qquad h_i = [\overline{H}_i^{(0)}; \ldots; \overline{H}_i^{(K)}]. \tag{4}$$

Here, $\overline{H}^{(h)}$ denotes the standardized h-hop representation obtained after the initial feature projection, and we use $z_i = h_i$ as the task-aware representation. A cross-fitted lightweight predictor separately provides out-of-fold logits s_i and posteriors $p_i \in \Delta^{C-1}$.

Let $\mathcal{Q}(\mathcal{S})$ denote the ideal downstream utility of state $\mathcal{S}$. The fixed-budget allocation problem is

$$\max_{\{P_t\}_{t=0}^{T-1}} \mathcal{Q}(\mathcal{S}_T), \qquad \mathcal{S}_{t+1} = \mathcal{U}(\mathcal{S}_t, P_t), \quad P_t \in \mathcal{A}_{\text{cov}}(\mathcal{S}_t). \tag{5}$$

The ideal marginal utility of refining P is

$$\Delta_{\mathcal{Q}}(\mathcal{S}_t, P) = \mathcal{Q}(\mathcal{U}(\mathcal{S}_t, P)) - \mathcal{Q}(\mathcal{S}_t). \tag{6}$$

Since evaluating it would require repeated downstream optimization, AUA replaces it with an analytic refinement surrogate, while LUA uses task-grounded supervision to rerank analytically promising candidates.

3.2 ANALYTIC UTILITY ALLOCATION

AUA replaces the inaccessible downstream marginal in Eq. 6 with an analytic surrogate that measures the residual structure resolved by a candidate refinement. For a region $P \subseteq V_c$, define

$$\begin{aligned} \mu_P &= \frac{1}{|P|} \sum_{i \in P} z_i, \qquad D(P) = \sum_{i \in P} \|z_i - \mu_P\|_2^2, \\ \bar{p}_P &= \frac{1}{|P|} \sum_{i \in P} p_i, \qquad J(P) = \sum_{i \in P} \text{KL}(p_i \,\|\, \bar{p}_P). \end{aligned} \tag{7}$$

Here, $D(P)$ measures representation dispersion, while $J(P)$ captures disagreement among cross-fitted predictive posteriors.

Let $\gamma_c = \max\{D(V_c), \varepsilon\}$ with $\varepsilon > 0$. We define the class-normalized residual potential

$$\psi(P) = \frac{D(P)}{\gamma_c} + \lambda_{\text{sem}} \frac{J(P)}{n_c \log C}, \qquad \lambda_{\text{sem}} \geq 0. \tag{8}$$

For any partition $\mathcal{P}_c$ of V_c, $\sum_{P \in \mathcal{P}_c} \psi(P) \leq 1 + \lambda_{\text{sem}}$. Thus, both residuals are measured on a class-relative scale, avoiding direct scaling of cross-class utilities with class size or raw dispersion.

For the feasible refinement $P \to \{P^{(1)}, P^{(2)}\}$ produced by the fixed splitter, AUA defines the utility as the residual potential removed by one additional representative:

$$u^{\text{AUA}}(P) = \psi(P) - \psi(P^{(1)}) - \psi(P^{(2)}) = \Delta_{\text{rep}}(P) + \lambda_{\text{sem}} \Delta_{\text{sem}}(P), \tag{9}$$

where

$$\Delta_{\text{rep}}(P) = \frac{D(P) - D(P^{(1)}) - D(P^{(2)})}{\gamma_c},$$
$$\Delta_{\text{sem}}(P) = \frac{J(P) - J(P^{(1)}) - J(P^{(2)})}{n_c \log C}. \tag{10}$$

Hence, AUA measures not merely how heterogeneous a region is, but how much of that heterogeneity its next feasible refinement can actually resolve.

Proposition 1 (Exact refinement decomposition) *Let* $n_a = |P^{(a)}|$, $\mu_a = \mu_{P^{(a)}}$, $\rho_a = n_a/|P|$, *and* $\boldsymbol{\rho} = (\rho_1, \rho_2)$. *Then*

$$\Delta_{\text{rep}}(P) = \frac{n_1 n_2}{|P| \gamma_c} \|\mu_1 - \mu_2\|_2^2, \qquad \Delta_{\text{sem}}(P) = \frac{|P|}{n_c \log C} \, \text{JS}_{\boldsymbol{\rho}}(\bar{p}_{P^{(1)}}, \bar{p}_{P^{(2)}}) \, . \tag{11}$$

Proposition 1 shows that both terms are non-negative and admit direct interpretations: the representation term is the exact reduction in within-region distortion, while the semantic term is the size-weighted Jensen–Shannon separation between the predictive distributions of the two children. AUA therefore quantifies the *refinability* exposed by the fixed splitter: a region is prioritized when replacing one representative by two exposes distinct representation or predictive structure.

The class normalization also yields replication invariance. When the floor in γ_c is inactive, uniformly replicating a class while preserving its representations, posteriors, and split structure leaves both Δ_{rep} and Δ_{sem} unchanged. Thus, pure replication that introduces no new represented or predictive structure does not increase the analytic refinement utility. Proofs are provided in Appendix B.

Finally, let $\Phi(\mathcal{S}) = \sum_{P \in \mathcal{S}} \psi(P)$ denote the global residual potential. Since a refinement changes only its selected region,

$$\Phi(\mathcal{S}_t) - \Phi(\mathcal{U}(\mathcal{S}_t, P)) = u^{\text{AUA}}(P), \qquad P_t^{\text{AUA}} = \arg \max_{P \in \mathcal{A}_{\text{cov}}(\mathcal{S}_t)} u^{\text{AUA}}(P). \tag{12}$$

AUA therefore allocates each additional synthetic node to the currently eligible refinement that maximally reduces the class-normalized residual potential.

3.3 Learning-based Utility Allocation

AUA provides an analytic measure of *refinability*, but a large analytic refinement gain need not translate into proportional predictive benefit. LUA therefore uses AUA as an efficient proposal mechanism and learns to rerank analytically promising refinements using lightweight task-grounded supervision.

Task-grounded marginal supervision. At state $\mathcal{S}_t$, AUA first identifies up to m analytically promising eligible refinements. Let

$$\mathcal{M}_t \subseteq \mathcal{A}_{\text{cov}}(\mathcal{S}_t) \tag{13}$$

denote the up to m candidate regions with the largest $u^{\text{AUA}}(P)$ values. This proposal step restricts learned decision making to refinements that are already plausible under the analytic criterion.

To estimate candidate-level task value without repeatedly training a downstream GNN, we construct lightweight held-out supervision entirely within the labeled training set. For each supervision split, V_{tr} is partitioned into disjoint fitting and calibration subsets, $\mathcal{V}_{\mathrm{fit}}$ and $\mathcal{V}_{\mathrm{cal}}$. Candidate states and ridge probes are constructed from $\mathcal{V}_{\mathrm{fit}}$, while $\mathcal{V}_{\mathrm{cal}}$ provides the held-out supervision used to evaluate probe-level marginal gains. The node-level task representations are constructed as described above, while the predictive quantities used during supervision are cross-fitted within the corresponding fitting subset.

For each active fitting region P, its task-space mean $\bar{z}_P^{\mathrm{fit}}$ is used as a region representative. The resulting state-dependent representative set $\mathcal{R}_{\mathrm{fit}}(\mathcal{S})$ is used to fit a lightweight ridge probe $\beta_{\mathcal{S}}$. Its held-out loss is

$$\mathcal{L}_{\mathrm{cal}}(\mathcal{S}) = \frac{1}{|\mathcal{V}_{\mathrm{cal}}|} \sum_{i \in \mathcal{V}_{\mathrm{cal}}} \|h_{\beta_{\mathcal{S}}}(z_i) - e_{y_i}\|_2^2 . \tag{14}$$

For a candidate refinement P, we define its probe-level marginal gain as

$$r_t(P) = \mathcal{L}_{\mathrm{cal}}(\mathcal{S}_t) - \mathcal{L}_{\mathrm{cal}}\left(\mathcal{U}(\mathcal{S}_t, P)\right). \tag{15}$$

Thus, $r_t(P) > 0$ indicates that assigning the next representative to P reduces held-out probe loss. These held-out marginals serve only as supervision targets for the reranker and are not evaluated during final allocation. During supervision collection, the analytic proposal set is additionally augmented with a small number of eligible refinements to improve local coverage. Further details are provided in Appendix C.

Learning refinement preferences. LUA represents each candidate using three factors: the structure resolved by the refinement, the complexity remaining afterward, and the predictive difficulty of the parent region. We denote them by $\mathcal{G}(P)$, $\mathcal{R}(P)$, and e_P, and define

$$\varphi(P) = (\mathcal{G}(P), \mathcal{R}(P), e_P), \qquad u_\theta^{\mathrm{LUA}}(P) = g_\theta(\varphi(P)). \tag{16}$$

The regional predictive difficulty is

$$e_P = \frac{1}{|P|} \sum_{i \in P} -\log p_i[y_i]. \tag{17}$$

During LUA supervision, the posterior-dependent terms in $\varphi(P)$ are computed from out-of-fold estimates constructed within the corresponding fitting subset, and candidate regions are restricted to the same subset. During final allocation, the full-training out-of-fold estimates in Eq. 29 are used.

Because allocation depends on within-state candidate ordering, LUA converts $r_t(P)$ into normalized preference targets and trains g_θ to reproduce this ordering. Detailed descriptor definitions and the learning objective are provided in Appendices C.2 and C.

Reranking policy. Reranker supervision is first collected from states visited by an AUA trajectory and is subsequently augmented with states induced by the learned policy to reduce train–deployment mismatch. At deployment, probe marginals are no longer evaluated. AUA constructs the analytic proposal set $\mathcal{M}_t$, and LUA selects

$$P_t^{\mathrm{LUA}} = \arg\max_{P \in \mathcal{M}_t} u_\theta^{\mathrm{LUA}}(P), \qquad \mathcal{S}_{t+1} = \mathcal{U}\left(\mathcal{S}_t, P_t^{\mathrm{LUA}}\right). \tag{18}$$

Thus, AUA provides an analytically promising candidate set, while LUA uses task-grounded evidence to correct its local ordering, accounting for the gap between analytic refinability and predictive utility without requiring probe evaluation at deployment.

3.4 Synthetic Graph Construction

After $T = B - C_+$ refinements, the terminal state $\mathcal{S}_T = \{P_1, \ldots, P_B\}$ directly defines the B synthetic nodes. For each class-pure region $P_k \subseteq V_c$, we construct its synthetic representative as the arithmetic mean of the member features,

$$x'_k = \frac{1}{|P_k|} \sum_{i \in P_k} x_i, \qquad y'_k = c. \tag{19}$$

Thus, additional budget increases representation resolution without mixing class identities.

To decouple representation resolution from total training mass, we preserve the mass associated with each class. Each representative $P_k \subseteq V_c$ receives

$$w_k = \pi_c \frac{|P_k|}{n_c}, \qquad \sum_{k:P_k \subseteq V_c} w_k = \pi_c, \tag{20}$$

where π_c is independent of the emergent class budget $b_c(B)$. Under the empirical class prior $\pi_c = n_c/|V_{\mathrm{tr}}|$, this reduces to $w_k = |P_k|/|V_{\mathrm{tr}}|$. Hence, adaptive allocation changes how finely a class is represented without mechanically increasing its total contribution to downstream optimization.

After allocation, graph structure is recovered by projecting source connectivity onto the terminal representatives. Let $\mathbf{Q} \in \mathbb{R}_+^{N_{\mathrm{obs}} \times B}$ denote a row-stochastic node-to-representative routing matrix. Training nodes inherit their terminal-region assignments exactly. In the transductive setting, observable non-training nodes are sparsely routed to compatible terminal representatives using task-aware and training-derived predictive information, without accessing non-training labels. The source connectivity is then projected as

$$\widetilde{A}' = \mathbf{Q}^\top A_{\mathrm{obs}} \mathbf{Q}. \tag{21}$$

Thus, allocation determines synthetic-node resolution, while topology is recovered afterward from the represented source connectivity. Further construction details are provided in Appendix D.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate on six node classification benchmarks: four transductive datasets, Cora, Citeseer (Kipf & Welling, 2016), ogbn-arxiv, and ogbn-products (Hu et al., 2020), and two inductive datasets, Flickr and Reddit (Zeng et al., 2019). We use the standard public splits. For transductive datasets, graph topology and node features are available during condensation, while only training labels provide supervision; for inductive datasets, condensation is restricted to the training subgraph. Dataset statistics are provided in Appendix F.1.

Baselines and evaluation. We compare against ten representative graph reduction and condensation methods: A-ConvMatch (A-CM), GCond, GCDM, DosCond, SGDD, SimGC, EXGC, GCSR, CGC, and SFGC. Following standard GC protocols (Jin et al., 2021; Gao et al., 2025a), we evaluate three condensation ratios per dataset under matched synthetic-node budgets. Unless otherwise specified, a two-layer GCN is used for evaluation, with results reported as mean $\pm$ standard deviation over five runs. Implementation details are provided in Appendix F.4.

Implementation. The task-aware space is constructed from projected multi-hop node features, while cross-fitted ridge predictions provide separate predictive descriptors. For LUA, fitting and calibration subsets are drawn exclusively from the labeled training set; fitting-side states and ridge probes are constructed from the fitting subset, while the calibration subset provides probe-level marginal supervision. The official validation labels are used only for hyperparameter selection.

Further implementation and synthetic graph construction details are provided in Appendices A and D.

4.2 Main Results

Overall Performance. Table 1 shows that LUA achieves the highest mean accuracy in 11 of the 18 settings, including 11 of the 12 settings on the four large-scale benchmarks, with gains of up to 2.4 percentage points over the best compared baseline. Across the 16 non-saturated settings, LUA consistently improves over AUA by an average of 0.76 percentage points, supporting the benefit of task-grounded reranking when the allocation policy retains refinement freedom. At the largest budgets on Cora and Citeseer, where $B = |V_{\mathrm{tr}}|$, the refinement process reaches the singleton boundary and no allocation freedom remains. The gains are particularly consistent on the four large-scale benchmarks.

Table 1: Node classification accuracy (%) at different condensation ratios. Bold and underlined values indicate the best and second-best condensation results, respectively.

		Baselines										Ours		
Dataset	r (%)	A-CM	GCond	GCDM	DosCond	SGDD	SimGC	EXGC	GCSR	CGC	SFGC	AUA	LUA	Full Graph
Cora	1.30	74.6±0.1	79.8±1.3	69.4±1.3	80.5±0.1	80.1±0.7	80.8±2.3	82.0±0.4	79.9±0.7	**82.7±0.3**	80.1±0.4	81.2±0.1	81.6±0.2	81.1±0.2
	2.60	72.8±0.2	80.1±0.6	77.2±0.4	80.1±0.5	80.6±0.8	80.9±2.6	81.9±1.0	80.6±0.8	**82.3±1.3**	81.7±0.5	80.9±0.1	82.0±0.2	
	5.20	78.0±0.3	79.3±0.3	79.4±0.1	80.3±0.4	80.4±1.6	82.1±1.3	82.3±0.9	81.2±0.9	**82.5±0.6**	81.6±0.8	81.4±0.1	81.4±0.1	
Citeseer	0.90	65.1±0.1	70.5±1.2	62.0±0.1	71.0±0.2	69.5±0.4	**73.8±2.5**	69.2±2.0	70.2±1.1	72.5±0.2	71.4±0.5	69.7±0.3	71.5±0.1	71.5±0.1
	1.80	66.0±0.2	70.6±0.9	69.5±1.1	71.2±0.2	70.2±0.8	72.2±0.5	70.1±0.7	71.7±0.9	**72.4±0.2**	**72.4±0.4**	69.6±0.2	71.7±0.2	
	3.60	66.1±0.2	69.8±1.4	69.8±0.2	70.7±0.1	70.3±1.7	71.1±2.8	70.6±0.9	**74.0±0.4**	72.0±0.5	70.6±0.7	70.6±0.1	70.6±0.1	
ogbn-arxiv	0.05	58.0±0.1	59.2±1.1	59.3±0.3	62.1±0.3	60.8±1.3	64.7±0.8	57.6±0.6	60.6±1.1	64.1±0.4	**65.5±0.6**	64.5±0.1	65.3±0.5	71.1±0.2
	0.25	60.0±0.3	63.2±0.3	59.6±0.4	63.5±0.1	65.8±1.2	66.4±0.3	62.3±0.3	65.4±0.8	66.4±0.1	66.1±0.3	66.1±0.2	**66.7±0.3**	
	0.50	61.0±0.2	64.0±0.4	62.4±0.1	63.7±0.2	66.3±0.7	66.8±0.4	65.0±0.8	65.9±0.6	67.2±0.4	66.8±0.2	66.7±0.1	**67.4±0.2**	
ogbn-products	0.03	58.8±0.9	64.2±0.1	66.1±0.1	62.3±0.3	64.5±0.1	63.7±1.1	62.1±0.7	66.5±0.2	68.0±0.1	67.1±0.2	68.3±0.2	**68.7±0.2**	73.1±0.1
	0.05	60.1±0.6	64.7±0.2	67.4±0.4	63.6±0.1	64.9±0.1	64.9±1.2	64.7±1.4	67.8±0.3	68.9±0.3	67.9±0.3	68.4±0.2	**69.1±0.5**	
	0.10	62.4±0.9	65.0±0.1	68.4±0.3	65.8±0.2	64.7±0.1	65.2±1.4	66.4±0.7	68.5±0.3	69.1±0.2	70.1±0.3	69.3±0.3	**70.3±0.3**	
Flickr	0.10	42.2±0.1	46.5±0.4	46.1±0.1	46.0±0.3	46.5±0.1	45.3±0.7	47.0±0.1	46.6±0.3	46.8±0.0	46.6±0.2	49.0±0.1	**49.4±0.5**	47.5±0.1
	0.50	45.2±0.3	47.1±0.1	46.8±0.1	46.2±0.2	46.4±0.2	45.6±0.4	48.3±0.5	46.6±0.2	47.1±0.1	47.0±0.1	49.2±0.2	**49.8±0.3**	
	1.00	45.1±0.1	47.1±0.1	46.7±0.1	46.1±0.1	46.3±0.1	43.8±1.5	48.4±0.9	46.8±0.2	47.0±0.1	47.1±0.1	49.4±0.1	**49.7±0.4**	
Reddit	0.05	72.2±1.2	88.0±1.8	89.3±0.1	89.8±0.1	90.3±0.1	89.6±0.6	90.2±0.1	91.4±0.2	90.6±0.2	89.7±0.2	92.8±0.1	**93.1±0.4**	94.2±0.1
	0.10	73.5±1.0	89.6±0.7	89.7±0.2	90.5±0.1	90.7±0.1	90.6±0.3	90.6±0.9	91.2±0.2	91.4±0.1	90.0±0.3	92.9±0.2	**93.5±0.2**	
	0.20	75.1±1.3	90.1±0.5	90.2±0.4	91.1±0.1	91.3±0.2	91.4±0.2	91.8±0.7	92.2±0.1	91.6±0.2	90.3±0.3	93.2±0.1	**93.5±0.3**	

Table 2: Condensation time (seconds) on six benchmark datasets. Lower is better.

Dataset	A-CM	GCond	GCDM	DosCond	SGDD	SimGC	EXGC	GCSR	CGC	SFGC	AUA	LUA
Cora (2.60%)	8.4	475.2	61.4	377.5	133.7	238.9	229.3	672.8	4.1	2,345.7	1.2	15.6
Citeseer (1.80%)	106.3	532.7	66.2	351.9	124.4	256.3	217.6	593.4	5.2	2,363.8	1.5	21.4
ogbn-arxiv (0.25%)	203.8	17,643.5	713.6	12,712.3	846.4	261.8	375.7	6,118.4	13.4	91,631.2	6.2	136.3
Flickr (0.50%)	178.2	1,297.3	198.7	1,123.8	1,354.6	377.3	407.2	1,157.4	86.7	69,995.8	2.8	144.6
Reddit (0.10%)	237.6	20,479.4	1,037.2	19,101.7	13,144.8	556.4	734.3	1,812.7	93.2	312,697.5	5.1	76.3
ogbn-products (0.05%)	6,551.4	27,689.7	8,216.3	25,379.6	23,518.3	4,749.5	5,138.6	21,245.4	456.3	391,142.3	41.2	537.6

Efficiency Analysis. Table 2 shows that AUA is the fastest method on all six benchmarks. Despite the additional cost of task-grounded supervision, LUA remains among the three fastest methods on five of the six datasets while consistently improving accuracy over AUA. Figure 3 highlights this accuracy–efficiency trade-off.

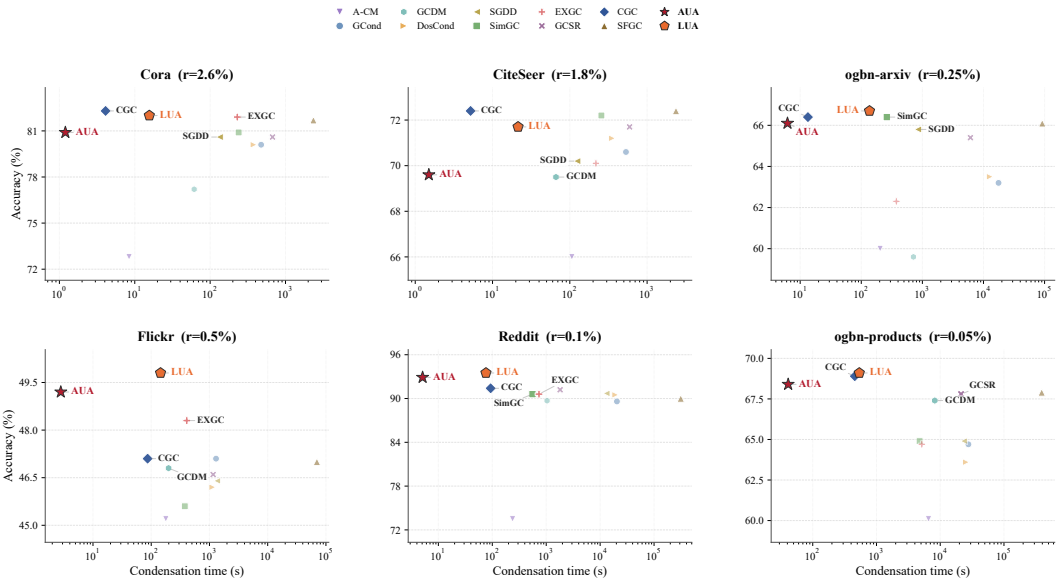


Figure 3: Accuracy–efficiency trade-off at the middle condensation ratio of each dataset. Condensation time is shown on a log scale; upper-left is better.

Table 3: Architecture-specific results are reported as mean ± standard deviation over five runs. Avg. denotes the mean of the six architecture-wise mean accuracies.

Method	MLP	SGC	GAT	GCN	SAGE	APPNP	Avg.
GCond	41.6±0.1	56.1±0.2	55.3±0.3	64.7±0.2	58.2±0.3	62.6±0.2	56.4
GCDM	45.4±0.1	59.7±0.2	66.1±0.2	67.4±0.4	61.6±0.3	66.1±0.2	61.1
SimGC	45.6±0.3	62.1±0.4	63.3±0.4	64.9±1.2	60.4±0.3	63.5±0.2	60.0
CGC	45.9±0.2	63.9±0.2	65.8±0.3	68.9±0.3	63.3±0.1	67.4±0.1	62.5
AUA	**46.3±0.2**	63.6±0.2	65.7±0.3	68.4±0.2	62.7±0.3	67.2±0.2	62.3
LUA	46.2±0.2	**64.5±0.3**	**66.2±0.2**	**69.1±0.5**	**64.1±0.2**	**68.3±0.2**	**63.1**

Table 4: Ablation study on representative datasets. Bold values denote the best mean accuracy within each method block.

	AUA			LUA			
Dataset	Full	w/o Sem.	w/o MH	Full	Direct Reg.	w/o $\mathcal{R}(P)$	w/o e_P
Flickr (0.10%)	**49.0±0.1**	48.2±0.1	48.5±0.2	**49.4±0.5**	48.9±0.5	49.1±0.2	49.2±0.4
Reddit (0.05%)	**92.8±0.1**	92.4±0.2	92.3±0.1	**93.1±0.4**	92.7±0.7	92.5±0.6	92.3±0.4
ogbn-products (0.03%)	**68.3±0.2**	66.7±0.2	67.2±0.1	**68.7±0.2**	68.2±0.3	68.5±0.5	68.4±0.3

Cross-Architecture Generalization. We evaluate the condensed graphs on ogbn-products (0.05%) using MLP, SGC, GAT, GCN, GraphSAGE, and APPNP. For each run, the same condensed graph is reused across all architectures, with results reported over five runs. As shown in Table 3, LUA achieves the highest average accuracy and performs best on five of the six architectures. This indicates that the resulting condensed graphs transfer beyond the GCN evaluation backbone.

Ablation Study. Table 4 shows that removing semantic utility or multi-hop propagation degrades AUA, while direct utility regression and removing either $\mathcal{R}(P)$ or e_P reduce LUA performance. These results support the complementary roles of analytic refinement and task-grounded preference modeling.

Controlled Allocation Analysis. Under the same condensation pipeline and synthetic budget, LUA outperforms ratio-driven and uniform allocation on all six benchmarks, while AUA improves over the stronger static baseline on four (Table 5). This supports the benefit of adaptive allocation; detailed dynamics are given in Appendix G.

Table 5: Controlled comparison of allocation strategies under a matched condensation pipeline and synthetic budget. Bold and underlined values denote the best and second-best results.

	Static Allocation		Adaptive Allocation	
Dataset	Ratio-driven	Uniform	AUA	LUA
Cora (2.60%)	80.4±0.1	80.4±0.1	80.9±0.1	**82.0±0.2**
Citeseer (1.80%)	69.9±0.1	69.9±0.1	69.6±0.2	**71.7±0.2**
ogbn-arxiv (0.25%)	64.7±0.3	55.5±0.2	66.1±0.2	**66.7±0.3**
Flickr (0.50%)	45.6±0.2	42.6±0.1	49.2±0.2	**49.8±0.3**
Reddit (0.10%)	93.2±0.2	91.5±0.2	92.9±0.2	**93.5±0.2**
ogbn-products (0.05%)	68.0±0.3	62.8±0.2	68.4±0.2	**69.1±0.5**

5 Conclusion

We revisit graph condensation from the perspective of synthetic budget allocation and formulate it as progressive intra-class refinement guided by marginal utility. We develop AUA, an efficient analytic allocator, and LUA, a task-grounded extension that reranks analytically promising refinements. Experiments on six benchmarks show that adaptive allocation improves the use of limited synthetic capacity, with AUA providing high efficiency and LUA offering stronger predictive performance and cross-architecture transfer. These results establish budget allocation as a complementary design dimension in graph condensation.

AI USE STATEMENT

LLM Usage. Large language models were used to assist with understanding technical concepts, editing and improving the clarity of the manuscript, including grammar, spelling, and word choice, and facilitating the implementation and execution of experiments. All technical content, experimental results, and conclusions were independently verified by the authors, who take full responsibility for the final manuscript.

REFERENCES

Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

Joshua Batson, Daniel A Spielman, Nikhil Srivastava, and Shang-Hua Teng. Spectral sparsification of graphs: theory and algorithms. *Communications of the ACM*, 56(8):87–94, 2013.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

Taoran Fang, Yunchao Zhang, Yang Yang, Chunping Wang, and Lei Chen. Universal prompt tuning for graph neural networks. *Advances in Neural Information Processing Systems*, 36:52464–52489, 2023.

Qizhang Feng, Zhimeng Stephen Jiang, Ruiquan Li, Yicheng Wang, Na Zou, Jiang Bian, and Xia Hu. Fair graph distillation. *Advances in Neural Information Processing Systems*, 36:80644–80660, 2023.

Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, et al. A survey of graph neural networks for recommender systems: Challenges, methods, and directions. *ACM Transactions on Recommender Systems*, 1(1):1–51, 2023.

Xinyi Gao, Tong Chen, Yilong Zang, Wentao Zhang, Quoc Viet Hung Nguyen, Kai Zheng, and Hongzhi Yin. Graph condensation for inductive node representation learning. In *2024 IEEE 40th international conference on data engineering (ICDE)*, pp. 3056–3069. IEEE, 2024a.

Xinyi Gao, Tong Chen, Wentao Zhang, Yayong Li, Xiangguo Sun, and Hongzhi Yin. Graph condensation for open-world graph learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 851–862, 2024b.

Xinyi Gao, Guanhua Ye, Tong Chen, Wentao Zhang, Junliang Yu, and Hongzhi Yin. Rethinking and accelerating graph condensation: A training-free approach with class partition. In *Proceedings of the ACM on Web Conference 2025*, pp. 4359–4373, 2025a.

Xinyi Gao, Hongzhi Yin, Tong Chen, Guanhua Ye, Wentao Zhang, and Bin Cui. Robgc: Towards robust graph condensation. *IEEE Transactions on Knowledge and Data Engineering*, 37(8):4791–4804, 2025b.

Xinyi Gao, Junliang Yu, Tong Chen, Guanhua Ye, Wentao Zhang, and Hongzhi Yin. Graph condensation: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 37(4):1819–1837, 2025c.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. Pmlr, 2017.

Mohammad Hashemi, Shengbo Gong, Juntong Ni, Wenqi Fan, B Aditya Prakash, and Wei Jin. A comprehensive survey on graph reduction: Sparsification, coarsening, and condensation. *arXiv preprint arXiv:2402.03358*, 2024.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Shixun Huang, Wenqing Lin, Zhifeng Bao, and Jiachen Sun. Influence maximization in real-world closed social networks. *arXiv preprint arXiv:2209.10286*, 2022.

Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. Scaling up graph neural networks via graph coarsening. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pp. 675–684, 2021.

Zhimeng Jiang, Xiaotian Han, Chao Fan, Zirui Liu, Na Zou, Ali Mostafavi, and Xia Hu. Chasing fairness in graphs: A gnn architecture perspective. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 21214–21222, 2024.

Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. Graph condensation for graph neural networks. *arXiv preprint arXiv:2110.07580*, 2021.

Wei Jin, Xianfeng Tang, Haoming Jiang, Zheng Li, Danqing Zhang, Jiliang Tang, and Bing Yin. Condensing graphs via one-step gradient matching. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 720–730, 2022.

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

Mengyang Liu, Shanchuan Li, Xinshi Chen, and Le Song. Graph condensation via receptive field distribution matching. *arXiv preprint arXiv:2206.13697*, 2022.

Yang Liu, Deyu Bo, and Chuan Shi. Graph condensation via eigenbasis matching. *arXiv preprint arXiv:2310.09202*, 2023a.

Yixin Liu, Yizhen Zheng, Daokun Zhang, Vincent CS Lee, and Shirui Pan. Beyond smoothing: Unsupervised graph representation learning with edge heterophily discriminating. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 4516–4524, 2023b.

Zhanyu Liu, Chaolv Zeng, and Guanjie Zheng. Graph data condensation via self-expressive graph structure reconstruction. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1992–2002, 2024.

Andreas Loukas. Graph reduction with spectral and cut guarantees. *Journal of Machine Learning Research*, 20(116):1–42, 2019.

Patrick Reiser, Marlen Neubert, André Eberhard, Luca Torresi, Chen Zhou, Chen Shao, Houssam Metni, Clint van Hoesel, Henrik Schopmans, Timo Sommer, et al. Graph neural networks for materials science and chemistry. *Communications Materials*, 3(1):93, 2022.

Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 563–568, 2008.

Lin Wang, Wenqi Fan, Jiatong Li, Yao Ma, and Qing Li. Fast graph condensation with structure-based neural tangent kernel. In *Proceedings of the ACM Web Conference 2024*, pp. 4439–4448, 2024.

Yansong Wang, Xiaomeng Wang, Yijun Ran, Radosław Michalski, and Tao Jia. Casseqgcn: Combining network structure and temporal sequence to predict information cascades. *Expert Systems with Applications*, 206:117693, 2022.

Ryan Wickman, Xiaofei Zhang, and Weizi Li. A generic graph sparsification framework using deep reinforcement learning. In *2022 IEEE International Conference on Data Mining (ICDM)*, pp. 1221–1226. IEEE, 2022.

Lirong Wu, Haitao Lin, Zhangyang Gao, Guojiang Zhao, and Stan Z Li. A teacher-free graph knowledge distillation framework with dual self-distillation. *IEEE Transactions on Knowledge and Data Engineering*, 36(9):4375–4385, 2024.

Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. *ACM computing surveys*, 55(5):1–37, 2022.

Zhenbang Xiao, Yu Wang, Shunyu Liu, Huiqiong Wang, Mingli Song, and Tongya Zheng. Simple graph condensation. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 53–71. Springer, 2024.

Hongjia Xu, Liangliang Zhang, Yao Ma, Sheng Zhou, Zhuonan Zheng, and Bu Jiajun. A survey on graph condensation. *arXiv preprint arXiv:2402.02000*, 2024.

Sijia Xu, Fan Li, Xiaoyang Wang, Zhengyi Yang, and Xuemin Lin. C^2TC: A training-free framework for efficient tabular data condensation. *arXiv preprint arXiv:2602.21717*, 2026.

Zhe Xu, Yuzhong Chen, Menghai Pan, Huiyuan Chen, Mahashweta Das, Hao Yang, and Hanghang Tong. Kernel ridge regression-based graph dataset distillation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2850–2861, 2023.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

He Zhang, Bang Wu, Xingliang Yuan, Shirui Pan, Hanghang Tong, and Jian Pei. Trustworthy graph neural networks: Aspects, methods, and trends. *Proceedings of the IEEE*, 112(2):97–139, 2024a.

Xikun Zhang, Dongjin Song, Yixin Chen, and Dacheng Tao. Topology-aware embedding memory for continual learning on expanding networks. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4326–4337, 2024b.

Yuchen Zhang, Tianle Zhang, Kai Wang, Ziyao Guo, Yuxuan Liang, Xavier Bresson, Wei Jin, and Yang You. Navigating complexity: Toward lossless graph condensation via expanding window matching. *arXiv preprint arXiv:2402.05011*, 2024c.

Xin Zheng, Miao Zhang, Chunyang Chen, Quoc Viet Hung Nguyen, Xingquan Zhu, and Shirui Pan. Structure-free graph condensation: From large-scale graphs to condensed graph-free data. *Advances in Neural Information Processing Systems*, 36:6026–6047, 2023.

Ziang Zhou, Jieming Shi, Renchi Yang, Yuanhang Zou, and Qing Li. Slotgat: slot-based message passing for heterogeneous graphs. In *International Conference on Machine Learning*, pp. 42644–42657. PMLR, 2023.

A Additional Method Details

This section provides implementation details shared by AUA and LUA. Both methods use the same task-aware representation, deterministic refinement operator, minimum class-coverage rule, and synthetic graph construction. They differ only in how eligible candidate refinements are ranked.

A.1 Task-Aware Representation

Following Eq. 4, we construct the task-aware space from multi-hop propagated observable features. Let

$$\widehat{A}_{\mathrm{obs}} = \widehat{D}_{\mathrm{obs}}^{-1/2}(A_{\mathrm{obs}} + I)\widehat{D}_{\mathrm{obs}}^{-1/2}. \tag{22}$$

To bound the feature dimension before propagation, we apply a fixed Gaussian random projection when the original feature dimension exceeds d_p. Specifically,

$$H^{(0)} = X_{\mathrm{obs}}R, \qquad R_{ij} \sim \mathcal{N}\left(0, \frac{1}{d_p}\right), \tag{23}$$

where the projection is omitted when the original feature dimension is no larger than d_p. We use $d_p = 64$.

Multi-hop views are obtained recursively as

$$H^{(h)} = \widehat{A}_{\mathrm{obs}}H^{(h-1)}, \qquad h = 1, \ldots, K. \tag{24}$$

Each hop block is standardized feature-wise using statistics estimated from the labeled training nodes only. Let $\overline{H}^{(h)}$ denote the standardized block. We first concatenate the standardized multi-hop feature blocks as

$$h_i = [\overline{H}_i^{(0)}; \overline{H}_i^{(1)}; \ldots; \overline{H}_i^{(K)}]. \tag{25}$$

We use $K = 2$ by default.

Graph propagation uses all features observable under the corresponding evaluation protocol, whereas the normalization statistics above depend only on labeled training nodes.

Cross-fitted predictive posteriors. Predictive information is constructed exclusively from V_{tr}. Let $\{\mathcal{I}_f\}_{f=1}^{F}$ denote a stratified partition of the labeled training nodes. For each fold, a ridge predictor is fitted without the held-out fold.

We augment the propagated representation with a bias coordinate,

$$\widetilde{h}_i = [h_i^\top, 1]^\top. \tag{26}$$

Let $\widetilde{H}_{-f}$ and Y_{-f} denote the design matrix and one-hot target matrix of all fitting folds. The ridge solution is

$$W_f = \left(\widetilde{H}_{-f}^\top \widetilde{H}_{-f} + \lambda_{\mathrm{oof}} I\right)^{-1} \widetilde{H}_{-f}^\top Y_{-f}, \tag{27}$$

and, for $i \in \mathcal{I}_f$,

$$s_i = \widetilde{h}_i^\top W_f. \tag{28}$$

The out-of-fold score is converted into a predictive posterior by temperature-scaled softmax:

$$p_i[c] = \frac{\exp(s_i[c]/T_{\mathrm{oof}})}{\sum_{c'=1}^{C} \exp(s_i[c']/T_{\mathrm{oof}})}. \tag{29}$$

The task-aware representation used for region splitting and representation dispersion is the propagated representation

$$z_i = h_i. \tag{30}$$

Thus, z_i depends only on observable node features and graph structure. The ridge construction above separately provides the predictive posteriors p_i used by the semantic and task-difficulty components.

We use $F = 5$ folds, $\lambda_{\text{oof}} = 10^{-2}$, and $T_{\text{oof}} = 1$. Hence, each posterior in this full-training construction is predicted by a model that excludes the corresponding node from fitting. No validation or test labels are used to construct z_i or these predictive estimates.

The construction above defines the full-training out-of-fold posteriors used by AUA and by LUA during final allocation. During LUA supervision, predictive posteriors are instead recomputed by cross-fitting within the corresponding fitting subset, while the calibration subset is used to evaluate candidate-level marginal gains.

A.2 Deterministic Region Splitter

Using the region centroid μ_P defined in Eq. 7, the initial centers are selected by deterministic farthest-point seeding:

$$i_1 = \arg\max_{i \in P} \|z_i - \mu_P\|_2^2, \qquad i_2 = \arg\max_{i \in P} \|z_i - z_{i_1}\|_2^2. \tag{31}$$

Given centers ν_1 and ν_2, Lloyd assignment uses

$$a_i = \arg\min_{k \in \{1,2\}} \|z_i - \nu_k\|_2^2. \tag{32}$$

We perform at most five Lloyd iterations and terminate early when assignments remain unchanged. The implementation supports minimum-child-size and balance constraints. For the reported configuration, the balance constraint is inactive and only non-empty children are required. If Lloyd assignment degenerates to a single non-empty child, we deterministically separate the farthest point from the remaining nodes, ensuring a valid non-empty binary split whenever $|P| > 1$. Consequently, under the reported configuration, only singleton regions are unsplittable.

Because all allocation policies reuse exactly the same candidate split for a given region, controlled comparisons modify *which* region receives the next unit of capacity rather than *how* that region is partitioned.

A.3 Minimum Coverage and Progressive Allocation

Starting from the initial state in Eq. 1, we use a class-independent minimum coverage parameter $b_{\min} \geq 1$ to ensure basic representation for every non-empty class before unrestricted cross-class competition. We choose $b_{\min}$ such that the coverage requirement is feasible under the target budget and splitter constraints; all reported budgets satisfy these feasibility conditions.

Let

$$b_c(\mathcal{S}) = |\{P \in \mathcal{S} : P \subseteq V_c\}| \tag{33}$$

denote the current budget of class c. The coverage-aware eligible set is

$$\mathcal{A}_{\text{cov}}(\mathcal{S}) = \begin{cases} \{P \in \mathcal{A}(\mathcal{S}) : b_{c(P)}(\mathcal{S}) < b_{\min}\}, & \exists c : \, b_c(\mathcal{S}) < b_{\min}, \\ \mathcal{A}(\mathcal{S}), & \text{otherwise}, \end{cases} \tag{34}$$

where $c(P)$ denotes the class containing region P.

When $b_{\min} = 1$, the initial state already satisfies the coverage requirement and all splittable regions compete globally from the first refinement. For $b_{\min} > 1$, refinements are first restricted to under-covered classes; once every class reaches the prescribed minimum, the remaining capacity is allocated through global region-level competition.

At each state, the selected region is refined according to the state update in Eq. 2. Each refinement increases the number of terminal regions by one. After $T = B - C_+$ refinements, the terminal state contains exactly $|\mathcal{S}_T| = B$ regions. The resulting class-wise budgets are those defined in Eq. 3. Beyond the minimum coverage requirement, these class-wise budgets emerge from sequential region-level refinement decisions rather than being prescribed from label frequency.

Algorithm 1 summarizes the final allocation rollout after the LUA reranker has been trained. Accordingly, the posteriors p_i used below are the full-training out-of-fold estimates defined in Eq. 29.

Algorithm 1 Progressive Marginal Allocation

Require: $G_{\text{obs}} = (A_{\text{obs}}, X_{\text{obs}})$, V_{tr}, target budget B, minimum coverage $b_{\min}$, policy $\Pi \in \{\text{AUA}, \text{LUA}\}$, and trained reranker g_θ when $\Pi = \text{LUA}$
Ensure: Terminal state $\mathcal{S}_T$

```
Construct task representations {z_i} and cross-fitted posteriors {p_i}
S_0 ← {V_c : |V_c| > 0}
for t = 0, ..., B − C_+ − 1 do
    Construct feasible candidate set A(S_t)
    Construct coverage-aware eligible set A_cov(S_t)
    if Π = AUA then
        P_t ← arg max_{P ∈ A_cov(S_t)} u^AUA(P)
    else
        M_t ← up to m regions in A_cov(S_t) with the largest u^AUA(P)
        P_t ← arg max_{P ∈ M_t} u_θ^LUA(P)
    end if
    S_{t+1} ← U(S_t, P_t)
end for
return S_T
```

Table 6: Default implementation configuration. Budget-dependent settings follow the rules described in the corresponding sections.

Component	Configuration	Setting
Shared	Propagation depth K	2
	Projection dimension d_p	64
	OOF predictor	5 folds, $\lambda_{\text{oof}} = 10^{-2}$, $T_{\text{oof}} = 1$
	Semantic weight λ_{sem}	1.0
	Minimum child size	1
Allocation	Minimum class coverage $b_{\min}$	$\min\{3, \lfloor B/C_+ \rfloor\}$
	LUA proposal size m	8
Topology	Relay temperature τ_{rel}	0.2
	Relay semantic weight β_{rel}	1.0
LUA	Fit/calibration split	90/10, stratified
	Probe regularization λ_{ridge}	10^{-2}
	Reranker	$5 \to 16 \to 8 \to 1$ MLP, GELU
	Preference temperature τ	1.0
	Auxiliary weight η	0.05
	Reranker optimization	AdamW, lr 10^{-3}, wd 10^{-4}

Because every update refines an existing terminal region,

$$\mathcal{S}_0 \prec \mathcal{S}_1 \prec \cdots \prec \mathcal{S}_T, \tag{35}$$

so a single rollout yields a nested sequence of progressively finer allocations.

A.4 Implementation Configuration

Table 6 summarizes the default configuration used throughout the reported experiments. Shared components use a two-hop task-aware representation with a 64-dimensional projected feature space and five-fold out-of-fold predictive estimates. The minimum class-coverage floor is determined by the available synthetic budget as $b_{\min} = \min\{3, \lfloor B/C_+ \rfloor\}$.

For LUA, AUA retains up to eight analytically promising candidates at each refinement state, which are subsequently reranked using task-grounded supervision. We use a stratified $90/10$ fitting–calibration split and a lightweight $5 \to 16 \to 8 \to 1$ MLP as the reranker. Additional details on supervision construction and optimization are provided in Sec. C.

B Theoretical Analysis

This section establishes properties of the analytic refinement potential used by AUA and clarifies which properties are preserved when LUA reranks feasible AUA proposals.

B.1 Class-Normalized Residual Potential

Recall the class-normalized residual potential $\psi(P)$ defined in Eq. 8. For any partition $\mathcal{P}_c$ of V_c, the within-region dispersion satisfies

$$\sum_{P\in\mathcal{P}_c} D(P) \le D(V_c) \le \gamma_c. \tag{36}$$

For the predictive component,

$$J(P) = |P|H(\bar{p}_P) - \sum_{i\in P} H(p_i). \tag{37}$$

Since

$$0 \le H(p) \le \log C,$$

we obtain

$$0 \le J(P) \le |P| \log C. \tag{38}$$

Consequently,

$$0 \le \sum_{P\in\mathcal{P}_c} \psi(P) \le 1 + \lambda_{\text{sem}}. \tag{39}$$

Thus, the residual associated with each class is measured on a class-relative scale rather than being proportional to raw class size. This normalization permits regions from different classes to compete under a single global refinement budget.

B.2 Exact Refinement Decomposition

Proof of Proposition 1. Consider a feasible refinement

$$P \to \{P^{(1)}, P^{(2)}\}.$$

Let

$$n_a = |P^{(a)}|, \qquad \mu_a = \mu_{P^{(a)}}, \qquad \rho_a = \frac{n_a}{|P|},$$

for $a \in \{1, 2\}$.

Because

$$\mu_P = \rho_1\mu_1 + \rho_2\mu_2, \tag{40}$$

the standard within–between decomposition gives

$$D(P) - D(P^{(1)}) - D(P^{(2)}) = \frac{n_1 n_2}{|P|}\|\mu_1 - \mu_2\|_2^2. \tag{41}$$

For the predictive component,

$$\bar{p}_P = \rho_1\bar{p}_{P^{(1)}} + \rho_2\bar{p}_{P^{(2)}}.$$

Therefore,

$$J(P) - J(P^{(1)}) - J(P^{(2)}) = |P|\,\mathrm{JS}_{\boldsymbol{\rho}}\left(\bar{p}_{P^{(1)}}, \bar{p}_{P^{(2)}}\right), \tag{42}$$

where

$$\boldsymbol{\rho} = (\rho_1, \rho_2).$$

After class normalization,

$$\Delta_{\text{rep}}(P) = \frac{n_1 n_2}{|P|\gamma_c}\|\mu_1 - \mu_2\|_2^2, \tag{43}$$

and

$$\Delta_{\mathrm{sem}}(P) = \frac{|P|}{n_c \log C} \, \mathrm{JS}_{\boldsymbol{\rho}} \left(\bar{p}_{P^{(1)}}, \bar{p}_{P^{(2)}} \right). \tag{44}$$

Both components are non-negative, and hence

$$u^{\mathrm{AUA}}(P) = \Delta_{\mathrm{rep}}(P) + \lambda_{\mathrm{sem}} \Delta_{\mathrm{sem}}(P) \geq 0. \tag{45}$$

AUA therefore measures structure that is actually exposed by the next feasible refinement, rather than parent-region heterogeneity alone.

B.3 Monotone Residual Descent

Define the global residual potential

$$\Phi(\mathcal{S}) = \sum_{P \in \mathcal{S}} \psi(P). \tag{46}$$

A refinement modifies only the selected region. Therefore,

$$\Phi(\mathcal{S}_t) - \Phi\left(\mathcal{U}(\mathcal{S}_t, P)\right) = u^{\mathrm{AUA}}(P) \geq 0. \tag{47}$$

Proposition 2 (Monotone residual descent) *If $\mathcal{S}_{t+1}$ is produced by either AUA or LUA, then*

$$\Phi(\mathcal{S}_{t+1}) \leq \Phi(\mathcal{S}_t). \tag{48}$$

Moreover, AUA achieves the largest one-step reduction in Φ among the currently eligible refinements, while LUA preserves monotone descent by reranking a subset of these eligible candidates.

Proof. For any feasible refinement P, Eq. 47 guarantees $\Phi(\mathcal{U}(\mathcal{S}_t, P)) \leq \Phi(\mathcal{S}_t)$. AUA selects

$$P_t^{\mathrm{AUA}} = \arg \max_{P \in \mathcal{A}_{\mathrm{cov}}(\mathcal{S}_t)} u^{\mathrm{AUA}}(P),$$

and therefore maximizes the one-step analytic reduction over the current eligible set.

For LUA,

$$P_t^{\mathrm{LUA}} \in \mathcal{M}_t \subseteq \mathcal{A}_{\mathrm{cov}}(\mathcal{S}_t) \subseteq \mathcal{A}(\mathcal{S}_t).$$

Hence,

$$u^{\mathrm{AUA}}\big(P_t^{\mathrm{LUA}}\big) \geq 0,$$

and substituting the selected candidate into Eq. 47 yields $\Phi(\mathcal{S}_{t+1}) \leq \Phi(\mathcal{S}_t)$. □

B.4 Class-Wise Replication Invariance

Proposition 3 (Class-wise replication invariance) *Suppose every node in a class is uniformly replicated by a factor $m_{\mathrm{rep}} \in \mathbb{N}$, with replicas preserving the same representations, predictive posteriors, and refinement structure. If the floor in γ_c is inactive, then every corresponding candidate satisfies*

$$u^{\mathrm{AUA}}_{m_{\mathrm{rep}}}(P) = u^{\mathrm{AUA}}(P). \tag{49}$$

Proof. Uniform replication leaves centroids, average predictive distributions, and child proportions unchanged. Meanwhile,

$$D(P), \ \gamma_c, \ |P|, \quad \text{and} \quad n_c$$

all scale by m_{rep}. Consequently,

$$\Delta_{\mathrm{rep}}^{(m_{\mathrm{rep}})}(P) = \Delta_{\mathrm{rep}}(P), \qquad \Delta_{\mathrm{sem}}^{(m_{\mathrm{rep}})}(P) = \Delta_{\mathrm{sem}}(P). \tag{50}$$

The claim follows immediately. □

This property formalizes the distinction between class frequency and refinement demand: duplicating observations without introducing new representation or predictive structure does not increase analytic utility.

B.5 Scope of the Analysis

The results above concern the explicit analytic potential Φ and its behavior under feasible refinement. They do not establish global optimality with respect to the downstream condensation objective $\mathcal{Q}$.

AUA should therefore be interpreted as a principled greedy descent rule for the explicit residual potential Φ. LUA retains the same feasible refinement space but uses held-out task evidence to modify the local ordering of analytically promising refinements.

C Learning-Based Utility Allocation Details

This section describes the construction of task-grounded marginal supervision, the candidate representation used by LUA, and the distinction between supervision collection and final deployment.

C.1 Probe-Based Marginal Supervision

LUA obtains task-grounded supervision entirely from the labeled training set. For each supervision split, we form a stratified partition

$$V_{\mathrm{tr}} = \mathcal{V}_{\mathrm{fit}} \cup \mathcal{V}_{\mathrm{cal}}, \qquad \mathcal{V}_{\mathrm{fit}} \cap \mathcal{V}_{\mathrm{cal}} = \emptyset. \tag{51}$$

The label-free task representations $\{z_i\}$ are constructed following Appendix A.1. For each supervision split, the predictive quantities used by fitting-side candidate states are cross-fitted within $\mathcal{V}_{\mathrm{fit}}$, while $\mathcal{V}_{\mathrm{cal}}$ is used to evaluate probe-level marginal gains.

During LUA supervision, the class normalizers are recomputed within the corresponding fitting subset. Specifically, letting

$$V_c^{\mathrm{fit}} = V_c \cap \mathcal{V}_{\mathrm{fit}}, \qquad n_c^{\mathrm{fit}} = |V_c^{\mathrm{fit}}|, \qquad \gamma_c^{\mathrm{fit}} = \max\{D(V_c^{\mathrm{fit}}), \varepsilon\}, \tag{52}$$

we use n_c^{fit} and γ_c^{fit} in place of n_c and γ_c when evaluating analytic proposal scores and LUA descriptors for that supervision split.

The supervision refinement forest contains only fitting nodes, so every active region satisfies

$$P \subseteq \mathcal{V}_{\mathrm{fit}}. \tag{53}$$

Accordingly, supervision trajectories are collected only over states feasible within the fitting subset. We define the maximum fitting-side supervision budget as

$$B_{\mathrm{sup}} = \min\left\{B,\, |\mathcal{V}_{\mathrm{fit}}|\right\}. \tag{54}$$

Thus, supervision collection need not reach the final target budget B when $B > |\mathcal{V}_{\mathrm{fit}}|$. This restriction applies only to reranker supervision; final allocation is performed on the full V_{tr} and reaches the prescribed budget B.

Within this forest, posterior-dependent candidate quantities, including semantic refinement gain, remaining semantic complexity, and regional predictive difficulty, are evaluated using the fitting-side out-of-fold predictive estimates. The propagated task representations are shared across supervision splits.

For a class-pure fitting region $P \subseteq V_c^{\mathrm{fit}}$, its probe representative is

$$\bar{z}_P^{\mathrm{fit}} = \frac{1}{|P|} \sum_{i \in P} z_i, \qquad y_P = c. \tag{55}$$

Under the empirical class prior, the corresponding probe weight is

$$w_P^{\mathrm{probe}} = \frac{|P|}{|\mathcal{V}_{\mathrm{fit}}|}. \tag{56}$$

A lightweight ridge probe is fitted from the active region representatives:

$$\beta_{\mathcal{S}} = \arg\min_{\beta} \sum_{P \in \mathcal{S}} w_P^{\mathrm{probe}} \left\| h_\beta(\bar{z}_P^{\mathrm{fit}}) - e_{y_P} \right\|_2^2 + \lambda_{\mathrm{ridge}} \|\beta\|_F^2. \tag{57}$$

We use $\lambda_{\text{ridge}} = 10^{-2}$.

Its held-out loss is

$$\mathcal{L}_{\text{cal}}(\mathcal{S}) = \frac{1}{|\mathcal{V}_{\text{cal}}|} \sum_{i \in \mathcal{V}_{\text{cal}}} \|h_{\beta_{\mathcal{S}}}(z_i) - e_{y_i}\|_2^2 . \tag{58}$$

The candidate-level marginal gain is then computed according to Eq. 15. Thus, $r_t(P) > 0$ indicates that assigning the next representative to P reduces held-out probe loss. The calibration subset provides the held-out labels used to evaluate the probe-level supervision targets, while proposal scores and reranker descriptors use the fitting-side out-of-fold predictive quantities defined above.

When the calibration subset is small, we stabilize the supervision target by bootstrap resampling and use

$$r_t^{\text{rob}}(P) = \overline{r}_t(P) - \beta_{\text{unc}} \, \text{SE}[r_t(P)]. \tag{59}$$

The default configuration uses eight bootstrap samples and $\beta_{\text{unc}} = 1$. This stabilization is used only during supervision construction; probe marginals are not evaluated during final allocation.

C.2 Candidate Descriptor

LUA describes each candidate refinement through three complementary factors: the structure resolved by the split, the complexity remaining after refinement, and the task difficulty of the parent region. For notational simplicity, the expressions below retain the posterior notation p_i introduced in the main method. During LUA supervision, p_i denotes the out-of-fold posterior constructed within the corresponding fitting subset. During final allocation, p_i denotes the full-training out-of-fold posterior constructed in Appendix A.1. During supervision collection, all active candidate regions are restricted to $\mathcal{V}_{\text{fit}}$.

Refinement gain. The first factor measures the structure exposed by the current refinement. We retain the representation and semantic reductions defined by AUA and group them as

$$\mathcal{G}(P) = (\Delta_{\text{rep}}(P), \Delta_{\text{sem}}(P)) , \tag{60}$$

where $\Delta_{\text{rep}}(P)$ and $\Delta_{\text{sem}}(P)$ are given in Eq. 10. Thus, $\mathcal{G}(P)$ characterizes how much geometric and predictive structure is resolved by replacing the parent region with its two children.

Remaining complexity. The second factor measures the structure that remains unresolved after the same refinement. For the representation component, we define

$$R_{\text{rep}}(P) = \frac{D(P^{(1)}) + D(P^{(2)})}{\gamma_c}. \tag{61}$$

For the semantic component, let

$$S(P) = |P| H(\bar{p}_P) \tag{62}$$

denote the predictive entropy mass of region P. Using

$$J(P) = S(P) - \sum_{i \in P} H(p_i), \tag{63}$$

the node-wise entropy terms cancel under partition refinement, giving

$$\Delta_{\text{sem}}(P) = \frac{S(P) - S(P^{(1)}) - S(P^{(2)})}{n_c \log C}. \tag{64}$$

We therefore define the remaining semantic complexity as

$$R_{\text{sem}}(P) = \frac{S(P^{(1)}) + S(P^{(2)})}{n_c \log C}. \tag{65}$$

The two residual components are grouped as

$$\mathcal{R}(P) = (R_{\text{rep}}(P), R_{\text{sem}}(P)) . \tag{66}$$

Hence, $\mathcal{R}(P)$ summarizes how much representation and predictive complexity remains after the candidate refinement.

Task difficulty. The third factor is the regional predictive difficulty e_P defined in Eq. 17. Together, $\mathcal{G}(P)$ and $\mathcal{R}(P)$ instantiate the structured descriptor $\varphi(P)$ defined in Eq. 16. The representation and semantic components within $\mathcal{G}(P)$ and $\mathcal{R}(P)$ are retained separately when evaluating g_θ.

C.3 Preference Learning

LUA learns to reproduce the relative task-grounded preference among candidate refinements within each state. At supervision state $\mathcal{S}_t$, AUA first provides the analytic proposal set $\mathcal{M}_t$ defined in Eq. 13. To improve local coverage during supervision collection, we augment this set with a small number of randomly sampled eligible refinements,

$$\widetilde{\mathcal{M}}_t = \mathcal{M}_t \cup \mathcal{R}_t^{\mathrm{rand}}, \tag{67}$$

where $\mathcal{R}_t^{\mathrm{rand}} \subseteq \mathcal{A}_{\mathrm{cov}}(\mathcal{S}_t) \setminus \mathcal{M}_t$. We use two additional candidates by default. These candidates are used only for supervision collection; deployment uses $\mathcal{M}_t$ alone.

For each candidate, let $r_t^\star(P)$ denote the task-grounded supervision target,

$$r_t^\star(P) = \begin{cases} r_t^{\mathrm{rob}}(P), & \text{when bootstrap stabilization is used,} \\ r_t(P), & \text{otherwise.} \end{cases} \tag{68}$$

Because the scale of probe marginals can vary across refinement states, we standardize them within each state:

$$\widetilde{r}_t(P) = \frac{r_t^\star(P) - \bar{r}_t^\star}{\max\{\sigma_t^\star, \varepsilon_{\mathrm{std}}\}}, \tag{69}$$

where $\bar{r}_t^\star$ and $\sigma_t^\star$ are the mean and standard deviation of $\{r_t^\star(P) : P \in \widetilde{\mathcal{M}}_t\}$, and $\varepsilon_{\mathrm{std}} > 0$ is a numerical stability constant.

The standardized marginals define a target preference distribution,

$$q_t(P) = \frac{\exp(\widetilde{r}_t(P)/\tau)}{\sum_{P' \in \widetilde{\mathcal{M}}_t} \exp(\widetilde{r}_t(P')/\tau)}, \qquad P \in \widetilde{\mathcal{M}}_t, \tag{70}$$

while the reranker induces

$$\widehat{q}_t(P) = \frac{\exp\big(u_\theta^{\mathrm{LUA}}(P)/\tau\big)}{\sum_{P' \in \widetilde{\mathcal{M}}_t} \exp\big(u_\theta^{\mathrm{LUA}}(P')/\tau\big)}. \tag{71}$$

We train the reranker with a preference-matching objective together with a small auxiliary regression term,

$$\begin{aligned} \mathcal{L}_{\mathrm{LUA}} = &- \sum_t \sum_{P \in \widetilde{\mathcal{M}}_t} q_t(P) \log \widehat{q}_t(P) \\ &+ \eta \sum_t \sum_{P \in \widetilde{\mathcal{M}}_t} \mathrm{Huber}\left(u_\theta^{\mathrm{LUA}}(P), \widetilde{r}_t(P)\right). \end{aligned} \tag{72}$$

The first term learns the within-state ordering required for allocation, whereas the auxiliary term stabilizes the scale of the learned scores.

The reranker g_θ is a lightweight MLP with hidden dimensions 16 and 8. We use AdamW with learning rate 10^{-3}, weight decay 10^{-4}, temperature $\tau = 1$, auxiliary weight $\eta = 0.05$, and at most 200 training epochs with early stopping.

C.4 State-Wise Affine Invariance of LUA Targets

Consider a fixed state $\mathcal{S}_t$ and an affine transformation of the effective supervision targets,

$$r_t^{\star\prime}(P) = a_t r_t^\star(P) + b_t, \qquad a_t > 0. \tag{73}$$

When the standard-deviation floor is inactive before and after the transformation,

$$\widetilde{r}_t'(P) = \widetilde{r}_t(P), \qquad q_t'(P) = q_t(P). \tag{74}$$

Indeed,

$$\bar{r}_t^{\star\prime} = a_t \bar{r}_t^{\star} + b_t, \qquad \sigma_t^{\star\prime} = a_t \sigma_t^{\star},$$

so the state-specific positive scale and shift cancel after standardization. LUA therefore learns relative within-state preferences without depending on arbitrary affine changes in the scale of its supervision targets.

C.5 Rollout Construction and Deployment

Reranker supervision is first collected from states visited by an AUA allocation trajectory. After training the initial reranker, we perform one rollout using the learned policy and add the newly visited states to the supervision set. This exposes the reranker to states induced by its own decisions and reduces train–deployment mismatch.

At deployment, task-grounded probe marginals are no longer evaluated. The full labeled training set is used to construct the out-of-fold posteriors in Eq. 29, which provide the posterior-dependent quantities for both AUA proposal scoring and LUA reranking descriptors. AUA then constructs the analytic proposal set $\mathcal{M}_t$ according to Eq. 13, and LUA performs the reranking and state update according to Eq. 18. Thus, final allocation requires only analytic candidate generation and a lightweight forward pass through g_θ, without probe-level marginal evaluation.

D Synthetic Graph Construction

The terminal refinement state determines the resolution of the synthetic graph. After allocation, each terminal region corresponds to one synthetic node. Representative construction, training weights, and topology recovery follow the same fixed post-processing pipeline for all allocation policies. Thus, AUA and LUA differ only in the terminal partition supplied to this common construction.

D.1 Synthetic Representatives

After $T = B - C_+$ refinements, the terminal state contains $|\mathcal{S}_T| = B$ class-pure regions. For each terminal region $P_k \subseteq V_c$, we construct the corresponding synthetic node as the arithmetic mean of its member features,

$$x'_k = \frac{1}{|P_k|} \sum_{i \in P_k} x_i, \qquad y'_k = c. \tag{75}$$

The same deterministic representative construction is used for all allocation policies.

D.2 Mass-Preserving Training Weights

Adaptive allocation changes the number of representatives assigned to each class. To decouple representation resolution from optimization importance, we preserve the total training mass associated with each class. For a representative corresponding to $P_k \subseteq V_c$, we assign

$$w_k = \pi_c \frac{|P_k|}{n_c}, \qquad \sum_{k: P_k \subseteq V_c} w_k = \pi_c, \tag{76}$$

where π_c is independent of the emergent class budget $b_c(B)$. Using the empirical class prior,

$$\pi_c = \frac{n_c}{|V_{\mathrm{tr}}|}, \tag{77}$$

gives

$$w_k = \frac{|P_k|}{|V_{\mathrm{tr}}|}. \tag{78}$$

The downstream objective on the synthetic graph is therefore

$$\mathcal{L}_{\mathrm{syn}} = \sum_{k=1}^{B} w_k \, \ell\left(f_\theta(A', X')_k, y'_k\right). \tag{79}$$

Hence, assigning additional representatives to a class increases its representation granularity without mechanically increasing its total optimization mass.

D.3 TOPOLOGY RECOVERY

Topology is recovered after the terminal partition has been determined. Let

$$\mathbf{Q} \in \mathbb{R}_{+}^{N_{\mathrm{obs}} \times B}, \qquad \sum_{k=1}^{B} Q_{ik} = 1, \tag{80}$$

denote the node-to-representative routing matrix.

Training nodes inherit their terminal-region assignments exactly. If $i \in P_k$, then

$$Q_{ik} = 1, \qquad Q_{ij} = 0 \quad (j \neq k). \tag{81}$$

For transductive datasets, observable non-training nodes are routed to terminal representatives using task-space compatibility together with a predictive prior obtained from labeled training nodes. For terminal region P_k, its task-space prototype is

$$\bar{z}_k = \frac{1}{|P_k|} \sum_{j \in P_k} z_j, \qquad z_k^{\mathrm{proto}} = \frac{\bar{z}_k}{\max\{\|\bar{z}_k\|_2, \varepsilon\}}. \tag{82}$$

where $\varepsilon > 0$ is used only for numerical stability.

A lightweight ridge predictor fitted on V_{tr} provides the training-derived class distribution ρ_i. We use the same ridge regularization and temperature as in the predictive construction described in Appendix A.1. For observable node i and representative k, the routing score is

$$s_{ik} = \frac{z_i^{\top} z_k^{\mathrm{proto}}}{\tau_{\mathrm{rel}}} + \beta_{\mathrm{rel}} \log \rho_i[y'_k]. \tag{83}$$

Here, τ_{rel} controls the temperature of task-space compatibility, while β_{rel} controls the contribution of the training-derived predictive prior. We use $\tau_{\mathrm{rel}} = 0.2$ and $\beta_{\mathrm{rel}} = 1.0$ for all datasets.

The routing scores are converted to non-negative compatibility weights by

$$a_{ik} = \frac{\exp(s_{ik})}{\sum_{j=1}^{B} \exp(s_{ij})}. \tag{84}$$

We define the routing confidence of node i as

$$c_i = \max_k a_{ik}. \tag{85}$$

In the reported configuration, the routing support size is selected deterministically as

$$r_i = \begin{cases} 1, & c_i \geq 0.65, \\ 2, & c_i < 0.65. \end{cases} \tag{86}$$

Let $\mathcal{N}_i$ contain the r_i representatives with the largest a_{ik}. The sparse routing weights are

$$Q_{ik} = \begin{cases} \dfrac{a_{ik}}{\sum_{j \in \mathcal{N}_i} a_{ij}}, & k \in \mathcal{N}_i, \\ 0, & k \notin \mathcal{N}_i. \end{cases} \tag{87}$$

Hence, $Q_{ik} \geq 0$ and $\sum_k Q_{ik} = 1$ for every observable node. No non-training labels are used in this routing procedure.

Given $\mathbf{Q}$, source connectivity is projected onto the synthetic representatives as

$$A_{\mathrm{proto}} = \mathbf{Q}^{\top} A_{\mathrm{obs}} \mathbf{Q}. \tag{88}$$

The projected connectivity is sparsified by retaining the top-k_{graph} projected neighbors of each representative, where

$$k_{\mathrm{graph}} = \min\left\{16,\ \max\left(2,\ \left\lfloor \sqrt{B} \right\rfloor\right)\right\}. \tag{89}$$

The resulting sparse matrix is then symmetrized, self-loops are added, and symmetric degree normalization is applied to obtain the final synthetic adjacency A'.

For inductive datasets, topology recovery is restricted to the training subgraph. In this case, $\mathbf{Q}$ contains only the hard terminal-region assignments of training nodes, and the same projection and normalization procedure is applied to the training-subgraph adjacency.

Overall, the allocation policy determines the terminal partition and hence the representation resolution of the synthetic graph, while representative construction, training weights, and topology recovery remain shared across allocation policies.

E Computational Complexity and Runtime Protocol

Let $N = |V_{\mathrm{obs}}|$, $E = |E_{\mathrm{obs}}|$, $N_{\mathrm{tr}} = |V_{\mathrm{tr}}|$, d denote the task-space dimension, K the propagation depth, I the maximum number of Lloyd iterations, and $T = B - C_+$ the number of refinements.

Sparse task-space propagation requires

$$O(KEd) \tag{90}$$

time. For a region P, bounded two-means requires

$$O(I|P|d). \tag{91}$$

Because the active regions partition V_{tr}, constructing all candidate refinements at one state is conservatively bounded by

$$O(IN_{\mathrm{tr}}d). \tag{92}$$

A full AUA trajectory therefore has complexity

$$O\left(KEd + TIN_{\mathrm{tr}}d\right). \tag{93}$$

For LUA, let m denote the maximum number of analytically retained candidates per state and $\mathcal{C}_{\mathrm{probe}}$ the cost of one probe-level candidate evaluation. If N_{state} states are used to construct reranker supervision, the training cost is conservatively

$$O\left(KEd + N_{\mathrm{state}}\left[IN_{\mathrm{tr}}d + m\mathcal{C}_{\mathrm{probe}}\right]\right). \tag{94}$$

During deployment, probe marginals are not recomputed. The additional learned-policy cost is therefore only

$$O(Tm\mathcal{C}_g), \tag{95}$$

where $\mathcal{C}_g$ is the cost of a lightweight reranker forward pass.

Runtime protocol. The condensation time reported in Table 2 measures the operations required to produce the final synthetic graph and excludes downstream student training. The reported condensation time covers task-space preprocessing, progressive refinement, and the complete synthetic-graph construction pipeline. For LUA, the reported time additionally includes task-grounded supervision collection and reranker training. All reported condensation times are averaged over five independent runs.

Computing infrastructure. All experiments were conducted on a Linux cluster node equipped with an Intel(R) Xeon(R) Platinum 8358P CPU at 2.60 GHz, a single NVIDIA A800 GPU with 80 GB of memory, and 520 GB of system memory.

F Experimental Details

F.1 Datasets and Splits

We evaluate on six public node-classification benchmarks: Cora and Citeseer (Kipf & Welling, 2016), ogbn-arxiv and ogbn-products (Hu et al., 2020), and Flickr and Reddit (Zeng et al., 2019). We use the standard public train/validation/test splits.

Cora, Citeseer, ogbn-arxiv, and ogbn-products are evaluated transductively. Graph structure and node features are observable during condensation, while supervision is restricted to training labels.

Flickr and Reddit are evaluated inductively, and condensation is performed using the training subgraph only.

Table 7 summarizes the dataset statistics and evaluation settings.

Table 7: Dataset statistics and evaluation settings.

Dataset	#Nodes	#Edges	#Features	#Classes	Train/Val/Test	Setting
Cora	2,708	5,429	1,433	7	140/500/1,000	Trans.
Citeseer	3,327	4,732	3,703	6	120/500/1,000	Trans.
ogbn-arxiv	169,343	1,166,243	128	40	90,941/29,799/48,603	Trans.
ogbn-products	2,449,029	61,859,140	100	47	196,615/39,323/2,213,091	Trans.
Flickr	89,250	899,756	500	7	44,625/22,312/22,313	Ind.
Reddit	232,965	23,213,838	602	41	153,932/23,699/55,334	Ind.

Table 8: The listed integer budget B is the authoritative synthetic-node budget used for each reported setting.

Dataset	Low	Middle	High
Cora	1.30% / 35	2.60% / 70	5.20% / 140
Citeseer	0.90% / 30	1.80% / 60	3.60% / 120
ogbn-arxiv	0.05% / 90	0.25% / 454	0.50% / 909
Flickr	0.10% / 44	0.50% / 223	1.00% / 446
Reddit	0.05% / 76	0.10% / 153	0.20% / 307
ogbn-products	0.03% / 734	0.05% / 1,224	0.10% / 2,449

F.2 Condensation Ratios and Synthetic Budgets

We follow the nominal condensation-ratio settings used in prior graph condensation studies. Each reported setting is associated with the fixed integer budget B listed in Table 8, which is treated as the authoritative budget in our experiments. All compared methods at the same dataset and ratio use the same synthetic-node budget. For Cora at $B = 140$ and Citeseer at $B = 120$, the synthetic budget equals the number of labeled training nodes.

F.3 Motivation Statistics

Figure 1 examines whether class frequency provides a useful proxy for where additional synthetic capacity is needed. We compare class size with two task-relevant quantities: cross-fitted predictive difficulty and intra-class representation heterogeneity.

Predictive difficulty. For class c, predictive difficulty is measured by the mean out-of-fold cross-entropy

$$d_c = \frac{1}{n_c} \sum_{i \in V_c} -\log p_i[y_i], \tag{96}$$

where p_i is the cross-fitted posterior defined in Eq. 29. Thus, the predictor used to score each node excludes that node from fitting, providing a training-only estimate of class-level predictive difficulty.

Intra-class heterogeneity. For class c, we measure representation heterogeneity by the average class-level distortion in the task-aware space,

$$h_c = \frac{D(V_c)}{n_c}. \tag{97}$$

Dividing by class size removes the direct linear dependence of the sum-of-squares distortion on the number of class samples, yielding an average within-class measure of representation dispersion.

Within-dataset rank normalization. Because class sizes and task-level statistics have substantially different scales across datasets, each quantity is independently converted to a within-dataset percentile rank. For class-level values $\{v_c\}_{c=1}^{C}$, we define

$$\mathrm{Pct}(v_c) = \frac{\mathrm{rank}(v_c) - 1}{C - 1}, \tag{98}$$

where ties receive their average rank. If all classes have identical values, the corresponding percentiles are set to 0.5.

Figure 1 therefore compares

$$\mathrm{Pct}(n_c) \quad \text{vs.} \quad \mathrm{Pct}(d_c) \tag{99}$$

and

$$\mathrm{Pct}(n_c) \quad \text{vs.} \quad \mathrm{Pct}(h_c). \tag{100}$$

This analysis is descriptive and is not used by either allocation policy. Its purpose is to examine whether class frequency provides a useful proxy for predictive difficulty or intra-class representational heterogeneity.

F.4 Baselines and Evaluation Protocol

We compare against ten representative graph reduction and condensation methods: A-ConvMatch (A-CM), GCond, GCDM, DosCond, SGDD, SimGC, EXGC, GCSR, CGC, and SFGC.

For every dataset and condensation ratio, competing methods are matched using the same synthetic-node budget. Unless otherwise specified, condensed graphs are evaluated using a two-layer GCN and results are reported as mean $\pm$ standard deviation over five runs.

The controlled allocation experiment uses the same task-aware representation, deterministic splitter, total synthetic budget, representative construction, training weights, topology-recovery protocol, and downstream evaluation protocol for ratio-driven, uniform, AUA, and LUA. Thus, only the allocation decision rule is changed in the controlled comparison.

For the cross-architecture experiment, the same condensed graph is reused without re-condensation and evaluated using MLP, SGC, GAT, GCN, GraphSAGE, and APPNP.

For the static policies, ratio-driven and uniform allocation first determine the target class-wise budgets. Given these targets, refinement within each class uses the same deterministic splitter as AUA and LUA; when multiple regions of the selected class are splittable, the region with the largest analytic refinement utility is refined. Thus, the controlled comparison changes the cross-class allocation rule while keeping the within-class refinement mechanism fixed.

G Detailed Allocation Analysis

The controlled comparison in Table 5 isolates the effect of allocation policy. Here we further characterize how class-wise budgets emerge from sequential region-level decisions.

G.1 Allocation Deviation

Let

$$\mathbf{b}_t = (b_{t,1}, \ldots, b_{t,C}) \tag{101}$$

be the class-wise budget vector at state $\mathcal{S}_t$, with

$$B_t = \sum_{c=1}^{C} b_{t,c}. \tag{102}$$

Let $\mathbf{b}_t^{\mathrm{ratio}}$ denote the ratio-driven allocation under the same current total budget B_t.

We define normalized deviation from ratio-driven allocation as

$$D_{\mathrm{ratio}}(\mathbf{b}_t) = \frac{\|\mathbf{b}_t - \mathbf{b}_t^{\mathrm{ratio}}\|_1}{2B_t}. \tag{103}$$

The normalization places the quantity on a scale independent of the current budget.

Similarly, the difference between AUA and LUA is

$$D_{\mathrm{A/L}}(\mathcal{S}_t) = \frac{\|\mathbf{b}_t^{\mathrm{AUA}} - \mathbf{b}_t^{\mathrm{LUA}}\|_1}{2B_t}. \tag{104}$$

Table 9: Final normalized allocation deviation (%) in the representative controlled rollout. AUA–Ratio and LUA–Ratio measure departure from ratio-driven allocation; AUA–LUA measures the difference between the two adaptive policies.

Dataset	Budget B	AUA–Ratio	LUA–Ratio	AUA–LUA
Cora	70	10.0	12.9	11.4
Citeseer	60	3.3	21.7	23.3
Flickr	223	40.4	39.0	2.2
Reddit	153	33.3	32.7	8.5
ogbn-arxiv	454	51.1	47.4	20.7
ogbn-products	1224	61.7	61.1	3.6

Table 10: Deviation from ratio-driven allocation (%) at representative stages of the refinement trajectory on the four larger benchmarks.

Dataset	Method	50% Progress	75% Progress	100% Progress
Flickr	AUA	15.7	29.0	40.4
	LUA	13.9	28.4	39.0
Reddit	AUA	14.4	24.0	33.3
	LUA	13.4	25.6	32.7
ogbn-arxiv	AUA	39.3	45.7	51.1
	LUA	31.6	39.1	47.4
ogbn-products	AUA	54.2	58.1	61.7
	LUA	51.5	56.8	61.1

These quantities are descriptive measures of allocation behavior. Larger deviation is not assumed to imply better condensation quality.

The magnitude of reallocation is strongly dataset-dependent. For example, the adaptive allocations remain relatively close to the static reference on Cora, whereas substantially larger departures occur on ogbn-arxiv and ogbn-products. However, deviation itself is not the optimization objective; the controlled accuracy comparison determines whether the resulting allocation improves condensation quality.

G.2 Progressive Allocation Dynamics

Because class-wise budgets emerge sequentially, the final budget vector alone does not reveal when adaptive behavior develops. We therefore track D_{ratio} throughout the refinement trajectory.

Figure 4 shows that adaptive class-wise budgets are not imposed through a post-hoc redistribution. Instead, deviation from the static reference accumulates through successive region-level refinement decisions.

Different datasets exhibit distinct trajectory shapes and magnitudes, consistent with the fact that the allocation policy responds to regional refinement demand rather than enforcing a fixed class-level rebalancing rule.

Table 10 reports representative trajectory checkpoints on the four larger benchmarks.

G.3 Task-Grounded Correction of AUA

We next measure $D_{\mathrm{A/L}}$ throughout refinement to characterize how strongly task-grounded reranking changes the analytic allocation trajectory.

The correction magnitude varies across datasets, showing that LUA does not apply a fixed global redistribution rule to AUA. Instead, its effect depends on the task evidence associated with the current candidate refinements.

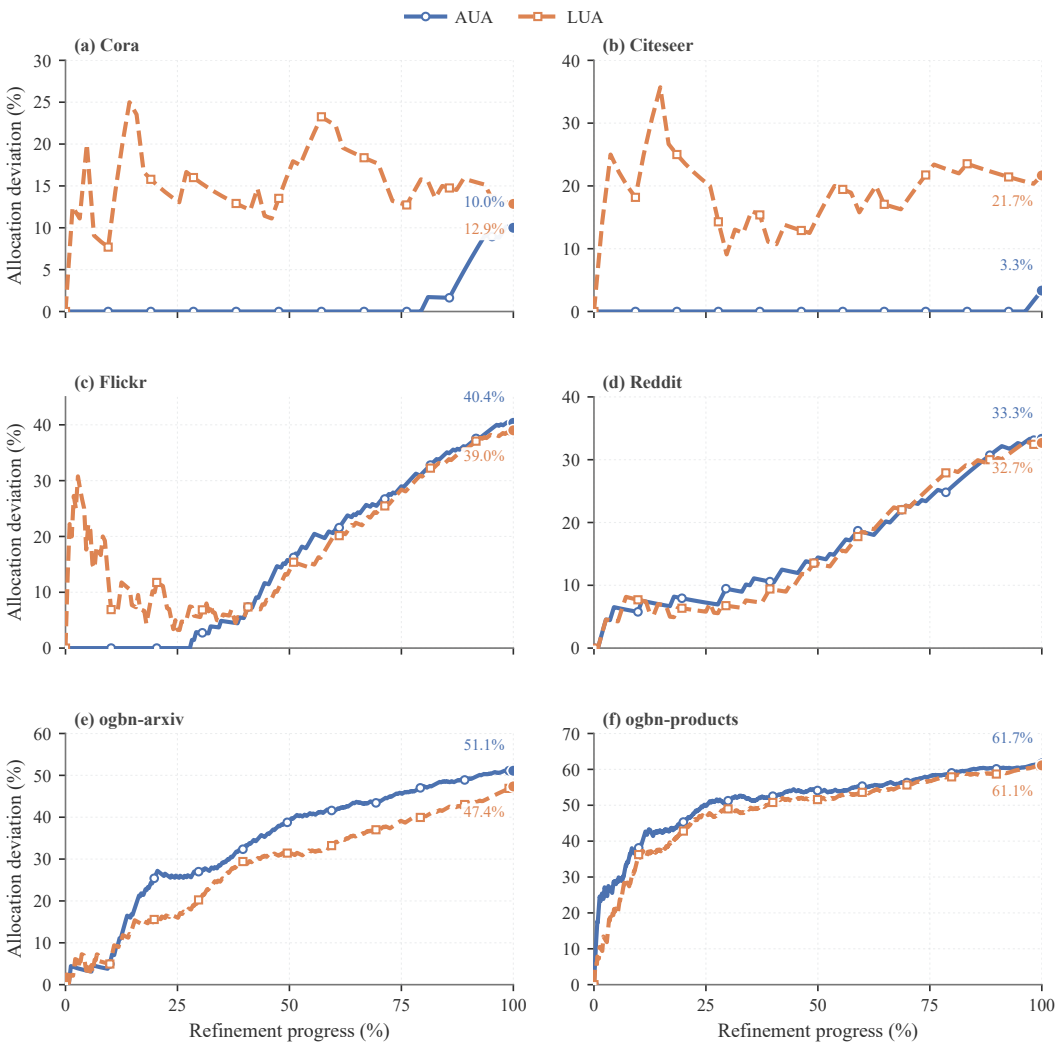


Figure 4: **Progressive allocation dynamics.** Normalized deviation from ratio-driven allocation over refinement progress. At each state, all policies are compared using the same current total synthetic budget.

In particular, a small difference in the terminal class-count vectors does not imply that AUA and LUA followed identical refinement trajectories. Their allocation variable is the *current region receiving the next representative*, not the final class count alone. Two policies can therefore produce similar class-wise totals while allocating capacity to different intra-class regions.

Allocation deviation should also not be interpreted as a predictor of accuracy. Uniform allocation uses the same condensation pipeline but can perform substantially worse than the adaptive policies. Thus, the gains of AUA and LUA cannot be explained simply by producing class budgets that are more balanced or more different from the original class ratios.

G.4 Trajectory Reporting Protocol

The trajectory figures and trajectory-derived statistics above are computed from a fixed allocation run used for the controlled analysis. They are used only to characterize allocation behavior.

Accuracy values in Table 5 remain mean $\pm$ standard deviation over five runs under the matched controlled pipeline.

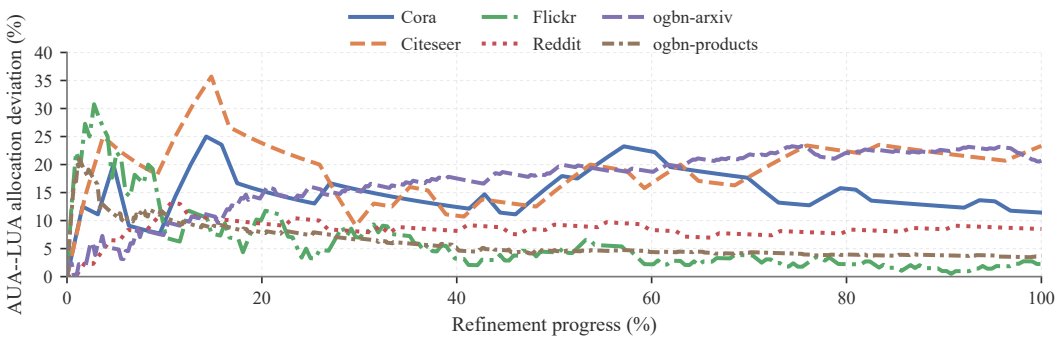


Figure 5: **Task-grounded correction of the analytic allocation trajectory.** Normalized class-budget deviation between AUA and LUA throughout progressive refinement.

G.5 Sensitivity to the Per-Class Budget Floor

We examine the sensitivity of AUA and LUA to the per-class budget floor $b_{\min}$ on Cora, ogbn-arxiv, and ogbn-products. The floor controls the initial level of class coverage before the remaining synthetic capacity is allocated according to refinement utility. For a fixed total budget B, increasing $b_{\min}$ reserves more capacity for initial class coverage and correspondingly leaves less capacity for utility-driven allocation.

As shown in Figure 6, both AUA and LUA remain relatively stable across the tested values of $b_{\min}$ on all three datasets. Increasing the budget floor does not yield a consistent improvement in downstream performance, indicating that the proposed allocation framework does not rely on careful tuning of $b_{\min}$. Overall, once basic class coverage is established, the final allocation remains largely governed by subsequent utility-driven refinement rather than the precise choice of the initial budget floor.

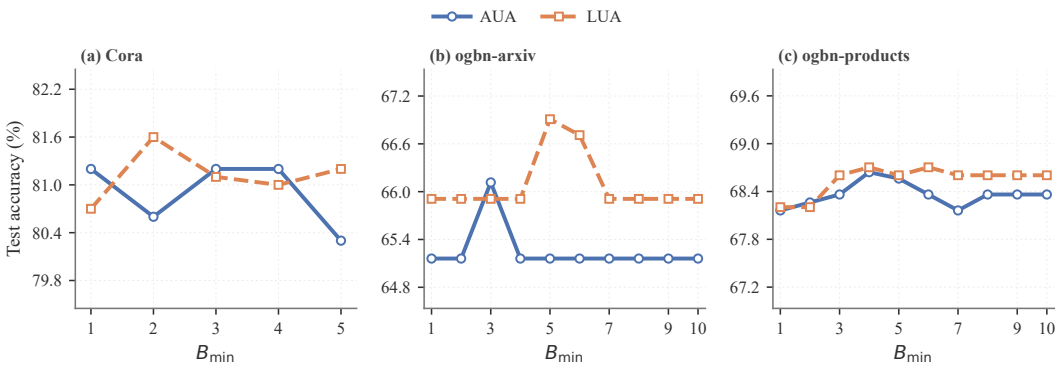


Figure 6: Sensitivity to the initial per-class budget floor $b_{\min}$ on Cora, ogbn-arxiv, and ogbn-products. The total synthetic budget is fixed for each dataset. Both AUA and LUA exhibit stable performance across different settings, showing limited sensitivity to $b_{\min}$.

G.6 Limitations

The benefit of adaptive allocation depends on the amount of heterogeneity exposed by the available supervision. As shown in Table 11, Cora and Citeseer contain only 20 labeled training nodes per class under the standard public splits, yielding exactly uniform training class-frequency priors. Consequently, ratio-driven and uniform allocations are effectively equivalent at the class level on these datasets, and adaptive allocation can benefit only from differences in predictive difficulty and intra-class heterogeneity rather than from correcting class-frequency imbalance. At their largest reported budgets, $B = |V_{\text{tr}}|$, so the refinement process reaches its saturation boundary and all termi-

nal regions become singletons; these settings therefore no longer retain policy-dependent allocation freedom.

Our allocation criteria are task-aware but remain surrogate objectives for downstream performance. AUA uses predictive disagreement and propagated task-space structure, while LUA further learns refinement preferences from task-grounded marginal gains. These signals avoid coupling allocation to a specific GNN architecture, which is consistent with the cross-architecture transfer results. A tighter integration with downstream model feedback while preserving such generality is left for future work.

Table 11: Training supervision and split characteristics of the evaluated datasets. Avg. Train/Class denotes the average number of labeled training nodes per class, while "Balanced" indicates whether the training protocol fixes the same number of labeled nodes for every class.

Dataset	Classes	Train	Avg. Train/Class	Balanced	Split Protocol
Cora	7	140	20	Yes	Fixed public
Citeseer	6	120	20	Yes	Fixed public
Flickr	7	44,625	6,375	No	Random
Reddit	41	153,932	3,754	No	Temporal
ogbn-arxiv	40	90,941	2,274	No	Publication time
ogbn-products	47	196,615	4,183	No	Sales rank