
Graph Data Augmentation via Contrastive Generator Inversion (DCBA)

Mateusz Stolarski*

Wrocław University of Science and Technology
University of Technology Sydney
mateusz.stolarski@pwr.edu.pl

Michał Czuba*

Wrocław University of Science and Technology
michal.czuba@pwr.edu.pl

Łukasz Kraiński*

SGH Warsaw School of Economics
lkrain@sgh.waw.pl

Katarzyna Musiał

University of Technology Sydney
katarzyna.musial-gabrys@uts.edu.au

Paweł Prałat

Toronto Metropolitan University
pralat@torontomu.ca

Bogumił Kamiński

SGH Warsaw School of Economics
bogumil.kaminski@sgh.waw.pl

Piotr Bródka

Wrocław University of Science and Technology
University of Technology Sydney
piotr.brodka@pwr.edu.pl

Abstract

Graphs provide a natural representation of many complex systems, ranging from social platforms to ecosystems. However, the development of graph-based machine learning methods is often constrained by the limited availability of large and diverse graph datasets. In this paper, we introduce DCBA, a model-based approach to graph data augmentation that infers the configuration of a synthetic graph generator from an observed network. We instantiate the proposed framework using the ABCD generator, which produces scale-free networks with community structure. Our model learns a joint representation of graphs and generator parametrisations using a multi-positive contrastive objective with soft negative weighting. The learned representation enables the prediction of an ABCD configuration whose stochastic realisations preserve the macrostructural properties encoded by the generator. Experiments show that DCBA recovers generator parameters more accurately and robustly than an algorithmic inverse-modelling baseline. Its downstream utility is further demonstrated in community detection, where inferred configurations used to fine-tune PProCD improve AMI on average by 161% on synthetic and 273% on real-world networks.

1 Introduction

Among the principal data domains considered within geometric deep learning [4] are graphs. They serve as a convenient means of modelling systems that depend on both individual entities and the relations between them, including social platforms [23], biological systems [14], and transport networks [5]. Although machine learning methods allow many problems arising in such domains to be addressed, their development remains constrained by the limited availability of large, diverse, and accessible graph datasets [36, 37]. Real-world networks may be costly to collect, restricted by privacy considerations, or available only as a single observed realisation of an underlying system.

Data augmentation is widely used to alleviate data scarcity. In computer vision, geometric transformations such as rotations or cropping can create additional samples while generally preserving their semantic content [27]. Such transformations are less straightforward for graph-structured data, whose semantics may depend jointly on connectivity, attributes, and higher-order structural patterns. Even small modifications can therefore remove meaningful relations or introduce implausible ones.

*Equal contribution.

In many applications, the desired outcome is not merely a perturbed version of the observed graph, but multiple plausible realisations that preserve selected macrostructural properties of the underlying system. This motivates performing augmentation through a fitted stochastic graph generator rather than through individual node- or edge-level operations.

1.1 Structural Graph Augmentation

Graph data augmentation methods can be classified according to the graph elements they modify, the operating scale, and whether the transformation is predefined or learned [36, 37]. Common structural operations include node dropping, edge perturbation, rewiring, and subgraph sampling. For example, GraphCL constructs contrastive views using several predefined transformations [34], whereas GAug learns task-oriented edge modifications [35]. Such methods are well suited to transforming a particular observed graph, but their local modifications are task-dependent and provide limited control over macrostructural properties. They are therefore less natural when the objective is to obtain multiple realisations of the same system while preserving properties such as its degree distribution and community organisation. It is important to note that structural modifications extend beyond data augmentation; for example, in network anonymisation, the observed graph is deliberately altered to reduce node uniqueness while preserving selected properties or analytical utility [15].

1.2 Generator-based Graph Augmentation

Synthetic graph generators provide a natural mechanism for producing additional network realisations under explicit structural assumptions. Different families encode different properties, including homogeneous connectivity [9], heterogeneous degree distributions [2], and community structure [12]. Once a suitable generator and its configuration are selected, multiple stochastic realisations can be sampled while controlling the macrostructural properties represented by the model.

Previous work has already explored this idea. For instance, a framework for synthetic attributed-graph generation was introduced in [8], combining a generalised R-MAT/SKG structural model [6, 20] with separate generators for node and edge attributes, enabling the generation of datasets substantially larger than the original input. G-Mixup instead estimates class-level *graphons* from collections of labelled graphs, interpolates between them, and samples synthetic graphs with mixed labels [11]. These methods require either fitting a multi-component generative pipeline to an available dataset or access to multiple labelled graphs, and do not directly address the recovery of a reusable parametric generator configuration from a single observed network.

1.3 Inverse Graph Modelling

Graph-generator calibration has previously been approached through simulation-based inference. For example, Approximate Bayesian Computation combined with spectral graph metrics can estimate generator parameters without a tractable likelihood, but requires repeated simulation and comparison for each observed network [10]. More recently, calibration of the multilayer mABCD [19] generator combined direct estimators with iterative optimisation [3]. This approach also highlights a limitation of estimating parameters separately: their interdependence allows errors in one step to affect subsequent estimates, motivating joint recovery of the complete generator configuration.

1.4 Proposed Approach and Contributions

In this paper, we introduce DCBA, a model-based approach that infers the configuration of a graph generator from an observed network. Rather than directly perturbing its structure, it predicts a reusable configuration from which multiple related graphs can be sampled. Following the taxonomies [36, 37], DCBA can be positioned as a learned, generation-based method for graph-level structural augmentation.

The principal contributions of this work are as follows:

- We formulate graph augmentation through learned, amortised inversion of a stochastic graph generator, enabling multiple related graphs to be generated from a single observed network.
- We introduce DCBA, which aligns graph and configuration representations through a multi-positive contrastive objective with soft negative weighting, jointly predicts the complete generator configuration, and is instantiated using the ABCD [16] synthetic graph generator.

- We demonstrate accurate parameter recovery (mean R^2 of 0.761/0.527 on casual/hard sets), outperforming algorithmic baselines, and show that DCBA-based augmentation improves community detection with PProCD [24] on real-world networks by 273% in AMI on average.

2 Problem Formulation

Having introduced the initial concept of the model, this section formalises the problem without restricting it to a particular synthetic graph generator. Let

$$G = (V, E, \mathbf{X}) \quad (1)$$

denote a graph, where V is the set of vertices, E is the set of edges, and $\mathbf{X} \in \mathbb{R}^{|V| \times d}$ is the matrix of vertex features, with d denoting the number of features associated with each vertex.

Next, let $\mathcal{G}$ denote a stochastic graph generator parameterised by a configuration vector $\theta \in \Theta$. The generator induces a conditional distribution over graphs such that

$$G \sim p_{\mathcal{G}}(G \mid \theta). \quad (2)$$

Given an observed graph G , the objective is to learn an estimator f_{ϕ} , parameterised by ϕ , that predicts the underlying generator configuration:

$$\hat{\theta} = f_{\phi}(G). \quad (3)$$

To address this problem, let the training dataset be defined as

$$\mathcal{D} = \left\{ \left(\theta_i, G_i^{(1)}, G_i^{(2)}, \dots, G_i^{(R_i)} \right) \right\}_{i=1}^N, \quad G_i^{(r)} \sim p_{\mathcal{G}}(G \mid \theta_i), \quad (4)$$

where N denotes the number of distinct generator configurations and R_i denotes the number of graph realisations generated for configuration θ_i . Each tuple in $\mathcal{D}$ is referred to as an *instance*, whereas each graph generated within an instance is referred to as a *replica*.

2.1 Model Components

Before introducing the objective, let us note that a graph G and a configuration θ constitute different modalities describing the same underlying generative process. Accordingly, we adopt a two-tower architecture inspired by CLIP-like representation learning [25]. The two modalities are processed by separate encoders, while their representations are aligned within a shared latent space. Let

$$e_{\phi_G} : G \mapsto h_G, \quad e_{\phi_{\theta}} : \theta \mapsto h_{\theta} \quad (5)$$

denote the graph encoder, parameterised by ϕ_G , and the configuration encoder, parameterised by ϕ_{θ} , respectively. Both encoders produce representations in the same k -dimensional latent space:

$$h_{G,i}^{(r)} = e_{\phi_G}(G_i^{(r)}) \in \mathbb{R}^k, \quad h_{\theta,i} = e_{\phi_{\theta}}(\theta_i) \in \mathbb{R}^k. \quad (6)$$

The configuration encoder forms part of an autoencoder whose decoder, parameterised by ϕ_D , is defined as

$$d_{\phi_D} : \mathbb{R}^k \rightarrow \Theta. \quad (7)$$

The decoder is shared between the configuration reconstruction and graph-to-configuration paths:

$$\tilde{\theta}_i = d_{\phi_D}(h_{\theta,i}), \quad \hat{\theta}_i^{(r)} = d_{\phi_D}(h_{G,i}^{(r)}). \quad (8)$$

Consequently, the estimator introduced previously can be expressed as the composition

$$f_\phi = d_{\phi_D} \circ e_{\phi_G}, \quad \phi = \{\phi_G, \phi_\theta, \phi_D\}. \quad (9)$$

The autoencoding path encourages the configuration tower to learn a stable and informative representation of the generator parameter space. The contrastive objective subsequently aligns graph representations with this space, allowing d_{ϕ_D} to be applied directly to graph representations at inference time. Although e_{ϕ_θ} participates in the training procedure, the graph-to-configuration inference path depends only on e_{ϕ_G} and the shared d_{ϕ_D} .

2.2 Configuration Prediction and Reconstruction

Let $\mathcal{I}$ denote the set of generator configurations included in a mini-batch, and let $\mathcal{R}_i$ denote the set of replica indices sampled for configuration θ_i . The graph-to-configuration regression objective is

$$\mathcal{L}_{\text{reg}} = \frac{1}{\sum_{i \in \mathcal{I}} |\mathcal{R}_i|} \sum_{i \in \mathcal{I}} \sum_{r \in \mathcal{R}_i} \ell_\Theta \left(\tilde{\theta}_i^{(r)}, \theta_i \right), \quad (10)$$

whereas the reconstruction objective of the configuration autoencoder is defined as

$$\mathcal{L}_{\text{rec}} = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \ell_\Theta \left(\tilde{\theta}_i, \theta_i \right). \quad (11)$$

Here, $\ell_\Theta : \Theta \times \Theta \rightarrow \mathbb{R}_+$ denotes a discrepancy function defined over pairs of generator configurations. Its particular form may depend on the parameterisation of the considered generator $\mathcal{G}$. In the simplest case, it may be instantiated as the mean squared error, whereas generator-specific implementations may additionally account for differences in parameter scales or penalise predicted configurations that violate the feasibility constraints of the generator, such as non-negativity.

2.3 Multi-positive Contrastive Objective with Soft Negative Weighting

As $\mathcal{G}$ is nondeterministic, several graphs may be generated from the same configuration. These replicas should occupy nearby regions of the latent space despite the differences introduced by the stochastic generation process. Thus, we employ a multi-positive supervised contrastive objective [18] with soft negative weighting, extending the bidirectional alignment principle used in [25].

The normalised latent representations are defined as

$$z_{G,i}^{(r)} = \frac{h_{G,i}^{(r)}}{\|h_{G,i}^{(r)}\|_2}, \quad z_{\theta,i} = \frac{h_{\theta,i}}{\|h_{\theta,i}\|_2}. \quad (12)$$

For a given mini-batch, let

$$\mathcal{Z}_G = \left\{ z_{G,i}^{(r)} : i \in \mathcal{I}, r \in \mathcal{R}_i \right\}, \quad \mathcal{Z}_\theta = \{ z_{\theta,i} : i \in \mathcal{I} \} \quad (13)$$

denote the collection of graph representations and the collection containing one configuration representation for each generator configuration, respectively. The complete representation pool is

$$\mathcal{Z} = \mathcal{Z}_G \cup \mathcal{Z}_\theta. \quad (14)$$

Each representation $z \in \mathcal{Z}$ is associated with an instance label $y(z)$ and its corresponding generator configuration $\theta(z)$. For an anchor $a \in \mathcal{Z}$, the positive and negative sets are defined, respectively, as

$$\mathcal{P}(a) = \{ z \in \mathcal{Z} \setminus \{a\} : y(z) = y(a) \}, \quad \mathcal{N}(a) = \{ z \in \mathcal{Z} : y(z) \neq y(a) \}. \quad (15)$$

Thus, for a graph anchor $z_{G,i}^{(r)}$, the positive set contains the configuration representation $z_{\theta,i}$ and all other graph replicas generated from θ_i . For a configuration anchor $z_{\theta,i}$, the positive set contains all graph replicas generated from that configuration.

Since all representations are normalised, their similarity scaled by the $\tau_s > 0$ constant is defined as

$$s(a, z) = \frac{a^\top z}{\tau_s}. \quad (16)$$

Generator configurations form a continuous parameter space. Consequently, two configurations located close to one another should not be treated as equally strong negatives as configurations lying far apart. For a negative pair (a, z) , we therefore define the distance-dependent weight

$$w(a, z) = 1 - \exp\left(-\frac{\|\theta(a) - \theta(z)\|_2}{\tau_w}\right), \quad (17)$$

where $\tau_w > 0$ controls the rate at which the weight increases with configuration distance. The weight approaches zero for nearly identical configurations and approaches one as the distance between configurations increases.

Therefore, for an anchor a , the normalising term is defined as

$$D(a) = \sum_{p \in \mathcal{P}(a)} \exp(s(a, p)) + \sum_{n \in \mathcal{N}(a)} w(a, n) \exp(s(a, n)). \quad (18)$$

The term $D(a)$ can be interpreted as the total similarity mass of all representations competing with respect to anchor a . The more similar a representation (whether positive or negative) is to the anchor, the more strongly it contributes to $D(a)$. For negatives, this contribution is additionally modulated by $w(a, n)$. Consequently, $D(a)$ ensures that the similarity of each positive representation is evaluated relative to the remaining representations in the mini-batch.

Finally, for a collection of anchors $\mathcal{A} \subseteq \mathcal{Z}$, the multi-positive contrastive loss, averaged over all anchor-positive pairs, is defined as

$$\mathcal{L}_{\text{MP}}(\mathcal{A}) = -\frac{1}{\sum_{a \in \mathcal{A}} |\mathcal{P}(a)|} \sum_{a \in \mathcal{A}} \sum_{p \in \mathcal{P}(a)} \log \frac{\exp(s(a, p))}{D(a)}. \quad (19)$$

The loss is evaluated bidirectionally. Graph representations act as anchors in the first direction, whereas configuration representations act as anchors in the second direction:

$$\mathcal{L}_{\text{con}} = \mathcal{L}_{\text{MP}}(\mathcal{Z}_G) + \mathcal{L}_{\text{MP}}(\mathcal{Z}_\theta). \quad (20)$$

In conclusions, the objective encourages graph replicas generated from the same configuration to form a coherent cluster around their configuration representation, while configurations that are distant in parameter space provide a stronger repulsive signal than nearby configurations.

2.4 Joint Learning Objective

The complete model is trained by minimising

$$\phi^* = \arg \min_{\phi} \mathcal{L}_{\text{total}}(\phi; \mathcal{D}), \quad (21)$$

where

$$\mathcal{L}_{\text{total}} = \lambda_{\text{reg}} \mathcal{L}_{\text{reg}} + \lambda_{\text{rec}} \mathcal{L}_{\text{rec}} + \lambda_{\text{con}} \mathcal{L}_{\text{con}}. \quad (22)$$

The regression term grounds the latent representation in the original generator-parameter space, the reconstruction term regularises the configuration tower, and the contrastive term aligns graph and configuration representations while promoting invariance to the stochasticity of the graph generator.

3 Framework Instantiation – DCBA

The framework introduced in Sec. 2 is instantiated using the ABCD model [16]. Before discussing the DCBA architecture (shown in Fig. 1), we first describe the generator itself and the dataset constructed from its realisations.

3.1 Employed Graph Generator

ABCD is a graph generator attractive due to its rich modelling space tailored for social networks and the non-trivial retrieval of input parameters from its realisations. The model is *scale-free* – both the degree sequence and the community-size sequence are drawn from power law distributions to mimic the heavy-tailed characteristic of real-world networks. Moreover, each generated graph is accompanied by a ground-truth community assignment.

Succinctly, ABCD generates a graph with n vertices as the union of independently sampled components: a background graph carrying inter-community noise and one graph associated with each community. A degree sequence $\mathbf{d}$ and a community-size sequence $\mathbf{m}$ are sampled from truncated power-law distributions. For each vertex v , its prescribed degree d_v is divided into a community contribution $d_v^{\text{com}} \approx (1 - \xi)d_v$ and a background contribution $d_v^{\text{bg}} \approx \xi d_v$. Hence, the mixing parameter $\xi \in [0, 1]$ controls the expected proportion of a vertex’s edges assigned to the background graph. Values approaching zero produce strongly separated communities, whereas values approaching one progressively weaken the community structure. The independently generated components are subsequently combined and simplified by removing self-loops and multiple edges.

ABCD+o [17] extends this construction with a specified number of outlier vertices – n_{out} . Outliers are not assigned to any community graph, and their incident edges are generated exclusively within the background component. Hence, they have no dominant community affiliation. Setting n_{out} to zero recovers the original ABCD model. We therefore adopt ABCD+o as the generator underlying DCBA, since it subsumes the original model while introducing an additional characteristic to be recovered.

3.2 Utilised Dataset

Training data were produced by sampling 10 000 candidate ABCD+o configurations θ from the ranges reported in Tab. 1 and generating independent stochastic replicas for each accepted configuration. Concretely, for every candidate configuration, each parameter was sampled independently and uniformly from its corresponding interval. Configurations that violated the feasibility constraints of ABCD+o were discarded. For details on the dataset generation, see App. B.

The procedure yielded 8,744 accepted configurations, each constituting one *instance* in the sense defined in Sec. 2. For every instance, ten independent graph *replicas* were generated using distinct random seeds, resulting in a dataset of 87,440 graphs together with their community assignments. Separately, a boundary challenge set comprising 486 configurations and 4,860 graphs was generated from combinations of parameter values located at the limits of the ranges reported in Tab. 1. It was intended to evaluate the model on challenging configurations and thereby assess its robustness.

Table 1: Sampling ranges used to construct the dataset. Graph-size-dependent parameters are drawn as fractions of n and subsequently converted to valid integer-valued parameters of ABCD+o.

Symbol	Sampling range	Description
n	[1 000, 10 000]	Number of nodes in the graph.
t_1	[2.0, 3.0]	Power-law exponent of the degree distribution.
$d_{\min}$	$[10^{-10}, 0.005] \cdot n$	Minimum node degree.
$d_{\max}$	$[0.005, 0.5] \cdot n$	Maximum node degree.
t_2	[1.0, 2.0]	Power-law exponent of the community-size distribution.
$c_{\min}$	$[10^{-10}, 0.05] \cdot n$	Minimum community size.
$c_{\max}$	$[0.05, 0.5] \cdot n$	Maximum community size.
ξ	[0.01, 1.0]	Fraction of edges placed in the background graph.
n_{out}	$[0.0, 0.05] \cdot n$	Number of outlier nodes (ABCD+o) excluded from every community.

3.3 Proposed Model

Everything the model can infer about an ABCD realisation has to fit into h_G . The central design question is therefore how much of a graph’s macrostructure can be compressed into a single vector. In particular, h_G must simultaneously capture the degree distribution, community organisation, and the extent of inter-community mixing. Since the components of θ are predicted jointly, information omitted by the graph encoder cannot be recovered further down the model. Figure 1 shows the architecture designed around this requirement, while App. A describes the employed loss function.

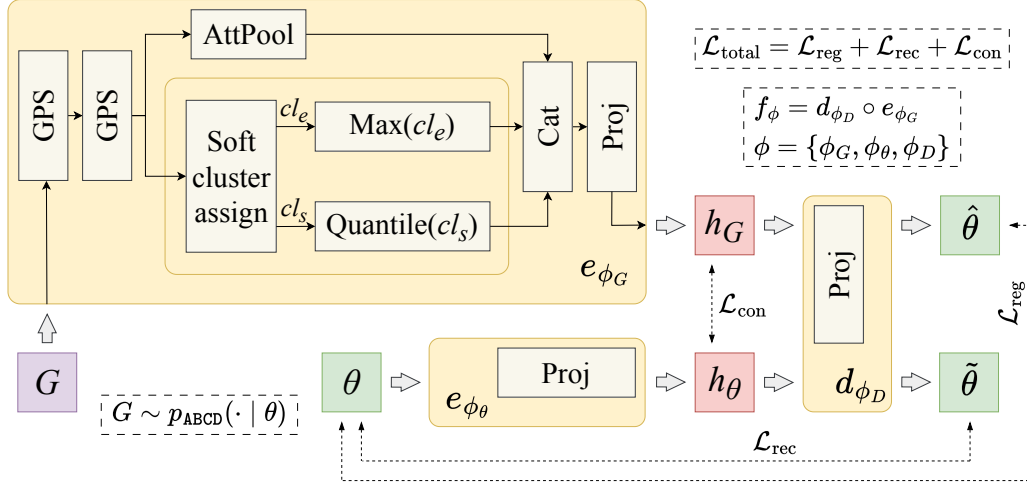


Figure 1: Architecture of DCBA. During training, ABCD configurations θ and their graph realisations G are mapped by e_{ϕ_θ} and e_{ϕ_G} , respectively, into a shared latent space. Representations originating from the same configuration are pulled together, whereas those of different configurations are pushed apart with a strength proportional to their distance in the configuration space. A soft-clustering pool assigns vertices to K latent slots and summarises their embeddings cl_e and sizes cl_s independently of slot order. At inference, only the path $f_\phi = d_{\phi_D} \circ e_{\phi_G}$ is used to recover $\hat{\theta}$ from an observed graph.

Encoding Vertices. The graph encoder e_{ϕ_G} is composed of GraphGPS layers [26]. Each layer combines local GIN message passing [32] with global multi-head self-attention, implemented using a linear approximation [7] to reduce memory requirements for the graph sizes in Tab. 1. This allows each layer to combine local and graph-wide information. Depending on the experimental setting, vertices receive either a constant feature or features encoding community membership.

Aggregating Vertices into h_G . Vertex embeddings are aggregated along two branches. Attention pooling [21] provides a weighted average, while the second branch softly assigns vertices to K learned slots and reads out order statistics of their representations and sizes. Consequently, h_G captures both the averaged vertex representation and how vertex embeddings accumulate within slots.

Soft-clustering Regularisation. To make the slot sizes behave as counts rather than diffuse mass, the training objective includes an additional entropy term that encourages each vertex to concentrate its assignment on a single slot. The slots need not correspond to communities, and K only bounds how many may be used. The complete formulation is provided in App. A.3.

Representing Configurations. The configuration encoder e_{ϕ_θ} and the shared decoder d_{ϕ_D} are multilayer perceptrons with mirrored layer widths. As several components of θ scale with the graph order, both ℓ_Θ and the weight $w(a, z)$ would otherwise be dominated by n . Therefore, configurations are scaled before being used by the model, with size-related components transformed logarithmically and the remaining ones linearly (App. A.1). The decoder outputs lie in the same normalised space.

3.4 Baseline Inverse Model

As a non-learned reference point for DCBA, we developed an algorithmic baseline that estimates the vector θ directly from an observed ABCD+o graph. Its formulation is provided in App. C. The

baseline follows a recovery procedure in which the components of θ are estimated independently, similarly to [3]. Parameters directly observable in the graph, such as the minimum degree, are obtained therefrom, whereas the remaining parameters are estimated from the corresponding structural characteristics. We consider two variants. Base-T uses the ground-truth community assignment associated with the generated ABCD+o graph, whereas Base-L detects communities using the Leiden algorithm [29]. Thus, it can be applied when the true community assignment is unavailable.

4 Experiments

The trained model was compared with the algorithmic baseline using the coefficient of determination, R^2 , computed separately for each component of θ between the ground-truth and predicted values. To prevent data leakage, all graphs generated from a given θ were assigned jointly to the same data fold.

4.1 Prediction Performance

We compare three learned variants: the standard DCBA; DCBA-Comm, which uses the size of the community containing each vertex as a node feature; and DCBA-Comm64, additionally trained with a batch size of 64. As Base-T uses ground-truth community information, it should be compared to DCBA-Comm and DCBA-Comm64, whereas Base-L corresponds to DCBA. Let us recall that the baselines also have an inherent advantage for directly observable parameters such as n and $d_{\min}$.

Table 2: Parameter-wise R^2 on the casual and hard test sets; first- and second-best results within each split are shown in dark and pale green, respectively; the bottom row reports the overall mean.

Param.	Casual test set					Hard test set				
	Base-T	Base-L	DCBA	DCBA Comm	DCBA Comm64	Base-T	Base-L	DCBA	DCBA Comm	DCBA Comm64
n	1.000	1.000	0.964	0.948	0.951	1.000	1.000	0.930	0.808	0.704
t_1	-5.425	-5.425	0.959	0.914	0.927	-66.760	-66.760	0.666	0.489	0.606
$d_{\min}$	1.000	1.000	0.971	0.946	0.970	1.000	1.000	0.946	0.791	0.731
$d_{\max}$	0.661	0.661	0.801	0.740	0.779	0.449	0.449	0.590	0.680	0.470
t_2	-2.031	-16.130	0.007	0.105	0.089	-86.160	-36.430	-0.012	-0.082	0.204
$c_{\min}$	0.635	-0.048	0.439	0.861	0.876	0.966	-0.194	-0.013	0.778	0.639
$c_{\max}$	0.619	0.404	0.544	0.782	0.790	0.471	-0.024	0.328	0.615	0.446
ξ	0.923	0.637	0.859	0.981	0.980	0.770	0.178	0.040	0.471	0.548
n_{out}	-2.760	-1.512	0.306	0.480	0.484	-45.250	-0.356	0.214	0.197	0.069
$\overline{R^2}$	-0.598	-2.157	0.650	0.751	0.761	-21.501	-11.237	0.410	0.527	0.491

Table 2 presents results for both sets. On the “casual” one, all DCBA variants substantially outperform the baselines for the power-law exponents and n_{out} . Community features markedly improve the recovery of $c_{\min}$, $c_{\max}$, and ξ , with DCBA-Comm64 achieving the highest overall $\overline{R^2}$. On the hard set, the algorithmic baselines deteriorate sharply for t_1 , t_2 , and n_{out} , whereas the learned variants remain substantially more robust. Here, DCBA-Comm attains the highest overall score, while the vanilla model performs best for several parameters, indicating better generalisation under distribution shift than the larger-batch variant. Community features remain particularly beneficial for community-related parameters. Finally, t_2 remains the most difficult parameter to recover, as its estimation is based on fewer community sizes (less than $\sqrt{n}$) than the sequence of node degrees (n) governed by t_1 . Outliers are also structurally similar to noise and hence challenging for both the baselines and DCBA.

4.2 Applicability to the Community Detection Task

Since the framework was instantiated with ABCD, a model designed primarily for benchmarking community detection methods [16], we evaluated whether DCBA-based augmentation can improve a trainable community detection model. To this end, we selected PRoCD [24], a neural method that does not require prior knowledge of the number of communities and can be trained on synthetic corpora, making it well suited to data augmentation. The experiment comprised two complementary parts.

The first one used a controlled synthetic setting. Since PRoCD’s robustness deteriorated for networks with weak communities, creating room for augmentation, we selected 22 graphs with $\xi > 0.25$ from the held-out “casual” dataset (Sec. 3.2). Performance without adaptation constituted the baseline

condition (B). For each target, we then fine-tuned (thrice with different random seeds) the model on cohorts of 20 synthetic ABCD graphs generated under three conditions. In the random condition (R), configurations were sampled independently of the target, controlling for the general effect of ABCD data. In the DCBA condition (D), the cohort was generated from the configuration inferred by DCBA trained without ground-truth community information. Finally, in the oracle condition (O), graphs were generated from the configuration used to create the target graph. Thus, R controls for non-targeted augmentation, while O provides a reference with exact knowledge of the target.

Table 3: Community detection with PRoCD on unseen ABCD graphs under different fine-tuning regimes. AMI is averaged while ΔB and p compare each regime with B using a paired Wilcoxon test.

Mode	Fine-tuning corpus	AMI	ΔB	p
B	none (frozen checkpoint)	0.189	–	–
R	random ABCD graphs	0.378	+0.189	0.0057
D	inferred $\hat{\theta}$ with DCBA	0.493	+0.304	$< 10^{-4}$
O	ground-true θ	0.541	+0.351	$< 10^{-4}$

The results in Tab. 3 support the applicability of the proposed augmentation method. Fine-tuning on cohorts generated from DCBA-inferred configurations improves AMI (Adjusted Mutual Information [30], a permutation-invariant clustering score adjusted for chance, with 1 denoting perfect agreement and 0 random expectation) by 0.304 over the frozen checkpoint, with improvements observed on 20 of 22 target graphs ($p < 10^{-4}$). Importantly, it also outperforms non-targeted ABCD augmentation by 0.115 on average (20 of 22 targets, $p < 10^{-3}$), indicating additional benefit from target matching. Moreover, DCBA-based augmentation recovers 88% of the oracle improvement.

Having demonstrated the utility of DCBA in augmenting in-distribution graphs, we next considered a real-world setting. The experiment was repeated on six networks with ground-truth communities (see Tab. 4). For each target, we evaluated the published PRoCD checkpoint directly and after fine-tuning, repeated with three random seeds, on 20 ABCD graphs generated from configurations inferred by either Base-L or vanilla DCBA. The results further support the applicability of DCBA-based augmentation, although the gains are less uniform than in the synthetic setting. Across the six networks, DCBA achieves a mean AMI of 0.271, compared with 0.209 for Base-L and 0.175 for the default checkpoint, corresponding to improvements of 29% and 55%, respectively. The largest gain is observed for PolBlogs, while EuroSiS and Cora-Full show no improvement over the original checkpoint, indicating that target-specific augmentation is most beneficial when there is sufficient room for adaptation.

Table 4: Community detection with PRoCD on six real-world networks from [13] under different fine-tuning regimes, measured by average AMI against ground-truth communities with standard deviation over three runs for the best-performing regime.

Graph	Pretrained	Base-L	DCBA	σ
PolBlogs [1]	0.021	0.020	0.291	0.000
Newsgroup 5cl-1 [33]	0.018	0.025	0.031	0.000
Newsgroup 5cl-2 [33]	0.112	0.194	0.263	0.053
Newsgroup 5cl-3 [33]	0.096	0.226	0.236	0.001
EuroSiS [31]	0.603	0.602	0.603	0.000
Cora-Full: OS [28]	0.199	0.187	0.199	0.000

5 Conclusions

This paper proposes a new approach to graph data augmentation through inverse graph modelling rather than micro-level perturbations. Inspired by CLIP [25], the introduced architecture aligns graph and generator-configuration representations through contrastive learning and uses a configuration autoencoder to support stable training. Experiments show that its ABCD-specific instantiation, DCBA, robustly recovers most generator parameters and outperforms the algorithmic baseline, except for directly observable quantities. The study also validates its utility in both curated and real-world community detection settings, showing that fine-tuning corpora obtained with the graph-DCBA-ABCD pipeline improve PRoCD by up to 161% and 273% in AMI, respectively. However, the major limitation is its dependence on a particular generator, whose assumptions may not necessarily hold for an observed graph. Future work should therefore address generator selection as a preceding stage.

Acknowledgements. This research was partially supported by: Horizon Europe, EU, grant no. 10108632; National Science Centre, Poland, grant no. 2022/45/B/ST6/04145; Polish National Agency for Academic Exchange, Strategic Partnerships, grant no. BPI/PST/2024/1/00129/U/00001; Wrocław University of Science and Technology, Academia Professorum Iuniorum and Minigrants projects. Views and opinions expressed here are those of the authors and do not necessarily reflect those of the funding agencies.

References

- [1] Lada A Adamic and Natalie Glance. “The political blogosphere and the 2004 US election: divided they blog”. In: *Proceedings of the 3rd international workshop on Link discovery*. 2005, pp. 36–43. DOI: [10.1145/1134271.1134277](https://doi.org/10.1145/1134271.1134277). 9
- [2] Bela Bollobas et al. “Directed Scale-Free Graphs”. In: *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Jan. 2003, pp. 132–139. URL: <https://www.microsoft.com/en-us/research/publication/directed-scale-free-graphs>. 2
- [3] Piotr Bródka et al. “Twinning Complex Networked Systems: Data-Driven Calibration of the mABCD Synthetic Graph Generator”. In: *Modelling and Mining Networks*. Cham: Springer Nature Switzerland, 2026, pp. 1–17. DOI: [10.1007/978-3-032-27193-8_1](https://doi.org/10.1007/978-3-032-27193-8_1). 2, 8
- [4] Michael M. Bronstein et al. “Geometric Deep Learning: Going beyond Euclidean data”. In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42. DOI: [10.1109/MSP.2017.2693418](https://doi.org/10.1109/MSP.2017.2693418). 1
- [5] Alessio Cardillo et al. “Emergence of network features from multiplexity”. In: *Scientific Reports* 3.1344 (Feb. 2013), pp. 1–6. DOI: [10.1038/srep01344](https://doi.org/10.1038/srep01344). 1
- [6] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. “R-MAT: A recursive model for graph mining”. In: *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM. 2004, pp. 442–446. DOI: [10.1137/1.9781611972740.43](https://doi.org/10.1137/1.9781611972740.43). 2
- [7] Krzysztof Marcin Choromanski et al. “Rethinking Attention with Performers”. In: *International Conference on Learning Representations*. 2021. URL: <https://openreview.net/forum?id=Ua6zuk0WRH>. 7, 17
- [8] Sajad Darabi et al. “A Framework for Large-Scale Synthetic Graph Dataset Generation”. In: *IEEE Transactions on Neural Networks and Learning Systems* 36.8 (2025), pp. 14258–14268. DOI: [10.1109/TNNLS.2025.3540392](https://doi.org/10.1109/TNNLS.2025.3540392). 2
- [9] Paul Erdős, Alfréd Rényi, et al. “On the evolution of random graphs”. In: *Publ. math. inst. hung. acad. sci* 5.1 (1960), pp. 17–60. URL: https://static.renyi.hu/~p_erdos/1960-10.pdf. 2
- [10] Damien Fay et al. “Graph metrics as summary statistics for Approximate Bayesian Computation with application to network model parameter estimation”. In: *Journal of Complex Networks* 3.1 (Mar. 2015), pp. 52–83. DOI: [10.1093/comnet/cnu009](https://doi.org/10.1093/comnet/cnu009). 2
- [11] Xiaotian Han et al. “G-Mixup: Graph Data Augmentation for Graph Classification”. In: *Proceedings of the 39th International Conference on Machine Learning*. Vol. 162. Proceedings of Machine Learning Research. PMLR, July 2022, pp. 8230–8248. URL: <https://proceedings.mlr.press/v162/han22c.html>. 2
- [12] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. “Stochastic blockmodels: First steps”. In: *Social networks* 5.2 (1983), pp. 109–137. DOI: [10.1016/0378-8733\(83\)90021-7](https://doi.org/10.1016/0378-8733(83)90021-7). 2
- [13] Vladimir Ivashkin and Pavel Chebotarev. “Do logarithmic proximity measures outperform plain ones in graph clustering?” In: *International Conference on Network Analysis*. Springer. 2016, pp. 87–105. DOI: [10.1007/978-3-319-56829-4_8](https://doi.org/10.1007/978-3-319-56829-4_8). 9
- [14] Hawoong Jeong et al. “Lethality and Centrality in Protein Networks”. In: *Nature* 411 (2001), pp. 41–42. DOI: [10.1038/35075138](https://doi.org/10.1038/35075138). 1
- [15] R. G. de Jong, M. P. J. van der Loo, and F. W. Takes. “The Anonymization Problem in Social Networks”. In: *Modelling and Mining Networks*. Cham: Springer Nature Switzerland, 2026, pp. 18–33. ISBN: 978-3-032-27193-8. DOI: [10.1007/978-3-032-27193-8_2](https://doi.org/10.1007/978-3-032-27193-8_2). 2
- [16] Bogumił Kamiński, Paweł Prałat, and François Théberge. “Artificial Benchmark for Community Detection (ABCD)—Fast random graph model with community structure”. In: *Network Science* 9.2 (2021), pp. 153–178. DOI: [10.1017/nws.2020.45](https://doi.org/10.1017/nws.2020.45). 2, 6, 8

- [17] Bogumił Kamiński, Paweł Prałat, and François Théberge. “Artificial benchmark for community detection with outliers (ABCD+o)”. In: *Applied Network Science* 8.25 (2023). DOI: [10.1007/s41109-023-00552-9](https://doi.org/10.1007/s41109-023-00552-9). 6, 17
- [18] Prannay Khosla et al. “Supervised Contrastive Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 18661–18673. DOI: [10.48550/arXiv.2004.11362](https://doi.org/10.48550/arXiv.2004.11362). 4
- [19] Łukasz Krainiński et al. “Multilayer artificial benchmark for community detection (mABCD)”. In: *Expert Systems with Applications* 307 (2026), p. 130920. DOI: [10.1016/j.eswa.2025.130920](https://doi.org/10.1016/j.eswa.2025.130920). 2
- [20] Jurij Leskovec et al. “Realistic, Mathematically Tractable Graph Generation and Evolution, Using Kronecker Multiplication”. In: *Knowledge Discovery in Databases: PKDD 2005* (2005), pp. 133–145. DOI: [10.1007/11564126_17](https://doi.org/10.1007/11564126_17). 2
- [21] Yujia Li et al. “Graph matching networks for learning the similarity of graph structured objects”. In: *Proceedings of the 36th International conference on machine learning*. PMLR, 2019, pp. 3835–3845. URL: <https://proceedings.mlr.press/v97/li19d.html>. 7
- [22] Prasanta Chandra Mahalanobis. “On the Generalised Distance in Statistics”. In: *Proceedings of the National Institute of Sciences of India* 2.1 (1936), pp. 49–55. DOI: [10.1007/s13171-019-00164-5](https://doi.org/10.1007/s13171-019-00164-5). 22
- [23] Michał B. Paradowski et al. “Peer Interaction Dynamics and Second Language Learning Trajectories During Study Abroad: A Longitudinal Investigation Using Dynamic Computational Social Network Analysis”. In: *Language Learning* 74.S2 (Dec. 2024), pp. 58–115. DOI: [10.1111/lang.12681](https://doi.org/10.1111/lang.12681). 1
- [24] Meng Qin et al. “Pre-train and refine: Towards higher efficiency in k-agnostic community detection without quality degradation”. In: *Proc. of the 30th Conf. on Knowledge Discovery and Data Mining*. Aug. 2024, pp. 2467–2478. DOI: [10.1145/3637528.3671686](https://doi.org/10.1145/3637528.3671686). 3, 8
- [25] Alec Radford et al. “Learning Transferable Visual Models From Natural Language Supervision”. In: *Proceedings of the 38th International Conference on Machine Learning*. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 8748–8763. URL: <https://proceedings.mlr.press/v139/radford21a.html>. 3, 4, 9
- [26] Ladislav Rampášek et al. “Recipe for a General, Powerful, Scalable Graph Transformer”. In: *Advances in Neural Information Processing Systems*. Vol. 35. Curran Associates, Inc., 2022, pp. 14501–14515. DOI: [10.52202/068431-1054](https://doi.org/10.52202/068431-1054). 7
- [27] Connor Shorten and Taghi M. Khoshgoftaar. “A Survey on Image Data Augmentation for Deep Learning”. In: *Journal of Big Data* 6 (2019), p. 60. DOI: [10.1186/s40537-019-0197-0](https://doi.org/10.1186/s40537-019-0197-0). 1
- [28] Lovro Šubelj and Marko Bajec. “Model of complex networks based on citation dynamics”. In: *Proceedings of the 22nd international conference on World Wide Web*. 2013, pp. 527–530. DOI: [10.1145/2487788.2487987](https://doi.org/10.1145/2487788.2487987). 9
- [29] Vincent A. Traag, Ludo Waltman, and Nees Jan van Eck. “From Louvain to Leiden: guaranteeing well-connected communities”. In: *Scientific Reports* 9.5233 (2019). DOI: [10.1038/s41598-019-41695-z](https://doi.org/10.1038/s41598-019-41695-z). 8, 16
- [30] Nguyen Xuan Vinh, Julien Epps, and James Bailey. “Information theoretic measures for clusterings comparison: is a correction for chance necessary?” In: *Proc. of the 26th international conference on machine learning*. 2009, pp. 1073–1080. DOI: [10.1145/1553374.1553511](https://doi.org/10.1145/1553374.1553511). 9
- [31] WebAtlas. *EuroSiS web mapping study: mapping interactions between Science in Society actors on the Web of 12 European countries*. Dataset distributed with Gephi. 2010. URL: https://docs.gephi.org/desktop/User_Manual/Datasets/. 9
- [32] Keyulu Xu et al. “How Powerful are Graph Neural Networks?” In: *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=ryGs6iA5Km>. 7, 17
- [33] Luh Yen et al. “Graph nodes clustering based on the commute-time kernel”. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2007, pp. 1037–1045. DOI: doi.org/10.1007/978-3-540-71701-0_117. 9
- [34] Yuning You et al. “Graph Contrastive Learning with Augmentations”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 5812–5823. DOI: [10.48550/arXiv.2010.13902](https://doi.org/10.48550/arXiv.2010.13902). 2

- [35] Tong Zhao et al. “Data Augmentation for Graph Neural Networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.12 (May 2021), pp. 11015–11023. DOI: [10.1609/aaai.v35i12.17315.2](https://doi.org/10.1609/aaai.v35i12.17315.2)
- [36] Tong Zhao et al. *Graph Data Augmentation for Graph Machine Learning: A Survey*. 2023. DOI: [10.48550/arXiv.2202.08871](https://doi.org/10.48550/arXiv.2202.08871). arXiv: [2202.08871](https://arxiv.org/abs/2202.08871) [cs.LG]. 1, 2
- [37] Jiajun Zhou et al. “Data Augmentation on Graphs: A Technical Survey”. In: *ACM Comput. Surv.* 57.11 (June 2025). DOI: [10.1145/3732282](https://doi.org/10.1145/3732282). 1, 2

A ABCD+o-specific Loss

Following the general formulation of the training objective from Sec. 2, we show below the concrete loss function used to train DCBA, as described in Sec. 3. As presented in Eq. 22, the objective comprises three principal terms addressing different aspects of the problem. In DCBA, their corresponding weights were set, based on empirical experience and intuition, to $\lambda_{reg} = 1.0$, $\lambda_{rec} = 1.0$, and $\lambda_{con} = 0.3$. The contrastive term retains the form defined in Sec. 2.3, with its two normalisation parameters set to $\tau_s = 0.07$ and $\tau_w = 1.0$. The regression and reconstruction terms, only outlined in the main text, are instantiated using an ABCD+o-specific configuration loss (see App. A.2). In addition, the cluster-pooling branch of the graph encoder (Sec. 3.3) contributes an auxiliary clustering objective, which is detailed below. The following text also describes the normalisation of the configuration space on which these terms operate.

A.1 Configuration Scaling

The components of θ span incomparable scales. The two power-law exponents and the mixing parameter are of order one, whereas n reaches 10^4 , as do the components defined as fractions of the graph order. Consequently, when computed on raw values, both a squared-error loss over Θ and the Euclidean distance between two configurations are dominated by n and the components that scale with it. Each configuration is therefore mapped to normalised coordinates using a scaler σ before being passed to the model, and both model outputs are produced in these coordinates. The setup reported in Sec. 4 uses a scaler that compresses the size-related components logarithmically and maps the remaining components linearly to their admissible ranges, as defined in Eq. 23:

$$\sigma(\theta)_j = \begin{cases} \log(1 + \theta_j) / \log(1 + n_{\max}) & j = n, \\ (\theta_j - 1) / (t_{\max} - 1) & j \in \{t_1, t_2\}, \\ \theta_j & j = \xi, \\ \log(1 + \theta_j) / \log(1 + n) & j \in \{c_{\max}, d_{\max}, n_{\text{out}}\}, \\ \log(1 + \theta_j) / \log(1 + c_{\max}) & j = c_{\min}, \\ \log(1 + \theta_j) / \log(1 + d_{\max}) & j = d_{\min} \end{cases} \quad (23)$$

Here, the components of θ are indexed by parameter name, with $n_{\max} = 10^4$ and $t_{\max} = 5$. Each logarithmic coordinate is divided by its anchor, the quantity the generator forbids the component to exceed, clamped from below by one so that no denominator vanishes. The bounds are fixed in advance rather than estimated from the training split, $t_1, t_2 \geq 1$ and $\xi \in [0, 1]$ being constraints of ABCD+o and $n_{\max}, t_{\max}$ chosen ceilings exceeding the ranges of Tab. 1.

It is important to note that the map is invertible. Within the ordering penalty of the following subsection the inverse is applied as written, so that the penalty remains differentiable with respect to the model output. A configuration handed back to the generator has $n, c_{\min}, c_{\max}, d_{\min}, d_{\max}$, and n_{out} rounded to integers. Alternative scalers are compared in App. D.

A.2 Constraint-aware Configuration Loss

Let us specify the instantiation of the configuration-space discrepancy function ℓ_{Θ} (Sec. 2.2) used in DCBA. The loss combines a parameter-wise weighted squared error with soft penalties discouraging predictions that violate the admissibility constraints of the generator.

The nine-dimensional configuration of ABCD+o can be written in the order used in Tab. 1 as

$$\theta = (n, t_1, d_{\min}, d_{\max}, t_2, c_{\min}, c_{\max}, \xi, n_{\text{out}}) \in \Theta. \quad (24)$$

Before being passed to the model, each configuration vector is transformed by the scaler of App. A.1. Both the graph-derived prediction $\hat{\theta}$ and the reconstruction $\tilde{\theta}$ are produced in this scaled parameter space. For compactness, let θ' denote either of these model outputs. The discrepancy between θ' and the corresponding target θ is evaluated in the scaled space, whereas the generator-specific constraints are evaluated after both vectors have been transformed back to their original parameter scale.

The parameter-wise discrepancy is measured using a weighted mean squared error,

$$\ell_{\text{mse}}(\theta', \theta) = \frac{1}{q} \sum_{j=1}^q \omega_j (\theta'_j - \theta_j)^2, \quad (25)$$

where $q = 9$ and $\omega_j \geq 0$ denotes the weight assigned to the j -th configuration component. Setting $\omega_j = 1$ for every component recovers the ordinary mean squared error. The parameter weights are treated as hyperparameters, and the settings considered in the experiments are reported in App. D.

A range penalty discourages outputs from lying outside the expected range of the scaled parameter space. For scalers mapping the configuration components to normalised coordinates, it is defined as

$$\ell_{\text{range}}(\theta') = \sum_{j=1}^q [-\theta'_j]_+^2 + \sum_{j=2}^q [\theta'_j - 1]_+^2, \quad (26)$$

where $[x]_+ = \max(0, x)$. The first term penalises negative values of all configuration components, whereas the second penalises values exceeding the upper normalisation bound. The graph-size parameter n is exempt from the upper-range penalty, allowing the estimator to predict graphs larger than those represented in the range assumed by the scaler. When no normalisation is applied, the second sum is omitted, since raw parameter values may naturally exceed one.

The ordering penalty enforces relationships between the ABCD+o parameters in their original scale. To avoid introducing additional notation, in this context θ' denotes the model output after reversing the scaling transformation. Each constraint violation is divided by the ground-truth graph order n , making it dimensionless and reducing its dependence on graph size. The penalty is defined as

$$\begin{aligned} \ell_{\text{ord}}(\theta', \theta) = & \left[\frac{\theta'_{c_{\min}} - \theta'_{c_{\max}}}{n} \right]_+^2 + \left[\frac{\theta'_{d_{\min}} - \theta'_{d_{\max}}}{n} \right]_+^2 \\ & + \left[\frac{\theta'_{c_{\max}} - \theta'_n}{n} \right]_+^2 + \left[\frac{\theta'_{d_{\max}} - \theta'_n}{n} \right]_+^2 + \left[\frac{\theta'_{n_{\text{out}}} - \theta'_n}{n} \right]_+^2. \end{aligned} \quad (27)$$

The individual terms penalise violations of

$$c_{\min} \leq c_{\max}, \quad d_{\min} \leq d_{\max}, \quad c_{\max} \leq n, \quad d_{\max} \leq n, \quad n_{\text{out}} \leq n. \quad (28)$$

Because the positive-part operator returns zero whenever a constraint is satisfied, only violated relationships contribute to the penalty.

The complete constraint-aware configuration loss is therefore defined as

$$\ell_{\Theta}(\theta', \theta) = \ell_{\text{mse}}(\theta', \theta) + \lambda_{\text{pen}} [\ell_{\text{range}}(\theta') + \ell_{\text{ord}}(\theta', \theta)], \quad (29)$$

where θ' denotes either type of model output introduced above and $\lambda_{\text{pen}} \geq 0$ controls the contribution of the generator-feasibility penalties. The scaler and component weights were treated as experimental hyperparameters (App. D describes their performance), while λ_{pen} was set up to 1.0 in every training run.

This discrepancy function, with the same parameterisation, was used in both objectives defined in Sec. 2.2. That is, in $\mathcal{L}_{\text{reg}}$ (Eq. 10), it compares the graph-derived prediction with the corresponding generator configuration, whereas in $\mathcal{L}_{\text{rec}}$ (Eq. 11), it compares the configuration-autoencoder reconstruction with the original configuration.

A.3 Graph-encoder Auxiliary Clustering Objective

We specify below the cluster-pooling branch of the graph encoder introduced in Sec. 3.3, together with the auxiliary objective accompanying it. Consider a single graph G and let $\mathbf{H} \in \mathbb{R}^{|V| \times m}$ collect the vertex embeddings $h_v \in \mathbb{R}^m$ produced by the encoder body. A learned affine map $g: \mathbb{R}^m \rightarrow \mathbb{R}^K$ scores every vertex against K slots, and a softmax over the slot axis turns the scores into a soft assignment,

$$\mathbf{S}_{v\kappa} = \frac{\exp(g(h_v)_\kappa)}{\sum_{\kappa'=1}^K \exp(g(h_v)_{\kappa'})}, \quad \mathbf{S} \in [0, 1]^{|V| \times K}, \quad \sum_{\kappa=1}^K \mathbf{S}_{v\kappa} = 1, \quad (30)$$

so that every vertex distributes one unit of mass over the slots. The grouping is not directly supervised: even when community information is provided as a node feature, no objective requires the learned slots to reproduce the generator-defined communities.

Each slot then receives a representation and a size,

$$cl_{e,\kappa} = \sum_{v \in V} \mathbf{S}_{v\kappa} h_v \in \mathbb{R}^m, \quad cl_{s,\kappa} = \sum_{v \in V} \mathbf{S}_{v\kappa}, \quad \sum_{\kappa=1}^K cl_{s,\kappa} = |V|, \quad (31)$$

where the representation sums the vertex embeddings weighted by their membership in slot κ , and the size is the mass the slot receives, which counts its vertices once the assignment is sharp. We write $cl_s \in \mathbb{R}^K$ for the vector of all K slot sizes.

Neither the representations nor the sizes can be passed on as they stand, because the slot axis carries no identity: permuting the slots describes the same grouping, and two graphs may order them differently. What enters h_G must therefore be invariant under such permutations. We reduce the slot representations by a component-wise maximum and the slot sizes by Q of their quantiles, both of them order statistics and so invariant, and concatenate the two with the attention-pooled average,

$$h_G = \text{Proj} \left(\left[\text{ATTPOOL}(\mathbf{H}) ; \max_{\kappa \leq K} cl_{e,\kappa} ; Q_{\alpha_1}(cl_s), \dots, Q_{\alpha_Q}(cl_s) \right] \right) \in \mathbb{R}^k, \quad (32)$$

where the maximum is taken component-wise, Q_α denotes the α -quantile of the K slot sizes, read at levels $\alpha_1, \dots, \alpha_Q$, and Proj is an affine map onto the k -dimensional latent space of Sec. 2, followed by layer normalisation.

Among the three terms in Eq. 22, only $\mathcal{L}_{\text{reg}}$ and $\mathcal{L}_{\text{con}}$ depend on the soft assignments $\mathbf{S}$; the reconstruction term is confined to the configuration tower. Moreover, the assignments affect these two losses only through the component-wise maximum over the slot representations and the quantiles of the slot sizes. Neither loss directly encourages each vertex to concentrate its assignment on a single slot. Take the uniform one, $\mathbf{S}_{v\kappa} = 1/K$ for every vertex and slot. All K slot representations are then equal to $\frac{1}{K} \sum_{v \in V} h_v$, so the maximum returns a rescaled average over the vertices and duplicates what the attention branch already supplies. All K slot sizes are equal to $|V|/K$, so the quantiles coincide and between them report nothing beyond $|V|$, which is the parameter n the model has to predict in any case. The branch is then indistinguishable from a second average pool, and the distributional information it was introduced to carry would never reach h_G .

The auxiliary term supplies the missing preference, acting on the entropy of the assignment. For a graph whose assignment is $\mathbf{S}$, let

$$\eta(\mathbf{S}) = - \sum_{v \in V} \sum_{\kappa=1}^K \mathbf{S}_{v\kappa} \log \mathbf{S}_{v\kappa}, \quad (33)$$

which vanishes when every vertex commits to a single slot and is largest, $|V| \log K$, under the uniform assignment. Averaging it over every vertex of a mini-batch, whose configurations $i \in \mathcal{I}$ and replicas $r \in \mathcal{R}_i$ are indexed as in Sec. 2.2, gives

$$\mathcal{L}_{\text{aux}} = \frac{\sum_{i \in \mathcal{I}} \sum_{r \in \mathcal{R}_i} \eta(\mathbf{S}_i^{(r)})}{\sum_{i \in \mathcal{I}} \sum_{r \in \mathcal{R}_i} |V_i^{(r)}|}, \quad (34)$$

where $\mathbf{S}_i^{(r)}$ and $V_i^{(r)}$ denote the assignment matrix and the vertex set of the replica $G_i^{(r)}$, each assignment being formed from the vertices of its own graph alone. The objective of Eq. 22 accordingly gains a fourth term,

$$\mathcal{L}_{\text{total}} = \lambda_{\text{reg}} \mathcal{L}_{\text{reg}} + \lambda_{\text{rec}} \mathcal{L}_{\text{rec}} + \lambda_{\text{con}} \mathcal{L}_{\text{con}} + \lambda_{\text{aux}} \mathcal{L}_{\text{aux}}. \quad (35)$$

Minimising $\mathcal{L}_{\text{aux}}$ drives each row of $\mathbf{S}$ towards a one-hot vector, so the slot sizes become counts of vertices. It does not decide how the vertices are divided, being equally minimal for the assignment that places all of them in one slot. How the vertices are divided among the slots is instead guided indirectly by $\mathcal{L}_{\text{reg}}$ and $\mathcal{L}_{\text{con}}$, which favour groupings useful for configuration recovery and representation alignment. Since the objective does not reward spreading the vertices over all K slots either, K bounds how many groups the model may take rather than prescribing their number, and it is set from the community structure of the dataset rather than tuned. Sampling graphs from the training split and counting their generator communities yields a distribution whose 95-th percentile, taken with a safety margin of 1.2, recommends 71 slots, which we round up to the next power of two. The procedure is part of the released code, so it can be repeated for any dataset the framework is applied to. The reported configuration uses $K = 128$ slots, $Q = 15$ quantile levels spanning 0.01 to 0.99 and $\lambda_{\text{aux}} = 0.1$. Encoder variants without this branch are compared in App. D.

We additionally verify that the selected K is not a binding constraint by measuring the effective number of occupied slots, defined as the exponential of the entropy of the slot-occupancy distribution $\text{cl}_s/|V|$. Its geometric mean over the test split is 1.00 for DCBA and 1.51 for both community-aware variants, compared with $K = 128$. Thus, the learned assignments use only a small fraction of the available slot capacity, suggesting that increasing K would not provide additional useful capacity in the reported setting. The concentrated assignments also provide some intuition for the role of this branch. With nearly one-hot assignments, each occupied slot representation approaches an aggregate of the embeddings of its assigned vertices. Unlike attention pooling, whose normalisation tends to suppress information about absolute group size, such aggregation retains count-sensitive information, while the subsequent quantile and maximum operators capture variation and extremes across groups. This is consistent with the improvements observed in Tab. 5: the clearest gain is for c_{max} , which directly describes an extreme of the community-size distribution, while n_{out} is likewise an absolute-count parameter. The improvement in ξ has a different interpretation: although ξ is not count-based, it controls inter-community mixing and therefore affects how clearly the node representations can be partitioned into distinct groups.

B Dataset Generation Details

This appendix provides readers with details on how the dataset was generated and contains useful information for reproducing it or adapting the pipelines to their own needs.

Parameter Conversion and Validation. Parameters expressed as fractions of the graph size n were converted to integer-valued generator parameters according to

$$\begin{aligned} d_{\min} &= \max(1, \text{round}(f_{d_{\min}} \cdot n)), & d_{\max} &= \max(d_{\min} + 1, \text{round}(f_{d_{\max}} \cdot n)), \\ c_{\min} &= \max(1, \text{round}(f_{c_{\min}} \cdot n)), & c_{\max} &= \max(c_{\min} + 1, \min(n, \text{round}(f_{c_{\max}} \cdot n))), \\ n_{\text{out}} &= \text{round}(f_{n_{\text{out}}} \cdot n), \end{aligned}$$

where $f_{(\cdot)}$ denotes the sampled fraction. The bounds on $d_{\min}$ and $c_{\min}$ guarantee minimum value of 1, and the coupling of the upper bounds to the sampled lower bounds guarantees $d_{\max} > d_{\min}$

and $c_{\min} < c_{\max} \leq n$ by construction. Every validated and accepted configuration constitutes one *instance* in the sense of Sec. 2. Each instance is then used to generate 10 independent *replicas*, differing only in their random seed. As a safeguard, a candidate configuration is redrawn up to 3 times and discarded if no valid draw is found.

Boundary Challenge Set. The additional dataset was constructed deterministically. It enumerated all $2^9 = 512$ combinations obtained by assigning either the lower or upper boundary value to each of the nine sampled parameters. All resulting parameter combinations satisfied the initial configuration-validation procedure.

Replica Generation and Failure Handling. Replica $r \in \{1, \dots, 10\}$ of each instance was generated using the seed $s_0 + r$, where s_0 denotes the run-level seed stored in the corresponding dataset configuration. Graph generation runs were subject to predefined wall-clock limits recorded in their configuration files: 240 s for the boundary challenge set and 60 s for the final five chunks of the principal dataset. If the generation of any replica exceeded the applicable limit or otherwise failed, the corresponding instance was excluded from the released dataset. Of the 9 051 valid configurations passed to the generator for the principal dataset, 8 744 complete instances were retained. Because wall-clock failures may depend on the execution environment, the experiments reported in this study use the released datasets as fixed artefacts rather than regenerating them during model training.

Implementation and Reproducibility. The dataset-generation pipeline, dataset configurations, and data loaders are available at <https://github.com/network-science-lab/DCBA-data-set>. Generated graphs are versioned using DVC and stored remotely, with access available on request. The repository contains the complete generation pipeline, including the configuration and random seeds associated with each dataset and a common entry point for executing the generation procedure. Dataset generation was performed using Python 3.12.11 and Julia 1.12.5. The Python environment was resolved and locked using uv, with the complete dependency specification provided in `uv.lock`. Graphs were generated using `ABCDGraphGenerator.jl` v0.2.0 (<https://github.com/bkamins/ABCDGraphGenerator.jl>, tree hash 780973a).

C Algorithmic Baseline Formulation

As stated in Sec. 3.4, the algorithmic baseline is based on the naïve assumption that the components of θ can be recovered independently. Thus, rather than learning a joint embedding of graphs and configurations, the baseline directly estimates each coordinate of θ from graph statistics, so that its errors reflect the extent to which the individual configuration components can be recovered without modelling their interdependence.

The community assignment is either taken directly from the ground-truth labels attached to the graph (Base-T) or, when the ground truth is assumed to be unavailable, obtained by running the Leiden algorithm [29] on the graph (Base-L). This second mode simulates the realistic setting in which no community labels accompany the observed network and allows us to assess how sensitive the baseline is to replacing the ground-truth labels with a detected partition.

Given a graph and a community assignment, $\hat{n}$ is the number of vertices in the graph, $\hat{d}_{\min}$ and $\hat{d}_{\max}$ are set to the minimum and maximum observed vertex degrees, and $\hat{c}_{\min}$ and $\hat{c}_{\max}$ are set to the minimum and maximum observed community sizes. The power-law exponents $\hat{t}_1$ and $\hat{t}_2$ are then obtained by numerically maximising the truncated power-law log-likelihood of the degree sequence and the community-size sequence, respectively. The noise level ξ is estimated as the fraction of edges crossing community boundaries,

$$\hat{\xi} = 1 - \frac{e_{\text{intra}}}{|E|}, \quad (36)$$

where $|E|$ is the number of edges in the graph and e_{intra} is the number of edges whose endpoints belong to the same community. The number of outliers $\hat{n}_{\text{out}}$ is estimated by scoring each vertex v as

$$B(v) = 2 \left(\frac{\deg_A(v)}{\deg_G(v)} - \frac{\text{vol}(A) - \deg_G(v)}{\text{vol}(V)} \right), \quad (37)$$

where A is the community containing v , $\deg_G(v)$ denotes the degree of v in G , $\deg_A(v)$ denotes the number of its intra-community edges within A , and $\text{vol}(\cdot)$ denotes the sum of degrees over a vertex set [17]. Isolated vertices, for which $\deg(v) = 0$, are excluded rather than counted as outliers.

The resulting $\hat{\theta}$ can be directly compared with the predictions of DCBA. Owing to the stochastic nature of ABCD and the collisions and corrections occurring during graph generation, only n can be recovered exactly from every realisation. The accuracy with which the baseline recovers the remaining parameters instead depends on the particular configuration being estimated.

D Model Selection, Ablations, and Interpretability

This appendix reports the experiments used to select the model configuration evaluated in Sec. 4 and to assess its sensitivity to key design choices. These experiments examine the graph encoder (including the way the clustering signal is processed), θ -scalers, the use of ground-truth community information as node features, and the mini-batch size. They were conducted sequentially, with each subsequent experiment adopting the best-performing configuration identified in the preceding one; unless stated otherwise, only the component under investigation was varied. We additionally analyse the stability of the learned graph representations, and their sensitivity to varying parameters of ABCD. Finally, the implementation and reproducibility details are presented.

D.1 Graph Encoders

The first experiment compares candidate graph-encoder architectures and selects the variant used in the subsequent experiments. All variants employed the logarithmic scaler described in App. A.1, a mini-batch size of 32, and constant node features, $\mathbf{X} = \mathbf{1}$. The evaluated encoders are described below, and their performance is compared in Tab. 5.

The GIN Encoder. A stack of GIN layers [32], whose outputs are combined by a trainable attention-weighted sum and aggregated into h_G by attention pooling. It relies on local message passing alone and therefore serves as a reference point for what can be recovered without a graph-wide view.

The GPS Encoder. Here the GIN layers are replaced with the GraphGPS blocks of Sec. 3.3, in which local message passing is combined with global multi-head self-attention, implemented using the linear approximation of [7]. The readout remains a single attention pooling, so the graph is summarised by a weighted average of its vertex embeddings.

The GPS-weighted Encoder. The architecture is left unchanged and only the configuration loss is modified, each parameter receiving a weight $w_i = 1 + \text{clip}(R_{\text{reg}}^2 - R_{\text{cross}}^2, 0, 1)$. The weight measures how much of θ_i is lost when it is decoded from z_G rather than from z_θ , so that the gradient is directed towards the parameters recovered worst by the graph encoder rather than by the configuration autoencoder. Under this rule t_2 receives the largest weight, close to 2.

The GPS-clustered Encoder. This variant adds the soft-clustering branch of Sec. 3.3 alongside the attention pooling. Vertices are softly assigned to at most $K = 128$ learned slots, and quantiles of the resulting slot sizes are appended to h_G . The slots receive no supervision, as ground-truth communities constitute generator metadata that would be unavailable for an observed graph.

The GPS-clustered-weighted Encoder. Both extensions are applied together, that is the cluster-pooling branch trained under the weighted configuration loss. This is the encoder presented in Sec. 3.3 and used by every DCBA variant evaluated in Sec. 4.

The GIN encoder performs considerably worse than all GPS-based variants, indicating the importance of incorporating global interactions in addition to local message passing. Adding the weighted pooling branch to GPS has a mixed effect, improving several parameters but reducing the overall $\bar{R}^2$. In contrast, the soft-clustering branch provides the most consistent improvement, particularly for c_{max} , ξ , and n_{out} , and yields the highest average score. Combining both aggregation mechanisms largely preserves these gains while recovering the performance on t_1 . As only 0.01 in $\bar{R}^2$ separates GPS-clustered and GPS-clustered-weighted, we use the latter in the remaining experiments.

Table 5: Parameter-wise R^2 on the casual test set for the compared graph encoders; first- and second-best results are shown in green.

Parameter	GIN	GPS	GPS-weighted	GPS-clustered	GPS-clustered-weighted
n	0.36	0.93	0.86	0.95	0.89
t_1	0.68	0.95	0.94	0.87	0.95
$d_{\min}$	0.84	0.92	0.94	0.96	0.96
$d_{\max}$	0.01	0.74	0.76	0.78	0.78
t_2	-0.02	0.00	-0.05	-0.02	-0.02
$c_{\min}$	0.13	0.46	0.15	0.44	0.44
$c_{\max}$	0.17	0.37	0.21	0.52	0.45
ξ	0.10	0.79	0.82	0.85	0.85
n_{out}	0.10	0.15	0.13	0.27	0.24
$\overline{R^2}$	0.26 (0.31)	0.59 (0.36)	0.53 (0.41)	0.62 (0.34)	0.61 (0.35)

D.2 Representing Cluster Structure

This experiment compares different hyperparameters of the soft-clustering module presented in Sec. 3.3 and App. A.3. All variants follow the base configuration from App. D.1 and use GPS-clustered-weighted, varying only the Q and K hyperparameters. The selected variants for this comparison are described below, and their results are shown in Tab. 6 and Tab. 7.

The K Ablation. The number of learned slots in the soft-clustering pool is varied across $K \in \{64, 128, 256\}$. The model can leave some slots unused (as K is a capacity budget only), and the Q-wide readout is used in this case, i.e., the same as reported in App. A.3.

Table 6: Parameter-wise R^2 on the casual test set for varying numbers of clusters K ; first- and second-best results are shown in green.

Parameter	K-64	K-128	K-256
n	0.86	0.89	0.81
t_1	0.94	0.95	0.92
$d_{\min}$	0.95	0.96	0.94
$d_{\max}$	0.70	0.78	0.75
t_2	0.01	-0.02	0.01
$c_{\min}$	0.32	0.44	0.38
$c_{\max}$	0.33	0.45	0.29
ξ	0.82	0.85	0.85
n_{out}	0.06	0.24	0.15
$\overline{R^2}$	0.55 (0.36)	0.61 (0.35)	0.57 (0.34)

The Q-wide Readout. A dense set of 15 hand-picked quantile levels is used, covering both size extremes and the bulk of the distribution: $\{0.01, 0.05, 0.1, 0.2, 0.25, 0.3, 0.4, 0.5, 0.6, 0.7, 0.75, 0.8, 0.9, 0.95, 0.99\}$. This provides a detailed representation of the community-size distribution across the full range. The soft-clustering pool uses $K = 128$ learned slots.

The Q-0 Readout. No size-distribution information is used. The encoder keeps only the max-pooled cluster embedding, without an explicit representation of how community sizes are distributed.

The Q-coarse Readout. A coarse four-point quantile summary is used, with quantile levels $\{0.25, 0.5, 0.75, 0.99\}$. This provides a compact representation combining information from the center and upper tail of the distribution.

The Q-top Readout. Only the upper tail of the community-size distribution is represented, using quantile levels $\{0.9, 0.99, 0.999, 1\}$. This focuses on the largest communities while discarding information from the bulk and lower end of the distribution.

The Q-uniform Readout. A set of 15 quantile levels is spaced uniformly across the full range, using $\text{linspace}(0, 1, 15)$. This provides an even representation of the community-size distribution without concentrating levels in particular regions.

Table 7: Parameter-wise R^2 on the casual test set for varying cluster-size quantile sets Q ; first- and second-best results are shown in green.

Parameter	Q-wide	Q-0	Q-coarse	Q-top	Q-uniform
n	0.89	0.92	0.94	0.92	0.94
t_1	0.95	0.94	0.95	0.95	0.92
$d_{\min}$	0.96	0.94	0.95	0.95	0.93
$d_{\max}$	0.78	0.79	0.82	0.78	0.73
t_2	-0.02	-0.02	-0.02	-0.01	-0.01
$c_{\min}$	0.44	0.45	0.50	0.37	0.39
$c_{\max}$	0.45	0.38	0.56	0.37	0.41
ξ	0.85	0.85	0.85	0.81	0.75
n_{out}	0.24	0.33	0.33	0.23	0.22
$\bar{R}^2$	0.61 (0.35)	0.62 (0.33)	0.65 (0.32)	0.60 (0.34)	0.59 (0.33)

Table 6 shows a non-monotonic trend in the number of clusters K . The default budget of 128 slots achieves the highest mean R^2 (0.61), while both $K = 64$ (0.55) and $K = 256$ (0.57) underperform it consistently. This supports $K = 128$ as a well-calibrated choice. The picture for the quantile set Q presented in Tab. 7 is less straightforward. The four-point Q-coarse configuration achieves the highest mean R^2 (0.65), followed by Q-0 (0.62) and Q-wide (0.61), while Q-top (0.60) and Q-uniform (0.59) perform worse. This suggests that most of the exploitable size-distribution signal is captured by a small number of well-placed quantiles. We nonetheless retain Q-wide as the default configuration for consistency with the main experiments reported in the paper. The additional ablation shows that Q-coarse provides a strong alternative, and may be preferable in follow-up studies, while leaving the conclusions regarding the utility of preserving the clustering signal unchanged.

D.3 Configuration Scalers

The second experiment evaluates alternative methods for scaling the generator configurations. As the components of θ differ substantially in magnitude and interpretation, the choice of scaler may affect both regression and contrastive learning. We were mainly interested in maps built from the generator’s own constraints rather than from statistics fitted to the training data, which would tie a trained model to the ranges the dataset was sampled from. The compared scalers are described below. The results are reported in Tab. 8.

Table 8: Parameter-wise R^2 on the casual test set for the compared configuration scalers; first- and second-best results are shown in green.

Parameter	identity	n-max	relative	log	empirical
n	0.98	0.98	0.98	0.89	0.98
t_1	-11.72	0.78	0.87	0.95	0.90
$d_{\min}$	0.32	-1.26	-0.32	0.96	-0.20
$d_{\max}$	0.80	0.81	0.82	0.78	0.80
t_2	-6.97	-0.04	-0.04	-0.02	-0.02
$c_{\min}$	0.31	0.29	-0.03	0.44	0.13
$c_{\max}$	0.45	0.42	0.51	0.45	0.55
ξ	-0.37	0.73	0.75	0.85	0.83
n_{out}	0.39	0.36	0.38	0.24	0.33
$\bar{R}^2$	-1.76 (4.48)	0.34 (0.68)	0.44 (0.47)	0.61 (0.35)	0.48 (0.44)

The identity Scaler. No transformation is applied, so that θ is presented to the model in its raw units. As n reaches several thousand while t_1 , t_2 and ξ remain confined to narrow ranges, it illustrates the effect of leaving the differences in magnitude unaddressed.

The n-max Scaler. A single linear map per feature, with bounds anchored on the largest admissible n rather than on the sampling ranges of Tab. 1. It is the simplest of the compared transformations, at the cost of compressing the small-valued parameters into a narrow band near zero.

The relative Scaler. Every constrained feature becomes a fraction of the parameter bounding it, namely $c_{\max}/n$, $d_{\max}/n$, n_{out}/n , $c_{\min}/c_{\max}$, and $d_{\min}/d_{\max}$, while n , t_1 , t_2 , and ξ remain scaled linearly by 10000, 5, and 5, respectively. The ordering constraints are then satisfied by construction whenever the outputs lie within $[0, 1]$.

The empirical Scaler. Min-max scaling from the sampling ranges reported in Tab. 1, with the exception of $c_{\min}$ and $d_{\min}$, which are again expressed relative to their constraint partners.

The log scaler attains the best result for most parameters and by a wide margin in terms of $\overline{R^2}$, and is therefore retained in the remaining experiments. The alternatives recover n almost exactly, which is expected as all of them keep it on a linear scale, but they lose $d_{\min}$ entirely. The identity scaler illustrates the consequence of leaving the raw magnitudes untouched, as the regression becomes dominated by n while both power-law exponents deteriorate.

D.4 Node Features

As described in Sec. 3.3, the graph encoder includes a soft-clustering branch intended to capture group-level structure. We additionally evaluated whether providing ground-truth community information as node features improves model performance. To preserve invariance to permutations of community labels, each vertex was assigned the size of its ground-truth community rather than its community identifier. Table 9 compares this setting with the baseline using constant node features, $\mathbf{X} = \mathbf{1}$.

Table 9: Parameter-wise R^2 on the casual test set for constant ($\mathbf{X} = \mathbf{1}$) and community-size node features; first- and second-best results are shown in green.

Parameter	base	comm
n	0.89	0.91
t_1	0.95	0.92
$d_{\min}$	0.96	0.96
$d_{\max}$	0.78	0.78
t_2	-0.02	0.07
$c_{\min}$	0.44	0.80
$c_{\max}$	0.45	0.71
ξ	0.85	0.97
n_{out}	0.24	0.34
$\overline{R^2}$	0.61 (0.35)	0.72 (0.31)

Community sizes improve precisely the parameters they describe, raising $c_{\min}$ and $c_{\max}$ by approximately 0.3 each and ξ to 0.97, with only a slight loss on t_1 . The information comes from the ground-truth assignment rather than from a community detection algorithm, which is the reason for pairing this variant with Base-T in Sec. 4.

D.5 Batch Size

The next experiment examines whether the mini-batch size affects model performance. Following common practice in contrastive learning, we expected larger mini-batches to improve representation learning by exposing the model to more positive and negative pairs within each update. The results are reported in Tab. 10.

Doubling the mini-batch improves most parameters after 20 epochs, which was the maximum training duration used in the ablation study, with per-parameter differences ranging from -0.03 to $+0.06$ and the largest gains observed for $c_{\min}$, t_2 , and $c_{\max}$. However, after extending training to 50 epochs (Tab. 2), this advantage largely disappears. This suggests that the larger batch primarily accelerates convergence by exposing the contrastive objective to more positive and negative pairs per update, rather than yielding a consistently better final solution.

Table 10: Parameter-wise R^2 on the casual test set for the compared mini-batch sizes; first- and second-best results are shown in green.

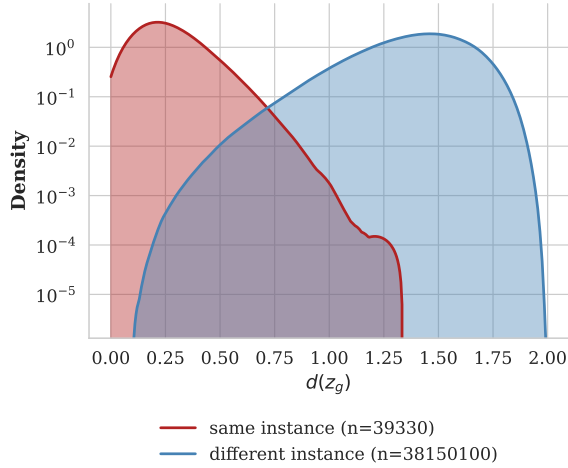
Parameter	Batch 32	Batch 64
n	0.91	0.94
t_1	0.92	0.94
$d_{\min}$	0.96	0.96
$d_{\max}$	0.78	0.74
t_2	0.07	0.13
$c_{\min}$	0.80	0.86
$c_{\max}$	0.71	0.75
ξ	0.97	0.97
n_{out}	0.34	0.33
$\overline{R^2}$	0.72 (0.31)	0.74 (0.31)

D.6 Stability of Graph Representations

Finally, we present an additional analysis of the embedding stability for the Batch 64 variant (described in Tab. 10). From the trained model, we retrieve z_G embeddings of test graphs along with their ABCD parameters θ . Then, an Euclidean distance between a subset of pairs of embeddings is calculated:

$$d(z_G) = d(z_{G,i}, z_{G,j}) = \|z_i - z_j\|_2. \quad (38)$$

Pairs are split into *same-instance* replicas ($\theta_i = \theta_j$) and *cross-instance* ($\theta_i \neq \theta_j$) sets. Figure 2 shows that the same-instance distance distribution is concentrated at small distances, consistent with embedding stability across replicas of the same underlying instance. The cross-instance distribution is shifted towards larger distances and is considerably more diffuse, reflecting greater embedding dissimilarity between distinct instances.

**Figure 2:** Distribution of pairwise z_G embedding distances for same-instance vs. cross-instance replica pairs, log-scaled density.

To check the stability of embeddings’ geometry with respect to each θ parameter, we project z_G embeddings from $\mathbb{R}^{64}$ to $\mathbb{R}^2$ space with the t-SNE algorithm. Colouring by the raw values of an ABCD parameter provides guidance as to whether the learned space visibly organises along a parameter. Figure 3 presents a good and bad fit for two selected parameters. Values of the community sizes exponent t_2 are scrambled within the low-dimensional space, suggesting weak performance of the model in t_2 retrieval (see Tab. 2). In contrast, high values of ξ are well concentrated in one area, pointing to higher performance for this parameter.

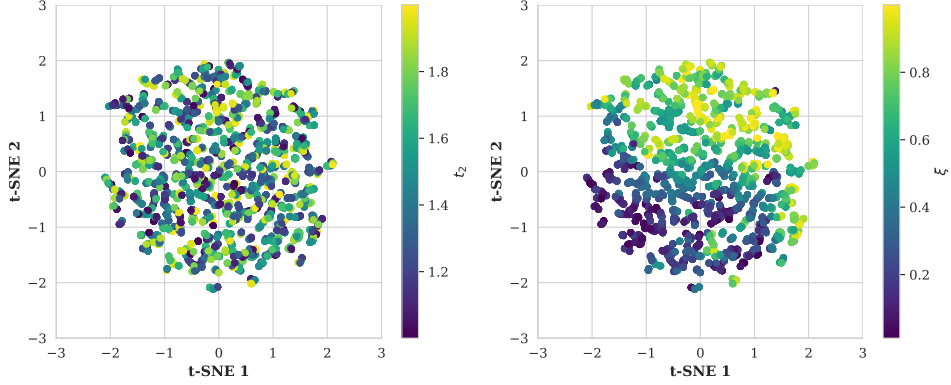


Figure 3: $\mathbb{R}^2$ representation of z_G with t_2 (left) and ξ (right) colour overlay.

D.7 Latent-space Sensitivity to Generator Parameters

To gain further understanding of the relationship between ABCD’s parameters and the learned space of graph embeddings obtained for the Batch 32 variant from Tab. 10, we performed a distance metric learning task. The goal of the experiment is to better understand the inner workings of DCBA in the spirit of eXplainable AI. Although the results are not directly related to model quality, they can shed light on the importance of the synthetic graph generator’s parameters. Let M_G and M_θ be the Mahalanobis [22] matrices defining the distances:

$$d_G(z_{G,i}, z_{G,j}) = \sqrt{(z_{G,i} - z_{G,j})^\top M_G (z_{G,i} - z_{G,j})}, \quad d_\theta(\theta_i, \theta_j) = \sqrt{(\theta_i - \theta_j)^\top M_\theta (\theta_i - \theta_j)} \quad (39)$$

where $z_{G,i}$ is the i -th normalised graph embedding and θ_i is the min-max scaled vector of the i -th graph’s ABCD parameters. We fitted M_G and M_θ to minimise the pairwise difference between distances in the two spaces using a scale-invariant MSE loss. Intuitively, we seek linear transformations, defined by M_G and M_θ , such that if two points are close in the embedding space, they are also close in the ABCD parameter space. Then, via eigendecomposition of M_θ , we can infer the sensitivity of the network’s learned geometry to individual ABCD parameters.

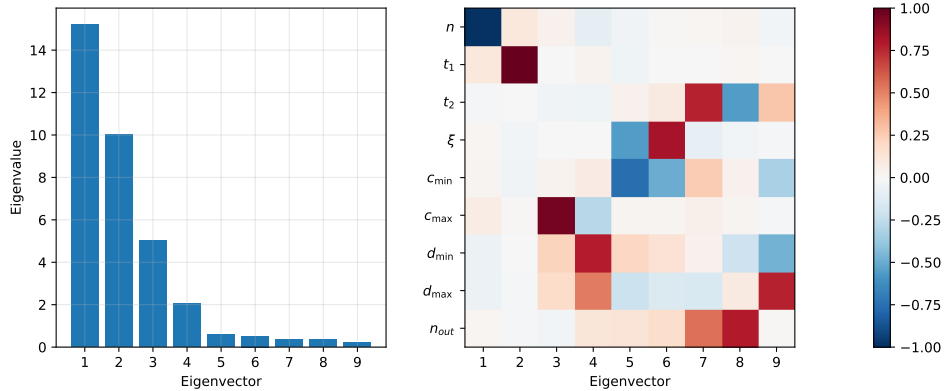


Figure 4: Eigenvalue spectrum (left) and per-parameter eigenvector loadings (right) of M_θ matrix.

With the optimal Mahalanobis matrices, distances in the two spaces on held-out data correlate strongly (Pearson $r = 0.867$), indicating that embedding-space proximity reflects closeness in the underlying generative parameters under a linear-transformation regime. Figure 4 shows the eigenvalue spectrum concentrated in three components – n , t_1 , and $c_{\max}$ – indicating their importance in shaping the embedding manifold. This suggests the model can capture diverse topological features, i.e., graph size (n), degree structure (t_1), and a community-related characteristic ($c_{\max}$). Graph size, a basic

characteristic, is expected to have a profound impact on the discriminative capabilities of the inverse model, but the vital role of t_1 and $c_{\max}$ is not as easily anticipated.

D.8 Implementation and Reproducibility

The source code of the training pipeline is available at <https://github.com/network-science-lab/DCBA-trainer>. The implementation was developed in Python 3.12 using PyTorch 2.5.1. All DCBA variants were optimised using AdamW (learning rate 0.001, weight decay 0.00001) and trained with a fixed random seed. The experiments presented in the main part of the manuscript were run for at most 50 epochs with an early-stopping patience of 10, whereas the ablations reported in the appendices were run for at most 20 epochs with a patience of 3. The former required on average 27.4 epochs and the latter 12.6 epochs, with one epoch on the “casual” dataset taking on average 4.1 hours. The experiments were conducted on two machines, both equipped with an Intel Xeon Gold 6330 CPU and 64 GB of RAM. One featured an NVIDIA TITAN RTX GPU, while the other, used for the experiments with batch size 64, with an NVIDIA A40 GPU.

E Author Contributions

Piotr Bródka: Methodology; Resources; Writing – Review & Editing; Supervision; Funding acquisition. **Michał Czuba:** Conceptualisation; Methodology; Software; Validation; Formal analysis; Investigation; Data Curation; Writing – Original Draft; Writing – Review & Editing; Visualisation; Project Administration. **Bogumił Kamiński:** Methodology; Supervision; Funding acquisition. **Łukasz Kraiński:** Methodology; Software; Validation; Formal analysis; Investigation; Data Curation; Writing – Original Draft; Writing – Review & Editing; Visualisation. **Katarzyna Musiał:** Supervision. **Paweł Prałat:** Supervision. **Mateusz Stolarski:** Methodology; Software; Validation; Formal analysis; Investigation; Data Curation; Writing – Original Draft; Writing – Review & Editing; Visualisation.²

AI Disclosure. In this work, generative AI tools were used for spelling corrections in the manuscript and to facilitate source-code development at the level of implementing the authors’ ideas. The authors take full responsibility for the final content of this work. No creative work was commissioned to AI.

²The statement has been prepared in accordance with the CRediT taxonomy (<https://www.elsevier.com/researcher/author/policies-and-guidelines/credit-author-statement>).