A Graph is Worth K Words: Euclideanizing Graph using Pure Transformer

Zhangyang Gao [*1 2] **Daize Dong** [*1] **Cheng Tan** [1 2] **Jun Xia** [1 2] **Bozhen Hu** [1 2] **Stan Z. Li** [† 1]

Abstract

Can we model Non-Euclidean graphs as pure language or even Euclidean vectors while retaining their inherent information? The Non-Euclidean property have posed a long term challenge in graph modeling. Despite recent graph neural networks and graph transformers efforts encoding graphs as Euclidean vectors, recovering the original graph from vectors remains a challenge. In this paper, we introduce GraphsGPT, featuring an Graph2Seq encoder that transforms Non-Euclidean graphs into learnable Graph Words in the Euclidean space, along with a GraphGPT decoder that reconstructs the original graph from Graph Words to ensure information equivalence. We pretrain GraphsGPT on 100M molecules and yield some interesting findings: (1) The pretrained Graph2Seq excels in graph representation learning, achieving state-of-the-art results on $8/9$ graph classification and regression tasks. (2) The pretrained GraphGPT serves as a strong graph generator, demonstrated by its strong ability to perform both few-shot and conditional graph generation. (3) Graph2Seq+GraphGPT enables effective graph mixup in the Euclidean space, overcoming previously known Non-Euclidean challenges. (4) The edge-centric pretraining framework GraphsGPT demonstrates its efficacy in graph domain tasks, excelling in both representation and generation. Code is available at GitHub.

1. Introduction

Graphs, inherent to Non-Euclidean data, are extensively applied in scientific fields such as molecular design, social network analysis, recommendation systems, and meshed 3D surfaces (Shakibajahromi et al., 2024; Zhou et al., 2020a; Huang et al., 2022; Tan et al., 2023; Li et al., 2023a; Liu et al., 2023a; Xia et al., 2022b;b; Gao et al., 2022). The Non-Euclidean nature of graphs has inspired sophisticated model designs, including graph neural networks (Kipf & Welling, 2016a; Veličković et al., 2017) and graph transformers (Ying et al., 2021; Min et al., 2022). These models excel in encoding graph structures through attention maps. However, the structural encoding strategies limit the usage of auto-regressive mechanism, thereby hindering pure transformer from revolutionizing graph fields, akin to the success of Vision Transformers (ViT) (Dosovitskiy et al., 2020) in computer vision. We employ pure transformer for graph modeling and address the following open questions: ***(1)*** *How to eliminate the Non-Euclidean nature to facilitate graph representation?* ***(2)*** *How to generate Non-Euclidean graphs from Euclidean representations?* ***(3)*** *Could the combination of graph representation and generation framework benefits from self-supervised pretraining?*

We present Graph2Seq, a pure transformer encoder designed to compress the Non-Euclidean graph into a sequence of learnable tokens called Graph Words in a Euclidean form, where all nodes and edges serve as the inputs and undergo an initial transformation to form Graph Words. Different from graph transformers (Ying et al., 2021), our approach doesn't necessitate explicit encoding of the adjacency matrix and edge features in the attention map. Unlike TokenGT (Kim et al., 2022), we introduce a Codebook featuring learnable vectors for graph position encoding, leading to improved training stability and accelerated convergence. In addition, we employ a random shuffle of the position Codebook, implicitly augmenting different input orders for the same graph, and offering each position vector the same opportunity of optimization to generalize to larger graphs.

We introduce GraphGPT, a groundbreaking GPT-style transformer model for graph generation. To recover the Non-Euclidean graph structure, we propose an edge-centric generation strategy that utilizes block-wise causal attention to sequentially generate the graph. Contrary to previous methods (Hu et al., 2020a; Shi et al., 2019; Peng et al., 2022) that generate nodes before predicting edges, the edge-centric technique jointly generates edges and their corresponding endpoint nodes, greatly simplifying the generative space. To align graph generation with language generation, we implement auto-regressive generation using block-wise causal attention, which enables the effective translation of Euclidean representations into Non-Euclidean graph structures.

[*]Equal contribution [1]Westlake University, Hangzhou, China [2]Zhejiang University, Hangzhou, China. Correspondence to: Stan Z. Li <Stan.ZQ.Li@westlake.edu.cn>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

Leveraging Graph2Seq encoder and GraphGPT decoder, we present GraphsGPT, an integrated end-to-end framework. This framework facilitates a natural self-supervised task to optimize the representation and generation tasks, enabling the transformation between Non-Euclidean and Euclidean data structures. We pretrain GraphsGPT on 100M molecule graphs and comprehensively evaluate it from three perspectives: Encoder, Decoder, and Encoder-Decoder. The pretrained Graph2Seq encoder is a strong graph learner for property prediction, outperforming baselines of sophisticated methodologies on $8/9$ molecular classification and regression tasks. The pretrained GraphGPT decoder serves as a powerful structure prior, showcasing both few-shot and conditional generation capabilities. The GraphsGPT framework seamlessly connects the Non-Euclidean graph space to the Euclidean vector space while preserving information, facilitating tasks that are known to be challenging in the original graph space, such as graph mixup. The good performance of pretrained GraphsGPT demonstrates that our edge-centric GPT-style pretraining task offers a simple yet powerful solution for graph learning. In summary, we tame pure transformer to convert Non-Euclidean graph into K learnable Graph Words , showing the capabilities of Graph2Seq encoder and GraphGPT decoder pretrained through self-supervised tasks, while also paving the way for various Non-Euclidean challenges like graph manipulation and graph mixing in Euclidean latent space.

2. Related Work

Graph2Vec. Graph2Vec methods create the graph embedding by aggregating node embeddings via graph pooling (Lee et al., 2019; Ma et al., 2019; Diehl, 2019; Ying et al., 2018). The node embeddings could be learned by either traditional algorithms (Ahmed et al., 2013; Grover & Leskovec, 2016; Perozzi et al., 2014; Kipf & Welling, 2016b; Chanpuriya & Musco, 2020; Xiao et al., 2020), or deep learning based graph neural networks (GNNs) (Kipf & Welling, 2016a; Hamilton et al., 2017; Wu et al., 2019; Chiang et al., 2019; Chen et al., 2018; Xu et al., 2018), and graph transformers (Ying et al., 2021; Hu et al., 2020c; Dwivedi & Bresson, 2020; Rampášek et al., 2022; Chen et al., 2022). These methods are usually designed for specific downstream tasks and can not be used for general pretraining.

Graph Transformers. The success of extending transformer architectures from natural language processing (NLP) to computer vision (CV) has inspired recent works to apply transformer models in the field of graph learning (Ying et al., 2021; Hu et al., 2020c; Dwivedi & Bresson, 2020; Rampášek et al., 2022; Chen et al., 2022; Wu et al., 2021b; Kreuzer et al., 2021; Min et al., 2022). To encode the graph prior, these approaches introduce structure-inspired position embeddings and attention mechanisms. For instance, Dwivedi & Bresson (2020); Hussain et al. (2021) adopt Laplacian eigenvectors and SVD vectors of the adjacency matrix as position encoding vectors. Dwivedi & Bresson (2020); Mialon et al. (2021); Ying et al. (2021); Zhao et al. (2021) enhance the attention computation based on the adjacency matrix. Recently, Kim et al. (2022) introduced a decoupled position encoding method that empowers the pure transformer as strong graph learner without the needs of expensive computation of eigenvectors and modifications on the attention computation.

Graph Self-Supervised Learning. The exploration of self-supervised pretext tasks for learning expressive graph representations has garnered significant research interest (Wu et al., 2021a; Liu et al., 2022; 2021c; Xie et al., 2022). Contrastive (You et al., 2020; Zeng & Xie, 2021; Qiu et al., 2020; Zhu et al., 2020; 2021; Peng et al., 2020b; Liu et al., 2023c;b; Lin et al., 2022; Xia et al., 2022a; Zou et al., 2022) and predictive (Peng et al., 2020a; Jin et al., 2020; Hou et al., 2022; Tian et al., 2023; Hwang et al., 2020; Wang et al., 2021) objectives have been extensively explored, leveraging strategies from the fields of NLP and CV. However, the discussion around generative pretext tasks (Hu et al., 2020a; Zhang et al., 2021) for graphs is limited, particularly due to the Non-Euclidean nature of graph data, which has led to few instances of pure transformer utilization in graph generation. This paper introduces an innovative approach by framing graph generation as analogous to language generation, thus enabling the use of a pure transformer to generate graphs as a novel self-supervised pretext task.

Motivation. The pure transformer has revolutionized the modeling of texts (Devlin et al., 2018; Brown et al., 2020; Achiam et al., 2023), images (Dosovitskiy et al., 2020; Alayrac et al., 2022; Dehghani et al., 2023; Liu et al., 2021d), and the point cloud (Li et al., 2023b; Yu et al., 2022; Pang et al., 2022) in both representation and generation tasks. However, due to the Non-Euclidean nature, extending transformers to graphs typically necessitates the explicit incorporation of structural information into the attention computation. Such constraint results in following challenges:

1. **Generation Challenge.** When generating new nodes or bonds, the undergone graph structure changes, resulting in a complete update of all graph embeddings from scratch for full attention mechanisms. Moreover, an additional link predictor is required to predict potential edges from a $|\mathcal{V}| \times |\mathcal{V}|$ search space.

2. **Non-Euclidean Challenge.** Previous methods do not provide Euclidean prototypes to fully describe graphs. The inherent Non-Euclidean nature poses challenges for tasks like graph manipulation and mixing.

3. **Representation Challenge.** Limited by the generation challenge, traditional graph self-supervised learning methods have typically focused on reconstructing corrupted sub-features and sub-structures. They overlook

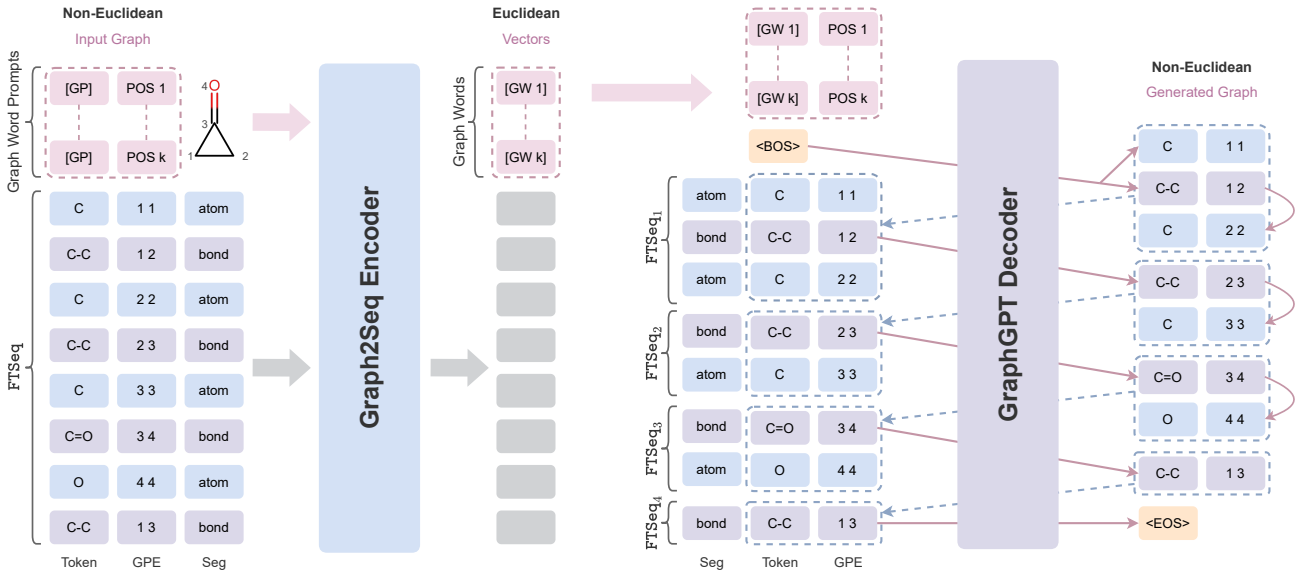


Figure 1: The Overall framework of GraphsGPT. Graph2Seq encoder transforms the Non-Euclidean graph into Euclidean Graph Words, which are further fed into GraphGPT decoder to auto-regressively generate the original Non-Euclidean graph. Both Graph2Seq and GraphGPT employ pure transformer as the structure.

of learning from the entire graph potentially limits the ability to capture the global topology.

To tackle these challenges, we propose GraphsGPT, which uses pure transformer to convert the Non-Euclidean graph into a sequence of Euclidean vectors (Graph2Seq) while ensuring informative equivalence (GraphGPT). For the first time, we bridge the gap between graph and sequence modeling in both representation and generation tasks.

3. Method

3.1. Overall Framework

Figure 1 outlines the comprehensive architecture of **GraphsGPT**, which consists of a **Graph2Seq** encoder and a **GraphGPT** decoder. The Graph2Seq converts Non-Euclidean graphs into a series of learnable feature vectors, named Graph Words. Following this, the GraphGPT utilizes these Graph Words to auto-regressively reconstruct the original Non-Euclidean graph. Both components, the Graph2Seq and GraphGPT, incorporate the pure transformer structure and are pretrained via a GPT-style pretext task.

3.2. Graph2Seq Encoder

Flexible Token Sequence (`FTSeq`). Denote $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as the input graph, where $\mathcal{V} = \{v_1, \cdots, v_n\}$ and $\mathcal{E} = \{e_1, \cdots, e_{n'}\}$ are sets of nodes and edges associated with features $\mathbf{X}^{\mathcal{V}} \in \mathbb{R}^{n,C}$ and $\mathbf{X}^{\mathcal{E}} \in \mathbb{R}^{n',C}$, respectively. With a slight abuse of notation, we use e_i^l and e_i^r to represent the left and right endpoint nodes of edge e_i. For example, we have $e_1 = (e_1^l, e_1^r) = (v_1, v_2)$ in Figure 2. Inspired by (Kim et al., 2022), we flatten the nodes and edges in a graph into a Flexible Token Sequence (`FTSeq`) consisting of:

1. **Graph Tokens**. The stacked node and edge features are represented by $\mathbf{X} = [\mathbf{X}^{\mathcal{V}}; \mathbf{X}^{\mathcal{E}}] \in \mathbb{R}^{n+n',C}$. We utilize a token Codebook $\mathcal{B}_t$ to generate node and edge features, incorporating $118+92$ learnable vectors. Specifically, we consider the atom type and bond type, deferring the exploration of other properties, such as the electric charge and chirality, for simplicity.

2. **Graph Position Encodings (GPE)**. The graph structure is implicitly encoded through decoupled position encodings, utilizing a position Codebook $\mathcal{B}_p$ comprising m learnable embeddings $\{\boldsymbol{o}_1, \boldsymbol{o}_2, \cdots, \boldsymbol{o}_m\} \in \mathbb{R}^{m,d_p}$. The position encodings of node v_i and edge e_i are expressed as $\boldsymbol{g}_{v_i} = [\boldsymbol{o}_{v_i}, \boldsymbol{o}_{v_i}]$ and $\boldsymbol{g}_{e_i} = [\boldsymbol{o}_{e_i^l}, \boldsymbol{o}_{e_i^r}]$, respectively. Notably, $\boldsymbol{g}_{v_i}^l = \boldsymbol{g}_{v_i}^r = \boldsymbol{o}_{v_i}$, $\boldsymbol{g}_{e_i}^l = \boldsymbol{o}_{e_i^l}$, and $\boldsymbol{g}_{e_i}^r = \boldsymbol{o}_{e_i^r}$. To learn permutation-invariant features and generalize to larger, unseen graphs, we ***randomly shuffle*** the position Codebook, giving each vector an equal optimization opportunity.

3. **Segment Encodings (Seg)**. We introduce two learnable segment tokens, namely `[node]` and `[edge]`, to designate the token types within the `FTSeq`.

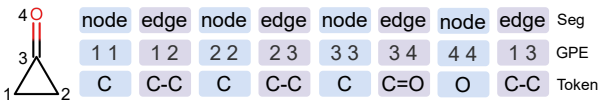


Figure 2: Graph to Flexible Sequence.

Algorithm 1 Construction of Flexible Token Sequence

Require: Canonical SMILES `CS`.
Ensure: Flexible Token Sequence `FTSeq`.
Convert canonical SMILES `CS` to graph $\mathcal{G}$.
Get the first node v_1 in graph $\mathcal{G}$ by `CS`.
Initialize sequence $\texttt{FTSeq} = [v_1]$.
for e_i in $\mathrm{DFS}(\mathcal{G}, v_1)$ **do**
  Update sequence $\texttt{FTSeq} \leftarrow [\texttt{FTSeq},\ e_i]$.
  if e_i^r not in `FTSeq` **then**
    Update sequence $\texttt{FTSeq} \leftarrow [\texttt{FTSeq},\ e_i^r]$.
  end if
end for

As depicted in Figure 2, we utilize the Depth-First Search (DFS) algorithm to convert a graph into a flexible token sequence, denoted as $\texttt{FTSeq} = [v_1, e_1, v_2, e_2, v_3, e_3, v_4, e_4]$, where the starting atom matches that in the canonical SMILES. Algorithm 1 provides a detailed explanation of our approach. It is crucial to emphasize that the resulting `FTSeq` remains Non-Euclidean data, as the number of nodes and edges may vary across different graphs.

Euclidean Graph Words. Is there a Euclidean representation that can completely describe the Non-Euclidean graph? Given the `FTSeq` and k graph prompts $[\texttt{[GP]}_1, \texttt{[GP]}_2, \cdots, \texttt{[GP]}_k]$, we use pure transformer to learn a set of Graph Words $\mathcal{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_k]$:

$$\mathcal{W} = \mathrm{Graph2Seq}([\texttt{[GP]}_1, \texttt{[GP]}_2, \cdots, \texttt{[GP]}_k, \texttt{FTSeq}]), \tag{1}$$

The token $\texttt{[GP]}_k$ is the sum of a learnable `[GP]` token and the k-th position encoding. The learned Graph Words $\mathcal{W}$ are ordered and of fixed length, analogous to a novel graph language created in the latent Euclidean space.

Graph Vocabulary. In the context of a molecular system, the complete graph vocabulary for molecules encompasses:

1. The Graph Word prompts `[GP]`;
2. Special tokens, including the begin-of-sequence token `[BOS]`, the end-of-sequence token `[EOS]`, and the padding token `[PAD]`;
3. The dictionary set of atom tokens $\mathcal{D}_v$ with a size of $|\mathcal{D}_v| = 118$, where the order of atoms is arranged by their atomic numbers, e.g., $\mathcal{D}_6$ is the atom C;
4. The dictionary set of bond tokens $\mathcal{D}_e$ with a size of $|\mathcal{D}_e| = 92$, considering the endpoint atom types, e.g., C-C and C-O are different types of bonds even though they are both single bonds.

3.3. GraphGPT Decoder

How to ensure that the learned Graph Words are information-equivalent to the original Non-Euclidean graph? Previous graph self-supervised learning methods focused on subgraph generation and multi-view contrasting, which suffer potential information loss due to insufficient capture of the global graph topology. In comparison, we adopt a GPT-style decoder to auto-regressively generate the whole graph from the learned Graph Words in a edge-centric manner.

GraphGPT Formulation. Given the learned Graph Words $\mathcal{W}$ and the flexible token sequence `FTSeq`, the complete data sequence is $[\mathcal{W}, \texttt{[BOS]}, \texttt{FTSeq}] = [\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_k, \texttt{[BOS]}, v_1, e_1, v_2, \cdots, e_i]$. We define $\texttt{FTSeq}_{1:i}$ as the sub-sequence comprising edges with connected nodes up to e_i:

$$\texttt{FTSeq}_{1:i} = \begin{cases} [v_1, e_1, \cdots, e_i, e_i^r], & \text{if } e_i^r \text{ is a new node} \\ [v_1, e_1, \cdots, e_i], & \text{otherwise} \end{cases}. \tag{2}$$

In an edge-centric perspective, we assert e_i^r belongs to e_i. If e_i^r is a new node, it will be put after e_i. Employing GraphGPT, we auto-regressively generate the complete `FTSeq` conditioned on $\mathcal{W}$:

$$\texttt{FTSeq}_{1:i+1} \xleftarrow{\texttt{FTSeq}_{1:i}} \mathrm{GraphGPT}([\mathcal{W}, \texttt{[BOS]}, \texttt{FTSeq}_{1:i}]), \tag{3}$$

where the notation above the left arrow signifies that the output $\texttt{FTSeq}_{1:i+1}$ corresponds to $\texttt{FTSeq}_{1:i}$.

Edge-Centric Graph Generation. Nodes and edges are the basic components of a graph. Traditional node-centric graph generation methods divide the problem into two parts:

(1) *Node Generation*; (2) *Link Prediction*.

We argue that node-centric approaches lead to imbalanced difficulties in generating new nodes and edges. For the molecular generation, let $|\mathcal{D}_v|$ and $|\mathcal{D}_e|$ denote the number of node and edge types, respectively. Also, let n and n' represent the number of nodes and edges. The stepwise classification complexities for predicting the new node and edge are $\mathcal{O}(|\mathcal{D}_v|)$ and $\mathcal{O}(n \times |\mathcal{D}_e|)$, respectively. Notably, we observe that $\mathcal{O}(n \times |\mathcal{D}_e|) \gg \mathcal{O}(|\mathcal{D}_v|)$, indicating a pronounced imbalance in the difficulties of generating nodes and edges. Considering that $\mathcal{O}(|\mathcal{D}_v|)$ and $\mathcal{O}(|\mathcal{D}_e|)$ are constants, the overall complexity of node-centric graph generation is $\mathcal{O}(n + n^2)$.

These approaches ignore the basic truism that naturally occurring and chemically valid bonds are sparse: there are only 92 different bonds (considering the endpoints) among 870M molecules in the ZINC database (Irwin & Shoichet, 2005). Given such an observation, we propose the edge-centric generation strategy that decouples the graph generation into:

(1) *Edge Generation*;
(2) *Left Node Attachment*; (3) *Right Node Placement*.

We provide a brief illustration of the three steps in Figure 3. The step-wise classification complexity of generating an edge is $\mathcal{O}(|\mathcal{D}_e|)$. Once the edge is obtained, the model automatically infers the left node attachment and right node placement, relieving the generation from the additional burden of generating atom types and edge connections, resulting in a reduced complexity of $\mathcal{O}(1)$. With edge-centric generation, we balance the classification complexities of predicting nodes and edge as constants. Notably, the overall generation complexity is reduced to $\mathcal{O}(n + n')$.

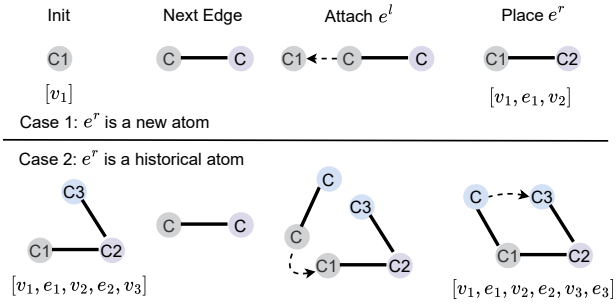


Figure 3: Overview of edge-centric graph generation.

Next, we introduce the edge-centric generation in detail.

Step 0: First Node Initialization. The first node token of FTSeq is generated by:

$$\begin{cases} \boldsymbol{h}_{v_1} \xleftarrow{\texttt{[BOS]}} \text{GraphGPT}([\mathcal{W}, \texttt{[BOS]}]) \\ \boldsymbol{p}_{v_1} = \text{Pred}_v(\boldsymbol{h}_{v_1}) \\ v_1 = \arg\max \boldsymbol{p}_{v_1} \quad \text{Node Type} \\ \boldsymbol{g}_{v_1} = [\boldsymbol{o}_1, \boldsymbol{o}_1] \quad \text{GPE} \end{cases}. \tag{4}$$

Here, $\text{Pred}_v(\cdot)$ denotes a linear layer employed for the initial node generation, producing a predictive probability vector $\boldsymbol{p}_{v_1} \in \mathbb{R}^{|\mathcal{D}_v|}$. The output v_1 corresponds to the predicted node type, and $\boldsymbol{o}_1$ represents the node position encoding retrieved from the position Codebook $\mathcal{B}'_p$ of the decoder, where we should explicitly note that the encoder Codebook $\mathcal{B}_p$ and the decoder Codebook $\mathcal{B}'_p$ are not shared.

Step 1: Next Edge Generation. The edge-centric graph generation method creates the next edge by:

$$\begin{cases} \boldsymbol{h}_{e_{i+1}} \xleftarrow{e_i} \text{GraphGPT}([\mathcal{W}, \texttt{[BOS]}, \texttt{FTSeq}_{1:i}]) \\ \boldsymbol{p}_{e_{i+1}} = \text{Pred}_e(\boldsymbol{h}_{e_{i+1}}) \\ e_{i+1} = \arg\max \boldsymbol{p}_{e_{i+1}} \quad \text{Edge Type} \end{cases}, \tag{5}$$

where Pred_e is a linear layer for the next edge prediction, and $\boldsymbol{p}_{e_{i+1}} \in \mathbb{R}^{|\mathcal{D}_e|+1}$ is the predictive probability. e_{i+1} belongs to the set $\mathcal{D}_e \cup \{\texttt{[EOS]}\}$, and the generation process will stop if $e_{i+1} = \texttt{[EOS]}$. Note that the edge position encoding $[\boldsymbol{o}_{e^l_{i+1}}, \boldsymbol{o}_{e^r_{i+1}}]$ remains undetermined. This information will affect the connection of the generated edge to the existing graph, as well as the determination of new atoms, i.e., left atom attachment and right atom placement.

Training Token Generation. The first node and next edge prediction tasks are optimized by the cross entropy loss:

$$\mathcal{L}_{\texttt{token}} = -\sum_i y_i \cdot \log p_i. \tag{6}$$

Step 2: Left Node Attachment. For the newly predicted edge e_{i+1}, we further determine how it connects to existing nodes. According to the principles of FTSeq construction, it is required that at least one endpoint of e_{i+1} connects to existing atoms, namely the left atom e^l_{i+1}. Given the set of previously generated atoms $\{v_1, v_2, \cdots, v_j\}$ and their corresponding graph position encodings $\mathbf{O}_j = [\boldsymbol{o}_{v_1}, \boldsymbol{o}_{v_2}, \cdots, \boldsymbol{o}_{v_j}] \in \mathbb{R}^{j,C}$ in $\mathcal{B}'_p$, we predict the position encoding of the left node using a linear layer $\text{PredPos}^l(\cdot)$:

$$\hat{\boldsymbol{g}}^l_{e_{i+1}} = \text{PredPos}^l(\boldsymbol{h}_{e_{i+1}}) \in \mathbb{R}^{1,C}. \tag{7}$$

We compute the cosine similarity between $\hat{\boldsymbol{g}}^l_{e_{i+1}}$ and $\mathbf{O}_j$ by $\boldsymbol{c}^l = \hat{\boldsymbol{g}}^l_{e_{i+1}} \mathbf{O}_j^T \in \mathbb{R}^t$. The index of existing atoms that e^l_{i+1} will attach to is $u_l = \arg\max \boldsymbol{c}^l$. This process implicitly infers edge connections by querying over existing atoms, instead of generating all potential edges from scratch. We update the graph position encoding of the left node as:

$$\boldsymbol{g}^l_{e_{i+1}} = \boldsymbol{o}_{v_{u_l}} \quad \text{Left Node GPE}. \tag{8}$$

Step 3: Right Node Placement. As for the right node e^r_{i+1}, we consider two cases: (1) it connects to one of the existing atoms; (2) it is a new atom. Similar to the step 2, we use a linear layer $\text{PredPos}^r(\cdot)$ to predict the position encoding of the right node:

$$\hat{\boldsymbol{g}}^r_{e_{i+1}} = \text{PredPos}^r(\boldsymbol{h}_{e_{i+1}}) \in \mathbb{R}^{1,C}. \tag{9}$$

We get the cosine similarity score $\boldsymbol{c}^r = \hat{\boldsymbol{g}}^r_{e_{i+1}} \mathbf{O}_j^T$ and the index of node with the highest similarity $u_r = \arg\max \boldsymbol{c}^r$. Given a predefined threshold ϵ, if $\boldsymbol{c}_k > \epsilon$, we consider e_{i+1} is connected to v_{u_r}, and update:

$$\boldsymbol{g}^r_{e_{i+1}} = \boldsymbol{o}_{v_{u_r}} \quad \text{Right Node GPE, Case 1;} \tag{10}$$

otherwise, e^r_{i+1} is a new atom v_{j+1}, and we set:

$$\boldsymbol{g}^r_{e_{i+1}} = \boldsymbol{o}_{j+1} \quad \text{Right Node GPE, Case 2.} \tag{11}$$

Finally, we update the FTSeq by:

$$\begin{cases} \texttt{FTSeq} \leftarrow [\texttt{FTSeq}, e_{i+1}] \quad \text{Case 1} \\ \texttt{FTSeq} \leftarrow [\texttt{FTSeq}, e_{i+1}, v_{j+1}] \quad \text{Case 2} \end{cases}. \tag{12}$$

By default, we set $\epsilon = 0.5$.

Training Node Attachment & Placement. We adopt a contrastive objective to optimize left node attachment and right node placement problems. Taking left node attachment as an example, given the ground truth t, i.e., the index of the attached atom in the original graph, the positive score is $s^+ = e^l_{i+1} \boldsymbol{o}^T_{v_t}$, while the negative scores are $\boldsymbol{s}^- = |\text{vec}(\mathbf{O}\mathbf{O}^T)| \in \mathbb{R}^{|\mathcal{B}'_p| \times (|\mathcal{B}'_p|-1)}$, where $\text{vec}(\cdot)$ is a flatten operation while ignoring the diagonal elements. The final contrastive loss is:

$$\mathcal{L}_{\texttt{attach}} = (1 - s^+) + \frac{1}{|\mathcal{B}'_p| \times (|\mathcal{B}'_p| - 1)} \sum s^-. \quad (13)$$

Block-Wise Causal Attention. In our method, node generation is closely entangled with edge generation. Specifically, on its initial occurrence, each node is connected to an edge, creating what we term a block. From the block view, we employ a causal mask for auto-regressive generation. However, within each block, we utilize the full attention. We show the block-wise causal attention in Figure 4.

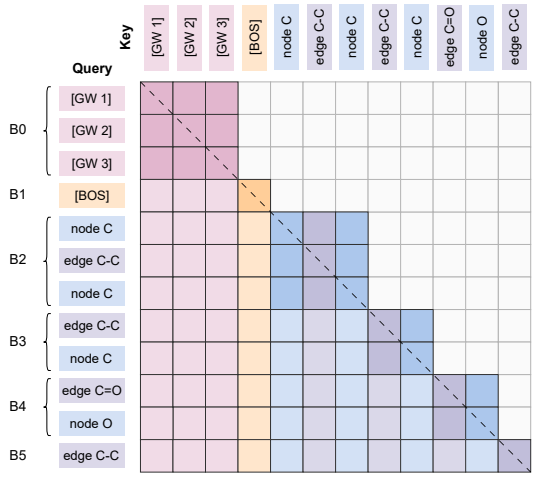


Figure 4: Block-Wise causal attention with grey cells indicating masked positions. Graph Words contribute to the generation through full attention, serving as prefix prompts.

4. Experiments

4.1. Experiment Settings

We extensively conduct experiments to assess GraphsGPT, delving into the following questions:

- **Representation (Q1):** Can Graph2Seq effectively learn expressive graph representation through pretraining?
- **Generation (Q2):** Could pretrained GraphGPT serve as a strong structural prior model for graph generation?
- **Euclidean Graph Words (Q3):** What opportunities do the Euclidean Graph Words offer that were previously considered challenging?

4.2. Datasets

ZINC (Pretraining). To pretrain GraphsGPT , we select the ZINC database (Irwin & Shoichet, 2005) as our pretraining dataset, which contains a total of $870,370,225$ (870M) molecules. we randomly shuffle and partition the dataset into training (99.7%), validation (0.2%), and test sets (0.1%). The model does not traverse all the data during pretraining, i.e., a total of about 100M molecules are used.

MoleculeNet (Representation). Wu et al. (2018) is a widely-used benchmark dataset for molecular property prediction and drug discovery. It offers a diverse collection of property datasets ranging from quantum mechanics, physical chemistry to biophysics and physiology. Both classification and regression tasks are considered. For rigorous evaluation, we employ standard scaffold splitting, as opposed to random scaffold splitting, for dataset partitioning.

MOSES & ZINC-C (Generation). For few-shot generation, we evaluate GraphsGPT on MOSES (Polykovskiy et al., 2020) dataset, which is designed for benchmarking generative models. Following MOSES, we compute molecular properties (LogP, SA, QED) and scaffolds for molecules collected from ZINC, obtaining ZINC-C. The dataset provides a standardized set of molecules in SMILES format.

4.3. Pretraining

Model Configurations. We adopt the transformer as our model structure. Both the Graph2Seq encoder and the GraphGPT decoder consist of 8 transformer blocks with 8 attention heads. For all layers, we use Swish (Ramachandran et al., 2017) as the activation function and RMSNorm (Zhang & Sennrich, 2019) as the normalizing function. The hidden size is set to 512, and the length of the Graph Position Encoding (GPE) is 128. The total number parameters of the model is 50M. Denote K as the number of Graph Words, multiple versions of GraphsGPT, referred to as GraphsGPT-KW, were pretrained. We mainly use GraphsGPT-1W, while we find that GraphsGPT-8W has better encoding-decoding consistency (Section 6, Q2).

Training Details. The GraphsGPT model undergoes training for 100K steps with a global batch size of 1024 on 8 NVIDIA-A100s, utilizing AdamW optimizer with 0.1 weight decay, where $\beta_1 = 0.9$ and $\beta_2 = 0.95$. The maximum learning rate is $1e^{-4}$ with 5K warmup steps, and the final learning rate decays to $1e^{-5}$ with cosine scheduling.

4.4. Representation

Can Graph2Seq effectively learn expressive graph representation through pretraining?

Table 1: Results of molecular property prediction. We report the mean (standard deviation) metrics of 10 runs with standard scaffold splitting (not random scaffold splitting). The **best** results and the second best are highlighted.

		ROC-AUC ↑						RMSD ↓		
		Tox21	ToxCast	Sider	HIV	BBBP	Bace	ESOL	FreeSolv	Lipo
	# Molecules	7,831	8,575	1,427	41,127	2,039	1,513	1128	642	4200
	# Tasks	12	617	27	1	1	1	1	1	1
No pretrain	GINs	74.6 (0.4)	61.7 (0.5)	58.2 (1.7)	75.5 (0.8)	65.7 (3.3)	72.4 (3.8)	1.050 (0.008)	2.082 (0.082)	0.683 (0.016)
No pretrain	Graph2Seq-1W	74.0 (0.4)	62.6 (0.3)	66.6 (1.1)	73.6 (3.4)	68.3 (1.4)	77.3 (1.2)	0.953 (0.025)	1.936 (0.246)	0.907 (0.021)
No pretrain	Relative gain to GIN	-0.8%	+1.4%	+12.6%	-2.6%	+3.8%	+6.3%	+10.2%	+7.5%	-24.7%
Pretrain	InfoGraph (Sun et al., 2019)	73.3 (0.6)	61.8 (0.4)	58.7 (0.6)	75.4 (4.3)	68.7 (0.6)	74.3 (2.6)			
Pretrain	GPT-GNN (Hu et al., 2020b)	74.9 (0.3)	62.5 (0.4)	58.1 (0.3)	58.3 (5.2)	64.5 (1.4)	77.9 (3.2)			
Pretrain	EdgePred (Hamilton et al., 2017)	76.0 (0.6)	64.1 (0.6)	60.4 (0.7)	64.1 (3.7)	67.3 (2.4)	77.3 (3.5)			
Pretrain	ContextPred (Hu et al., 2019)	73.6 (0.3)	62.6 (0.6)	59.7 (1.8)	74.0 (3.4)	70.6 (1.5)	78.8 (1.2)			
Pretrain	GraphLoG (Xu et al., 2021)	75.0 (0.6)	63.4 (0.6)	59.6 (1.9)	75.7 (2.4)	68.7 (1.6)	78.6 (1.0)			
Pretrain	G-Contextual (Rong et al., 2020)	75.0 (0.6)	62.8 (0.7)	58.7 (1.0)	60.6 (5.2)	69.9 (2.1)	79.3 (1.1)			
Pretrain	G-Motif (Rong et al., 2020)	73.6 (0.7)	62.3 (0.6)	61.0 (1.5)	77.7 (2.7)	66.9 (3.1)	73.0 (3.3)			
Pretrain	AD-GCL (Suresh et al., 2021)	74.9 (0.4)	63.4 (0.7)	61.5 (0.9)	77.2 (2.7)	70.7 (0.3)	76.6 (1.5)			
Pretrain	JOAO (You et al., 2021)	74.8 (0.6)	62.8 (0.7)	60.4 (1.5)	66.6 (3.1)	66.4 (1.0)	73.2 (1.6)	1.120 (0.003)		0.708 (0.004)
Pretrain	SimGRACE (Xia et al., 2022a)	74.4 (0.3)	62.6 (0.7)	60.2 (0.9)	75.5 (2.0)	71.2 (1.1)	74.9 (2.0)			
Pretrain	GraphCL (You et al., 2020)	75.1 (0.7)	63.0 (0.4)	59.8 (1.3)	77.5 (3.8)	67.8 (2.4)	74.6 (2.1)	0.947 (0.038)	2.233 (0.261)	0.739 (0.009)
Pretrain	GraphMAE (Hou et al., 2022)	75.2 (0.9)	63.6 (0.3)	60.5 (1.2)	76.5 (3.0)	71.2 (1.0)	78.2 (1.5)			
Pretrain	3D InfoMax (Stärk et al., 2022)	74.5 (0.7)	63.5 (0.8)	56.8 (2.1)	62.7 (3.3)	69.1 (1.2)	78.6 (1.9)	0.894 (0.028)	2.337 (0.227)	0.695 (0.012)
Pretrain	GraphMVP (Liu et al., 2021b)	74.9 (0.8)	63.1 (0.2)	60.2 (1.1)	79.1 (2.8)	70.8 (0.5)	79.3 (1.5)	1.029 (0.033)		0.681 (0.010)
Pretrain	MGSSL (Zhang et al., 2021)	75.2 (0.6)	63.3 (0.5)	61.6 (1.0)	77.1 (4.5)	68.8 (0.6)	78.8 (0.9)			
Pretrain	AttrMask (Hu et al., 2019)	75.1 (0.9)	63.3 (0.6)	60.5 (0.9)	73.5 (4.3)	65.2 (1.4)	77.8 (1.8)	1.100 (0.006)	2.764 (0.002)	0.739 (0.003)
Pretrain	MolCLR (Wang et al., 2022)	75.0 (0.2)		58.9 (1.4)	78.1 (0.5)	72.2 (2.1)	82.4 (0.9)	1.271 (0.040)	2.594 (0.249)	0.691 (0.004)
Pretrain	Graphformer (Rong et al., 2020)	74.3 (0.1)	**65.4** (0.4)	64.8 (0.6)	62.5 (0.9)	70.0 (0.1)	82.6 (0.7)	0.983 (0.090)	2.176 (0.052)	0.817 (0.008)
Pretrain	Mole-BERT (Xia et al., 2023)	76.8 (0.5)	64.3 (0.2)	62.8 (1.1)	78.9 (3.0)	71.9 (1.6)	80.8 (1.4)	1.015 (0.030)		**0.676** (0.017)
Pretrain	Relative gain to GIN	+2.9%	+6.0%	+11.3%	+4.8%	+9.9%	+14.1%	+14.9%	-4.5%	+1.0%
Pretrain	Graph2Seq-1W	**76.9** (0.3)	**65.4** (0.5)	**68.2** (0.9)	**79.4** (3.9)	**72.8** (1.5)	**83.4** (1.0)	**0.860** (0.024)	**1.797** (0.237)	0.716 (0.019)
Pretrain	Relative gain to GIN	+3.1%	+6.0%	+17.2%	+5.2%	+10.8%	+15.2%	+18.1%	+13.7%	-4.8%
Pretrain	Relative gain to Graph2Seq-1W	+3.9%	+4.5%	+2.4%	+7.9%	+6.6%	+7.9%	+9.8%	+7.2%	+21.1%

Setting & Baselines. We finetune the pretrained Graph2Seq-1W on the MoleculeNet dataset. The learned Graph Words are input into a linear layer for graph classification or regression. We adhere to standard scaffold splitting (not random scaffold splitting) for rigorous and meaningful comparison. We do not incorporate the 3D structure of molecules for modeling. Recent strong molecular graph pretraining baselines are considered for comparison.

We show property prediction results in Table 1, finding that:

Pure Transformer is Competitive to GNN. Without pretraining, Graph2Seq-1W demonstrates a comparable performance to GNN. Specifically, in 4 out of 9 cases, Graph2Seq-1W outperforms GIN with gains exceeding 5%, and in another 4 out of 9 cases, it achieves similar performance with an absolute relative gain of less than 5%. In addition, pure transformer runs much faster than GNNs, i.e., we finish the pretraining of GraphsGPT within 6 hours using 8 A100.

GPT-Style Pretraining is All You Need. Pretrained Graph2Seq demonstrates a non-trivial improvement over 8 out of 9 datasets when compared to baselines. These results are achieved without employing complex pretraining strategies such as multi-pretext combination and hard-negative sampling, highlighting that GPT-pretraining alone is sufficient for achieving SOTA performance and providing a simple yet effective solution for graph SSL.

Graph2Seq Benefits More from GPT-Style Pretraining. The non-trivial improvement has not been observed by previous GPT-GNN (Hu et al., 2020b), which adopts a node-centric generation strategy and GNN architectures. This suggests that the transformer model is more suitable for scaling to large datasets. In addition, previous pretrained transformers without the GPT-style pretraining (Rong et al., 2020) perform worse than Graph2Seq. This underscores that generating the entire graph enhances the learning of global topology and results in more expressive representations.

4.5. Generation

Could pretrained GraphGPT serve as a strong structural prior model for graph generation?

GraphGPT Generates Novel Molecules with High Validity. We assess pretrained GraphGPT-1W on the MOSES dataset through few-shots generation without finetuning. By extracting Graph Word embeddings $\{\boldsymbol{h}_i\}_{i=1}^{M}$ from M training molecules, we construct a mixture Gaussian distribution $p(\boldsymbol{h}, s) = \sum_{i=1}^{M} \mathcal{N}(\boldsymbol{h}_i, s\mathrm{I})$, where s is the standard variance. We sample M molecules from $p(\boldsymbol{h}, s)$ and report the validity, uniqueness, novelty and IntDiv in Table 2. We observe that GraphGPT generates novel molecules with high validity. Without any finetuning, GraphGPT outperforms MolGPT on validity, uniqueness, novelty, and diversity. Definition of metrics could be found in the Appendix B.

Table 2: Few-shot generation results of GraphGPT-1W. We use $M = 100$K shots and sample the same number of Graph Word embeddings under different variance s.

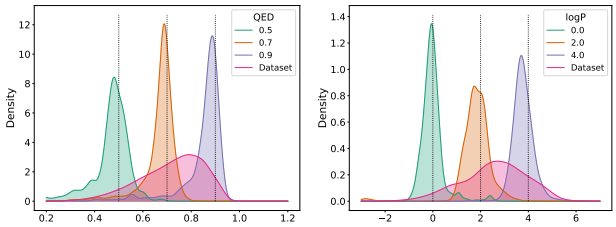
	Model	Validity ↑	Unique ↑	Novelty ↑	IntDiv$_1$ ↑	IntDiv$_2$ ↑
Unconditional	HMM	0.076	0.567	**0.999**	0.847	0.810
	NGram	0.238	0.922	0.969	0.874	0.864
	Combinatorial	**1.0**	0.991	0.988	0.873	0.867
	CharRNN	0.975	0.999	0.842	0.856	0.850
	VAE	0.977	0.998	0.695	0.856	0.850
	AEE	0.937	0.997	0.793	0.856	0.850
	LatentGAN	0.897	0.997	0.949	0.857	0.850
	JT-VAE	**1.0**	0.999	0.914	0.855	0.849
	MolGPT	0.994	**1.0**	0.797	0.857	0.851
Few Shot	GraphGPT-1W$_{s=0.25}$	**0.995**	0.995	0.255	0.854	0.850
	GraphGPT-1W$_{s=0.5}$	0.993	0.996	0.334	0.856	0.848
	GraphGPT-1W$_{s=1.0}$	0.978	0.997	0.871	0.860	0.857
	GraphGPT-1W$_{s=2.0}$	0.972	**1.0**	**1.0**	0.850	0.847

GraphGPT-C is a Controllable Molecule Generator. Following (Bagal et al., 2021), we finetune GraphsGPT-1W on 100M molecules from ZINC-C with properties and scaffolds as prefix inputs, obtaining GraphsGPT-1W-C. We access whether the model could generate molecules satisfying specified properties. We present summarized results in Figure 5 and Table 3, while providing the full results in the appendix due to space limit. The evaluation is conducted using the scaffold "c1ccccc1", demonstrating that GraphGPT can effectively control the properties of generated molecules. Table 3 further confirms that unsupervised pretraining enhances the controllability and validity of GraphGPT. More details can be found in Appendix B.2.

4.6. Euclidean Graph Words

What opportunities do the Euclidean Graph Words offer that were previously considered challenging?

For graph classification, let the i-th sample be denoted as $(\mathcal{G}_i, \boldsymbol{y}_i)$, where $\mathcal{G}_i$ and $\boldsymbol{y}_i$ represent the graph and one-hot label, respectively. When considering paired graphs $(\mathcal{G}_i, \boldsymbol{y}_i)$ and $(\mathcal{G}_j, \boldsymbol{y}_j)$, and employing a mixing ratio λ sampled from the $Beta(\alpha, \alpha)$ distribution, the mixed label is defined as



(a) QED (b) logP

Figure 5: Property distribution of generated molecules on different conditions using GraphsGPT-1W-C. "Dataset" denotes the distribution of the training dataset (ZINC-C).

Table 3: Comparison with MolGPT on different properties. "MAD" denotes the Mean Absolute Deviation in generated molecule properties compared to the oracle value. "SD" denotes the Standard Deviation of the generated property.

	Pretrain	Metric	QED=0.5	SA=0.7	logP=0.0	Avg.
MolGPT	✗	MAD ↓	0.081	0.024	0.304	0.136
		SD ↓	**0.065**	**0.022**	**0.295**	**0.127**
		Validity ↑	0.985	0.975	0.982	0.981
GraphGPT-1W-C	✗	MAD ↓	0.041	0.012	0.103	0.052
		SD ↓	0.079	0.055	0.460	0.198
		Validity ↑	0.988	**0.995**	0.980	0.988
	✓	MAD ↓	**0.032**	**0.002**	**0.017**	**0.017**
		SD ↓	0.080	0.042	0.404	0.175
		Validity ↑	**0.996**	**0.995**	**0.994**	**0.995**

$\boldsymbol{y}_{mix} = \lambda \boldsymbol{y}_i + (1-\lambda)\boldsymbol{y}_j$. However, due to the irregular, unaligned, and Non-Euclidean nature of graph data, applying mixup to get $\mathcal{G}_{mix}$ is nontrivial. Recent efforts (Zhou et al., 2020b; Park et al., 2022; Wu et al., 2022; Zhang et al., 2023; Guo & Mao, 2023) have attempted to address this challenge by introducing complex hand-crafted rules. Additionally, $\mathcal{G}$-mixup (Han et al., 2022) leverages estimated graphons for generating mixed graphs. To our best knowledge, there are currently no learnable model for mixing in Euclidean space while generating new graphs.

Table 4: Graph mixup results. We compare Graph2Seq with $\mathcal{G}$-mixup on multiple tasks from MoleculeNet.

	mixup	HIV ↑	BBBP ↑	Bace ↑	Tox21 ↑	ToxCast ↑	Sider ↑
G-Mix	✗	77.1	68.4	75.9			
	✓	77.1	70.2	77.8			
	gain	+0.0	+1.8	+1.9			
Ours	✗	79.4	72.8	83.4	76.9	65.4	68.2
	✓	79.8	73.4	85.4	77.2	65.5	68.9
	gain	+0.4	+0.6	+2.0	+0.3	+0.1	+0.7

GraphsGPT is a Competitive Graph Mixer. We mixup the learned Graph Words encoded by Graph2Seq-1W, then generate the mixed graph using GraphGPT-1W. Formally, the Graph Words of $\mathcal{G}_i$ and $\mathcal{G}_j$ are $\mathcal{W}_i = \mathrm{Graph2Seq}(\mathcal{G}_i)$ and $\mathcal{W}_j = \mathrm{Graph2Seq}(\mathcal{G}_j)$, and the mixed graph is $\mathcal{G}_{mix} = \mathrm{GraphGPT}(\lambda\mathcal{W}_i + (1-\lambda)\mathcal{W}_j)$. We conduct experiments on MoleculeNet and show the results in Table 4. We observe that the straightforward latent mixup outperforms the elaborately designed $\mathcal{G}$-mixup proposed in the ICML'22 outstanding paper (Han et al., 2022).

Due to page limit, more results are moved to the appendix.

5. Conclusion

We propose GraphsGPT, the first framework with pure transformer that converts Non-Euclidean graph into Euclidean representations, while preserving information using an edge-centric GPT-style pretraining task. We show that the Graph2Seq and GraphGPT serve as strong graph learners for representation and generation, respectively. The Euclidean representations offer more opportunities previously known to be challenging. The GraphsGPT may create a new paradigm of graph modeling.

6. Rebuttal Details

Q1 Missing discussion on diffusion-based molecular generative models.

R1 We conduct additional experiments following (Kong et al., 2023) to compare GraphGPT-1W with the diffusion-based methods on ZINC-250K. We follow the same few-shots generation setting described in the Section 4.5, where we set $M = 10$K for fair comparison. As shown in Table 5, we find that GraphGPT-1W surpasses these methods in a large margin on various metrics, which can further validate the strong generation ability of GraphGPT.

Table 5: Comparison with diffusion-based methods on ZINC-250K. We use $M = 10$K shots and sample the same number of Graph Word under different variance s.

Model	Valid ↑	Unique ↑	Novel ↑	NSPDK ↓	FCD ↓
GraphAF (Shi et al., 2020)	68.47	98.64	**100**	0.044	16.02
GraphDF (Luo et al., 2021)	90.61	99.63	**100**	0.177	33.55
MoFlow (Zang & Wang, 2020)	63.11	99.99	**100**	0.046	20.93
EDP-GNN (Niu et al., 2020)	82.97	99.79	**100**	0.049	16.74
GraphEBM (Liu et al., 2021a)	5.29	98.79	**100**	0.212	35.47
SPECTRE (Martinkus et al., 2022)	90.20	67.05	**100**	0.109	18.44
GDSS (Jo et al., 2022)	97.01	99.64	**100**	0.019	14.66
DiGress (Vignac et al., 2022)	91.02	81.23	**100**	0.082	23.06
GRAPHARM (Kong et al., 2023)	88.23	99.46	**100**	0.055	16.26
GraphGPT-1W$_{s=0.25}$	**99.67**	99.95	93.0	**0.0002**	**1.78**
GraphGPT-1W$_{s=0.5}$	99.57	99.97	93.6	0.0003	1.79
GraphGPT-1W$_{s=1.0}$	98.44	**100**	98.0	0.0012	2.89
GraphGPT-1W$_{s=2.0}$	97.64	**100**	**100**	0.0056	8.47

Q2 How do the method consider the symmetry of graphs? Graph data is invariant to permutation.

R2 In Section 3.2, we mention that "we introduce a random shuffle of the position Codebook". We should explicitly state that this random shuffle of position vectors is equivalent to randomly shuffling the input order of atoms. This allows the model to learn from the data with random order augmentation. We point that building a permutation-invariant encoder is easy and necessary, however, developing a decoder with permutation invariance poses a significant challenge for auto-regressive generation models. We randomly shuffle the position vectors, allowing the model to learn representations with different orders for molecules.

To further verify the effectiveness of our method in handling the permutation invariance, we conduct an additional experiment. Given an input molecular graph sequence, we randomly permute its order for 1024 times and encode the shuffled sequences with Graph2Seq, obtaining a set of 1024 Graph Words. We then decode them back to the graph sequences and observe the consistency, which is defined as the maximum percentage of the decoded sequences that share the same results. Table 6 shows the results on 1000 molecules from the test set, where we find both models are resistant to a certain degree of permutation invariance, i.e., 96.1% and 97.9% of the average consistency for GraphsGPT-1W and GraphsGPT-8W, respectively.

Table 6: Self-consistency of decoded sequences. "C@N" denotes the decoded results of N out of the total 1024 permutations for each molecule are consistent. "Avg." denotes the average consistency of all test data.

Models	C@256	C@512	C@768	C@1024	Avg.
GraphsGPT-1W	100%	99.2%	94.1%	77.3%	96.1%
GraphsGPT-8W	100%	99.4%	96.5%	85.3%	**97.9%**

In addition, there is a contradiction between permutation-invariant model and auto-regressive model. Previous work (TokenGT (Kim et al., 2022)) focuses on representation learning, therefore, do not suffer from the issue of permutation-invariant. We combine representation with generation tasks in the same model, and propose the technique of randomly shuffling position vectors so that all tasks can work well. We should note that randomly shuffling the position vector Codebook is more effective than shuffling the atom order itself. Readers can read the openreview rebuttal.

Acknowledgements

This work was supported by the Science & Technology Innovation 2030 Major Program Project No. 2021ZD0150100, National Natural Science Foundation of China Project No. U21A20427, Project No. WU2022A009 from the Center of Synthetic Biology and Integrated Bioengineering of Westlake University, and Project No. WU2023C019 from the Westlake University Industries of the Future Research. Finally, we thank the Westlake University HPC Center for providing part of the computational resources.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here. GraphsGPT provides a new paradigm for graph representation, generation and manipulation. The Non-Euclidean to Euclidean transformation may affect broader downstream graph applications, such as graph translation and optimization. The methodology could be extend to other modalities, such as image and sequence.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Ahmed, A., Shervashidze, N., Narayanamurthy, S., Josifovski, V., and Smola, A. J. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pp. 37–48, 2013.

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.

Bagal, V., Aggarwal, R., Vinod, P., and Priyakumar, U. D. Molgpt: molecular generation using a transformer-decoder model. *Journal of Chemical Information and Modeling*, 62(9):2064–2076, 2021.

Brown, N., Fiscato, M., Segler, M. H., and Vaucher, A. C. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling*, 59(3):1096–1108, 2019.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chanpuriya, S. and Musco, C. Infinitewalk: Deep network embeddings as laplacian embeddings with a nonlinearity. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1325–1333, 2020.

Chen, D., O'Bray, L., and Borgwardt, K. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, pp. 3469–3489. PMLR, 2022.

Chen, J., Ma, T., and Xiao, C. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 257–266, 2019.

Dehghani, M., Djolonga, J., Mustafa, B., Padlewski, P., Heek, J., Gilmer, J., Steiner, A. P., et al. Scaling vision transformers to 22 billion parameters. In *ICML*, pp. 7480–7512. PMLR, 2023.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018.

Diehl, F. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990*, 2019.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Dwivedi, V. P. and Bresson, X. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

Gao, Z., Tan, C., and Li, S. Z. Pifold: Toward effective and efficient protein inverse folding. In *The Eleventh International Conference on Learning Representations*, 2022.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Guo, H. and Mao, Y. Interpolating graph pair to regularize graph classification. In *AAAI*, volume 37, pp. 7766–7774, 2023.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Han, X., Jiang, Z., Liu, N., and Hu, X. G-mixup: Graph data augmentation for graph classification. In *ICML*, pp. 8230–8248. PMLR, 2022.

Hou, Z., Liu, X., Cen, Y., Dong, Y., Yang, H., Wang, C., and Tang, J. Graphmae: Self-supervised masked graph autoencoders. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 594–604, 2022.

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

Hu, Z., Dong, Y., Wang, K., Chang, K.-W., and Sun, Y. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1857–1867, 2020a.

Hu, Z., Dong, Y., Wang, K., Chang, K.-W., and Sun, Y. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1857–1867, 2020b.

Hu, Z., Dong, Y., Wang, K., and Sun, Y. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*, pp. 2704–2710, 2020c.

Huang, Y., Peng, X., Ma, J., and Zhang, M. 3dlinker: an e (3) equivariant variational autoencoder for molecular linker design. *arXiv preprint arXiv:2205.07309*, 2022.

Hussain, M. S., Zaki, M. J., and Subramanian, D. Edge-augmented graph transformers: Global self-attention is enough for graphs. *arXiv preprint arXiv:2108.03348*, 2021.

Hwang, D., Park, J., Kwon, S., Kim, K., Ha, J.-W., and Kim, H. J. Self-supervised auxiliary learning with meta-paths for heterogeneous graphs. *Advances in Neural Information Processing Systems*, 33:10294–10305, 2020.

Irwin, J. J. and Shoichet, B. K. Zinc- a free database of commercially available compounds for virtual screening. *Journal of chemical information and modeling*, 45(1): 177–182, 2005.

Jin, W., Derr, T., Liu, H., Wang, Y., Wang, S., Liu, Z., and Tang, J. Self-supervised learning on graphs: Deep insights and new direction. *arXiv preprint arXiv:2006.10141*, 2020.

Jo, J., Lee, S., and Hwang, S. J. Score-based generative modeling of graphs via the system of stochastic differential equations. In *International Conference on Machine Learning*, pp. 10362–10383. PMLR, 2022.

Kim, J., Nguyen, D., Min, S., Cho, S., Lee, M., Lee, H., and Hong, S. Pure transformers are powerful graph learners. *Advances in Neural Information Processing Systems*, 35: 14582–14595, 2022.

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

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Kong, L., Cui, J., Sun, H., Zhuang, Y., Prakash, B. A., and Zhang, C. Autoregressive diffusion model for graph generation. In *International conference on machine learning*, pp. 17391–17408. PMLR, 2023.

Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

Lee, J., Lee, I., and Kang, J. Self-attention graph pooling. In *International conference on machine learning*, pp. 3734–3743. PMLR, 2019.

Li, X., Sun, L., Ling, M., and Peng, Y. A survey of graph neural network based recommendation in social networks. *Neurocomputing*, pp. 126441, 2023a.

Li, Z., Gao, Z., Tan, C., Li, S. Z., and Yang, L. T. General point model with autoencoding and autoregressive. *arXiv preprint arXiv:2310.16861*, 2023b.

Lin, Z., Tian, C., Hou, Y., and Zhao, W. X. Improving graph collaborative filtering with neighborhood-enriched contrastive learning. In *Proceedings of the ACM Web Conference 2022*, pp. 2320–2329, 2022.

Liu, C., Li, Y., Lin, H., and Zhang, C. Gnnrec: Gated graph neural network for session-based social recommendation model. *Journal of Intelligent Information Systems*, 60(1): 137–156, 2023a.

Liu, M., Yan, K., Oztekin, B., and Ji, S. Graphebm: Molecular graph generation with energy-based models. *arXiv preprint arXiv:2102.00546*, 2021a.

Liu, S., Wang, H., Liu, W., Lasenby, J., Guo, H., and Tang, J. Pre-training molecular graph representation with 3d geometry. *arXiv preprint arXiv:2110.07728*, 2021b.

Liu, X., Zhang, F., Hou, Z., Mian, L., Wang, Z., Zhang, J., and Tang, J. Self-supervised learning: Generative or contrastive. *IEEE transactions on knowledge and data engineering*, 35(1):857–876, 2021c.

Liu, Y., Jin, M., Pan, S., Zhou, C., Zheng, Y., Xia, F., and Philip, S. Y. Graph self-supervised learning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(6):5879–5900, 2022.

Liu, Y., Yang, X., Zhou, S., Liu, X., Wang, S., Liang, K., Tu, W., and Li, L. Simple contrastive graph clustering. *IEEE Transactions on Neural Networks and Learning Systems*, 2023b.

Liu, Y., Yang, X., Zhou, S., Liu, X., Wang, Z., Liang, K., Tu, W., Li, L., Duan, J., and Chen, C. Hard sample aware network for contrastive deep graph clustering. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 8914–8922, 2023c.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, pp. 10012–10022, 2021d.

Luo, Y., Yan, K., and Ji, S. Graphdf: A discrete flow model for molecular graph generation. In *International conference on machine learning*, pp. 7192–7203. PMLR, 2021.

Ma, Y., Wang, S., Aggarwal, C. C., and Tang, J. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 723–731, 2019.

Martinkus, K., Loukas, A., Perraudin, N., and Wattenhofer, R. Spectre: Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators. In *International Conference on Machine Learning*, pp. 15159–15179. PMLR, 2022.

McInnes, L. and Healy, J. Accelerated hierarchical density based clustering. In *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*, pp. 33–42. IEEE, 2017.

McInnes, L., Healy, J., and Melville, J. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Mialon, G., Chen, D., Selosse, M., and Mairal, J. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021.

Min, E., Chen, R., Bian, Y., Xu, T., Zhao, K., Huang, W., Zhao, P., Huang, J., Ananiadou, S., and Rong, Y. Transformer for graphs: An overview from architecture perspective. *arXiv preprint arXiv:2202.08455*, 2022.

Niu, C., Song, Y., Song, J., Zhao, S., Grover, A., and Ermon, S. Permutation invariant graph generation via score-based generative modeling. In *International Conference on Artificial Intelligence and Statistics*, pp. 4474–4484. PMLR, 2020.

Pang, Y., Wang, W., Tay, F. E., Liu, W., Tian, Y., and Yuan, L. Masked autoencoders for point cloud self-supervised learning. In *ECCV*, pp. 604–621. Springer, 2022.

Park, J., Shim, H., and Yang, E. Graph transplant: Node saliency-guided graph mixup with local structure preservation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 7966–7974, 2022.

Peng, X., Luo, S., Guan, J., Xie, Q., Peng, J., and Ma, J. Pocket2mol: Efficient molecular sampling based on 3d protein pockets. In *International Conference on Machine Learning*, pp. 17644–17655. PMLR, 2022.

Peng, Z., Dong, Y., Luo, M., Wu, X.-M., and Zheng, Q. Self-supervised graph representation learning via global context prediction. *arXiv:2003.01604*, 2020a.

Peng, Z., Huang, W., Luo, M., Zheng, Q., Rong, Y., Xu, T., and Huang, J. Graph representation learning via graphical mutual information maximization. In *Proceedings of The Web Conference 2020*, pp. 259–270, 2020b.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

Polykovskiy, D., Zhebrak, A., Sanchez-Lengeling, B., Golovanov, S., Tatanov, O., Belyaev, S., Kurbanov, R., Artamonov, A., Aladinskiy, V., Veselov, M., et al. Molecular sets (moses): a benchmarking platform for molecular generation models. *Frontiers in pharmacology*, 11:565644, 2020.

Qiu, J., Chen, Q., Dong, Y., Zhang, J., Yang, H., Ding, M., Wang, K., and Tang, J. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1150–1160, 2020.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *arXiv:1710.05941*, 2017.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

Rong, Y., Bian, Y., Xu, T., Xie, W., Wei, Y., Huang, W., and Huang, J. Self-supervised graph transformer on large-scale molecular data. *Advances in Neural Information Processing Systems*, 33:12559–12571, 2020.

Shakibajahromi, B., Kim, E., and Breen, D. E. Rimeshgnn: A rotation-invariant graph neural network for mesh classification. In *WACV*, pp. 3150–3160, 2024.

Shi, C., Xu, M., Zhu, Z., Zhang, W., Zhang, M., and Tang, J. Graphaf: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations*, 2019.

Shi, C., Xu, M., Zhu, Z., Zhang, W., Zhang, M., and Tang, J. Graphaf: a flow-based autoregressive model for molecular graph generation. *arXiv preprint arXiv:2001.09382*, 2020.

Stärk, H., Beaini, D., Corso, G., Tossou, P., Dallago, C., Günnemann, S., and Liò, P. 3d infomax improves gnns for molecular property prediction. In *ICML*, pp. 20479–20502. PMLR, 2022.

Sun, F.-Y., Hoffmann, J., Verma, V., and Tang, J. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000*, 2019.

Suresh, S., Li, P., Hao, C., and Neville, J. Adversarial graph augmentation to improve graph contrastive learning. *Advances in Neural Information Processing Systems*, 34: 15920–15933, 2021.

Tan, C., Gao, Z., and Li, S. Z. Target-aware molecular graph generation. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 410–427. Springer, 2023.

Tian, Y., Dong, K., Zhang, C., Zhang, C., and Chawla, N. V. Heterogeneous graph masked autoencoders. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 9997–10005, 2023.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Vignac, C., Krawczuk, I., Siraudin, A., Wang, B., Cevher, V., and Frossard, P. Digress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734*, 2022.

Wang, P., Agarwal, K., Ham, C., Choudhury, S., and Reddy, C. K. Self-supervised learning of contextual embeddings for link prediction in heterogeneous networks. In *Proceedings of the web conference 2021*, pp. 2946–2957, 2021.

Wang, Y., Wang, J., Cao, Z., and Barati Farimani, A. Molecular contrastive learning of representations via graph neural networks. *NMI*, 4(3):279–287, 2022.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *ICML*, pp. 6861–6871. PMLR, 2019.

Wu, L., Lin, H., Tan, C., Gao, Z., and Li, S. Z. Self-supervised learning on graphs: Contrastive, generative, or predictive. *IEEE Transactions on Knowledge and Data Engineering*, 2021a.

Wu, L., Xia, J., Gao, Z., et al. Graphmixup: Improving class-imbalanced node classification by reinforcement mixup and self-supervised context prediction. In *ECML-PKDD*, pp. 519–535. Springer, 2022.

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

Wu, Z., Jain, P., Wright, M., Mirhoseini, A., Gonzalez, J. E., and Stoica, I. Representing long-range context for graph neural networks with global attention. *NeurIPS*, 34:13266–13279, 2021b.

Xia, J., Wu, L., Chen, J., Hu, B., and Li, S. Z. Simgrace: A simple framework for graph contrastive learning without data augmentation. In *Proceedings of the ACM Web Conference 2022*, pp. 1070–1079, 2022a.

Xia, J., Zhao, C., Hu, B., Gao, Z., Tan, C., Liu, Y., Li, S., and Li, S. Z. Mole-bert: Rethinking pre-training graph neural networks for molecules. In *The Eleventh International Conference on Learning Representations*, 2022b.

Xia, J., Zhao, C., Hu, B., Gao, Z., Tan, C., Liu, Y., Li, S., and Li, S. Z. Mole-bert: Rethinking pre-training graph neural networks for molecules. In *The Eleventh International Conference on Learning Representations*, 2023.

Xiao, W., Zhao, H., Zheng, V. W., and Song, Y. Vertex-reinforced random walk for network embedding. In *Proceedings of the 2020 SIAM International Conference on Data Mining*, pp. 595–603. SIAM, 2020.

Xie, Y., Xu, Z., Zhang, J., Wang, Z., and Ji, S. Self-supervised learning of graph neural networks: A unified review. *IEEE transactions on pattern analysis and machine intelligence*, 45(2):2412–2429, 2022.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Xu, M., Wang, H., Ni, B., Guo, H., and Tang, J. Self-supervised graph-level representation learning with local and global structure. In *International Conference on Machine Learning*, pp. 11548–11558. PMLR, 2021.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., and Shen, Y. Graph contrastive learning with augmentations. *NeurIPS*, 33:5812–5823, 2020.

You, Y., Chen, T., Shen, Y., and Wang, Z. Graph contrastive learning automated. In *International Conference on Machine Learning*, pp. 12121–12132. PMLR, 2021.

Yu, X., Tang, L., Rao, Y., et al. Point-bert: Pre-training 3d point cloud transformers with masked point modeling. In *CVPR*, pp. 19313–19322, 2022.

Zang, C. and Wang, F. Moflow: an invertible flow model for generating molecular graphs. In *Proceedings of the 26th*

ACM SIGKDD international conference on knowledge discovery & data mining, pp. 617–626, 2020.

Zeng, J. and Xie, P. Contrastive self-supervised learning for graph classification. In *AAAI*, volume 35, pp. 10824–10832, 2021.

Zhang, B. and Sennrich, R. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Zhang, J., Luo, D., and Wei, H. Mixupexplainer: Generalizing explanations for graph neural networks with data augmentation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3286–3296, 2023.

Zhang, Z., Liu, Q., Wang, H., Lu, C., and Lee, C.-K. Motif-based graph self-supervised learning for molecular property prediction. *Advances in Neural Information Processing Systems*, 34:15870–15882, 2021.

Zhao, J., Li, C., Wen, Q., Wang, Y., Liu, Y., Sun, H., Xie, X., and Ye, Y. Gophormer: Ego-graph transformer for node classification. *arXiv preprint arXiv:2110.13094*, 2021.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020a.

Zhou, J., Shen, J., and Xuan, Q. Data augmentation for graph classification. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 2341–2344, 2020b.

Zhu, Y., Xu, Y., Yu, F., Liu, Q., Wu, S., and Wang, L. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*, 2020.

Zhu, Y., Xu, Y., Yu, F., et al. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, pp. 2069–2080, 2021.

Zou, D., Wei, W., Mao, X.-L., et al. Multi-level cross-view contrastive learning for knowledge-aware recommender system. In *SIGIR*, pp. 1358–1368, 2022.

A. Representation

When applying graph mixup, the training samples are drawn from the original data with probability p_{self} and from mixed data with probability $(1 - p_{self})$. The mixup hyperparameter α and p_{self} are shown in Table 7.

	Tox21	ToxCast	Sider	HIV	BBBP	BACE	ESOL	FreeSolv	LIPO
batch size	16	16	16	64	128	16	16	64	16
lr	1e-5	5e-5	1e-4	1e-4	5e-4	1e-5	1e-4	1e-4	5e-5
dropout	0.0	0.0	0.0	0.0	0.1 or 0.3	0.0	0.1	0.1	0.0
epoch	50	50	50	50	50 or 100	50	50	50	50
α for mixup	0.5	0.1	0.5	0.5	0.5	0.5	0.5	0.5	0.1
p_{self} for mixup	0.7	0.7	0.7	0.5	0.5	0.7	0.7	0.9	0.7

Table 7: Hyperparameters for property prediction.

B. Generation

B.1. Few-Shots Generation

We introduce metrics (Bagal et al., 2021) of few-shots generation as follows:

- **Validity**: the fraction of a generated molecules that are valid. We use RDkit for validity check of molecules. Validity measures how well the model has learned the SMILES grammar and the valency of atoms.
- **Uniqueness**: the fraction of valid generated molecules that are unique. Low uniqueness highlights repetitive molecule generation and a low level of distribution learning by the model.
- **Novelty**: the fraction of valid unique generated molecules that are not in the training set. Low novelty is a sign of overfitting. We do not want the model to memorize the training data.
- **Internal Diversity (IntDiv$_p$)**: measures the diversity of the generated molecules, which is a metric specially designed to check for mode collapse or whether the model keeps generating similar structures. This uses the power (p) mean of the Tanimoto similarity (T) between the fingerprints of all pairs of molecules $(s1, s2)$ in the generated set (S).

$$\text{InvDiv}_p(S) = 1 - \sqrt[p]{\frac{1}{|S|^2} \sum_{s1,s2 \in S} T(s1, s2)^p} \tag{14}$$

B.2. Conditional Generation

We provide a detailed description of the conditions used for conditional generation as follows:

- **QED (Quantitative Estimate of Drug-likeness):** a measure that quantifies the "drug-likeness" of a molecule based on its pharmacokinetic profile, ranging from 0 to 1.
- **SA (Synthetic Accessibility):** a score that predicts the difficulty of synthesizing a molecule based on multiple factors. Lower SA scores indicate easier synthesis.
- **logP (Partition Coefficient):** a key parameter in studies of drug absorption and distribution in the body that measuring a molecule's hydrophobicity.
- **Scaffold:** the core structure of a molecule, which typically includes rings and the atoms that connect them. It provides a framework upon which different functional groups can be added to create new molecules.

In order to integrate conditional information into our model, we set aside an additional 100M molecules from the ZINC database for finetuning, which we denote as the dataset $\mathcal{D}_{\mathcal{G}}$. For each molecule $\mathcal{G} \in \mathcal{D}_{\mathcal{G}}$, we compute its property values v_{QED}, v_{SA} and v_{logP} and normalize them to 0 mean and 1.0 variance, yielding $\bar{v}_{\text{QED}}$, $\bar{v}_{\text{SA}}$ and $\bar{v}_{\text{logP}}$.

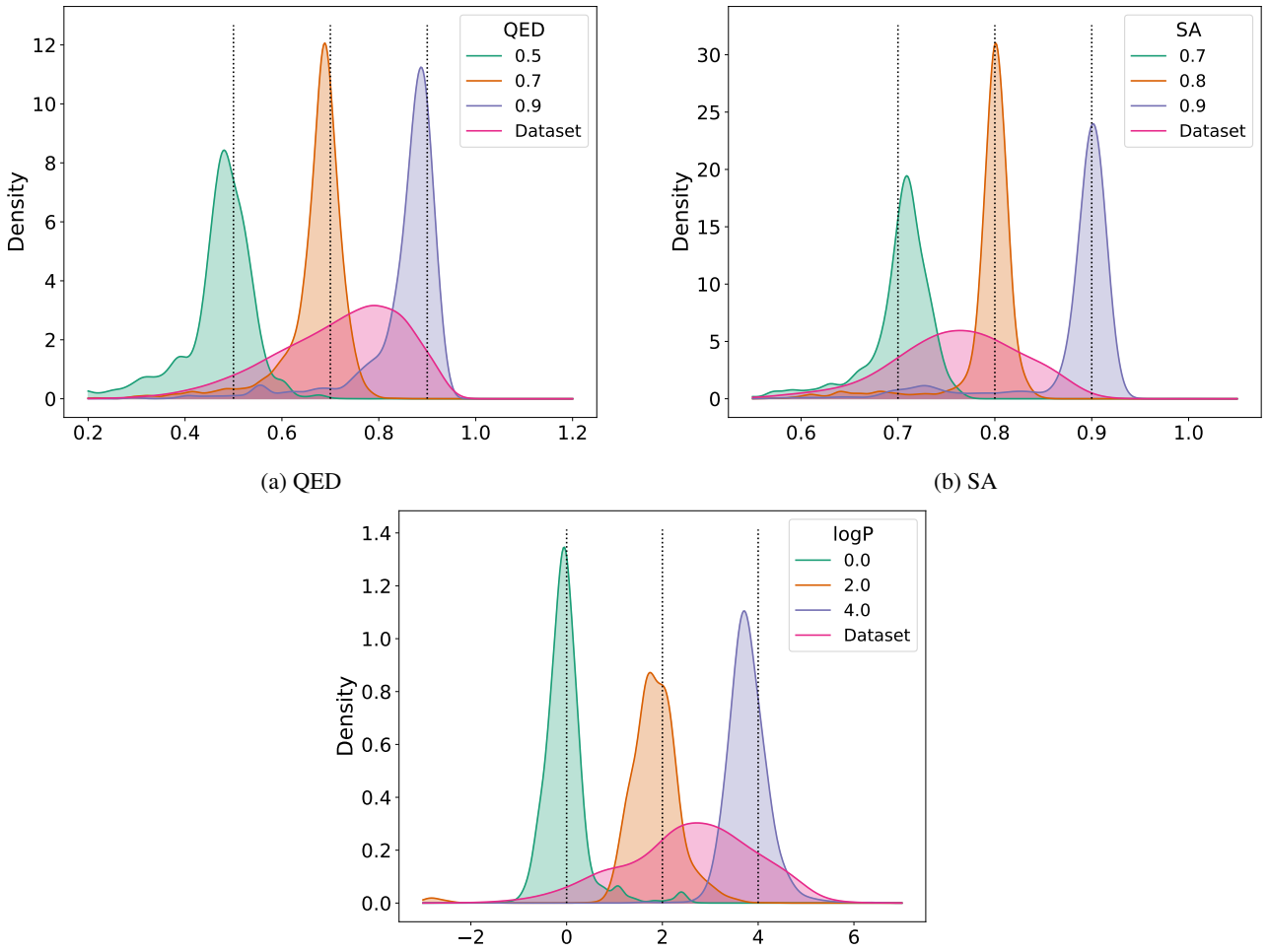
The Graph2Seq model takes all properties and scaffolds as inputs and transforms them into the Graph Word sequence $\mathcal{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_k]$. The additional property and scaffold information enables Graph2Seq to encode Graph Words with conditions. The Graph Words are then subsequently decoded by GraphGPT following the same implementation in Section 3.3. In summary, the inputs of the Graph2Seq encoder comprises:

1. **Graph Word Prompts** $[\texttt{[GW 1]}, \cdots, \texttt{[GW k]}]$, which are identical to the word prompts discussed in Section 3.2.
2. **Property Token Sequence** $[\texttt{[QED]}, \texttt{[SA]}, \texttt{[logP]}]$, which is encoded from the normalized property values $\bar{v}_{\text{QED}}$, $\bar{v}_{\text{SA}}$ and $\bar{v}_{\text{logP}}$.
3. **Scaffold Flexible Token Sequence** $\texttt{FTSeq}_{\text{Scaf}}$, representing the sequence of the scaffold for the molecule.

For the sake of comparison, we followed Bagal et al. (2021) and trained a MolGPT model on the GuacaMol dataset (Brown et al., 2019) using QED, SA, logP, and scaffolds as conditions for 10 epochs. We compare the conditional generation ability by measuring the MAD (Mean Absolute Deviation), SD (Standard Deviation), validity and uniqueness. Table 8 presents the full results, underscoring the superior control of GraphGPT-1W-C over molecular properties.

	Pretrain	Metric	QED=0.5	QED=0.7	QED=0.9	SA=0.7	SA=0.8	SA=0.9	logP=0.0	logP=2.0	logP=4.0	Avg.
MolGPT	✗	MAD ↓	0.081	0.082	0.097	0.024	0.019	0.013	0.304	0.239	0.286	0.127
		SD ↓	0.065	0.066	0.092	0.022	0.016	0.013	0.295	0.232	0.258	**0.118**
		Validity ↑	0.985	0.985	0.984	0.975	0.988	0.995	0.982	0.983	0.982	0.984
GraphGPT-1W-C	✗	MAD ↓	0.041	0.031	0.077	0.012	0.028	0.031	0.103	0.189	0.201	0.079
		SD ↓	0.079	0.077	0.121	0.055	0.062	0.070	0.460	0.656	0.485	0.229
		Validity ↑	0.988	0.995	0.991	0.995	0.991	0.998	0.980	0.992	0.991	0.991
	✔	MAD ↓	0.032	0.033	0.051	0.002	0.009	0.022	0.017	0.190	0.268	**0.069**
		SD ↓	0.080	0.075	0.090	0.042	0.037	0.062	0.463	0.701	0.796	0.261
		Validity ↑	0.996	0.998	0.999	0.995	0.999	0.996	0.994	0.990	0.992	**0.995**

Table 8: Overall comparison between GraphGPT-1W-C and MolGPT on different properties with scaffold SMILES "c1ccccc1". "MAD" denotes the Mean Absolute Deviation of the property value in generated molecules compared to the oracle value. "SD" denotes the Standard Deviation of the generated property.



(a) QED

(b) SA

(c) logP

Figure 6: Property distribution of generated molecules on different conditions using GraphGPT-1W-C.

C. Graph Words

C.1. Clustering

The efficacy of the Graph2Seq encoder hinges on its ability to effectively map Non-Euclidean graphs into Euclidean latent features in a structured manner. To investigate this, we visualize the latent Graph Words space using sampled features, encoding 32,768 molecules with Graph2Seq-1W and employing HDBSCAN (McInnes & Healy, 2017) for clustering the Graph Words.

Figures 7 and 8 respectively illustrate the clustering results and the molecules within each cluster. An intriguing observation emerges from these results: the Graph2Seq model exhibits a propensity to cluster molecules with similar properties (e.g., identical functional groups in clusters 0, 1, 4, 5; similar structures in clusters 2, 3, 7; or similar Halogen atoms in cluster 3) within the latent Graph Words space. This insight could potentially inform and inspire future research.

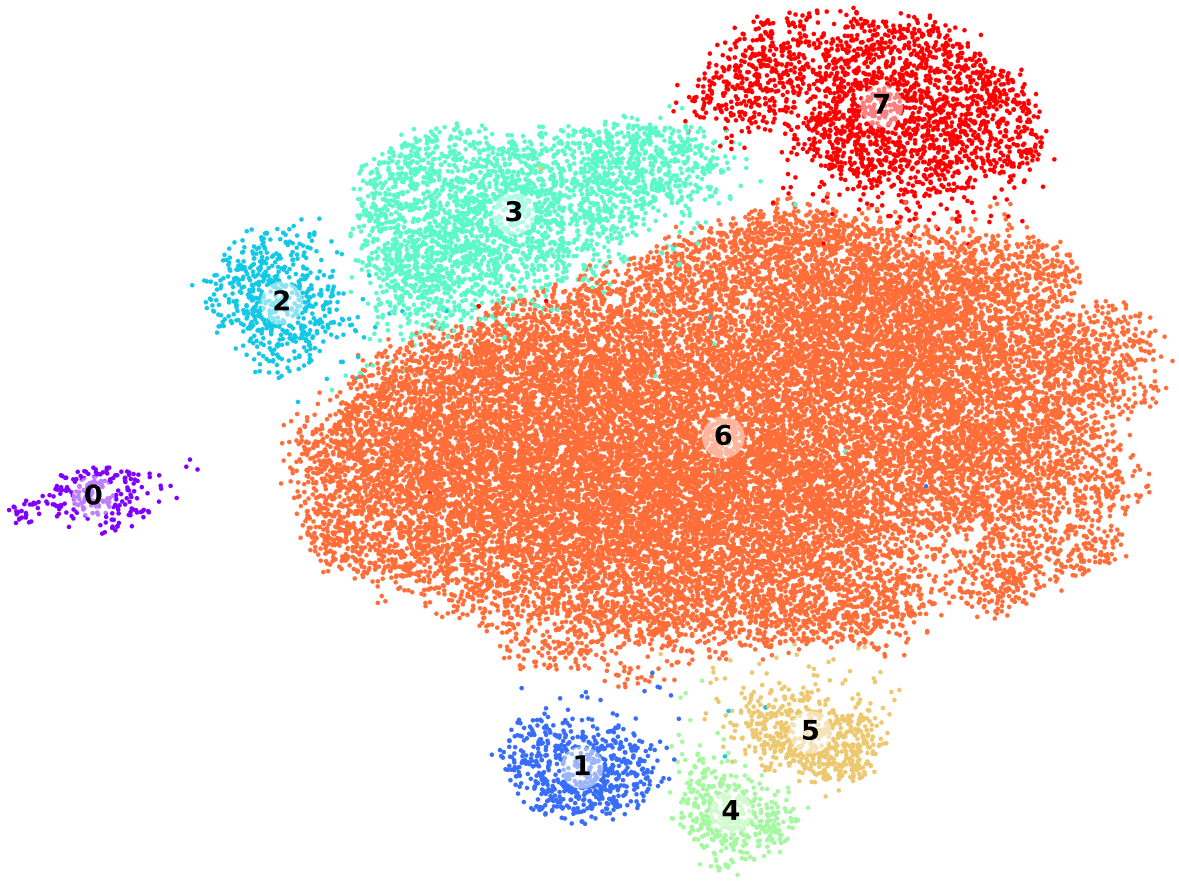
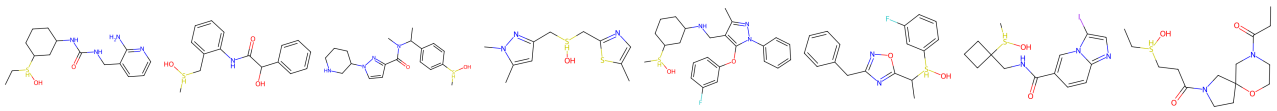
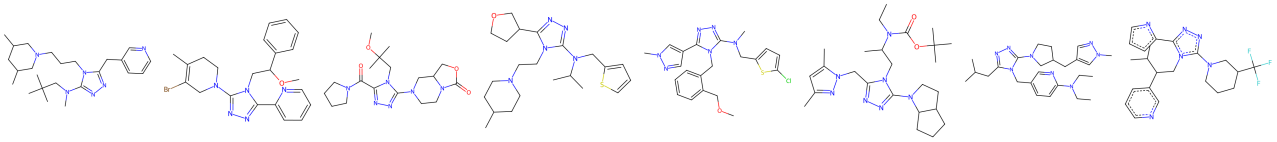


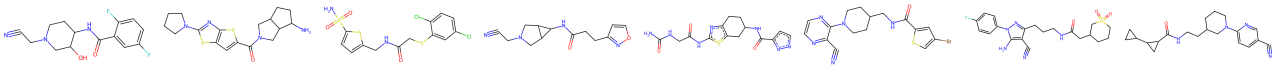
Figure 7: UMAP (McInnes et al., 2018) visualization of the clustering result on the Graph Words of Graph2Seq-1W.



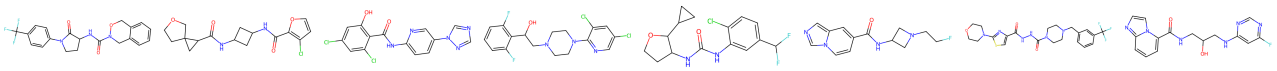
(a) Cluster 0



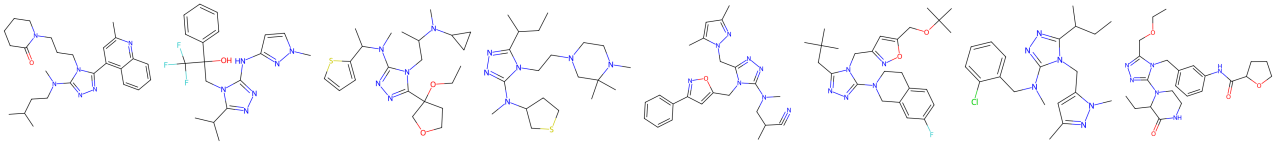
(b) Cluster 1



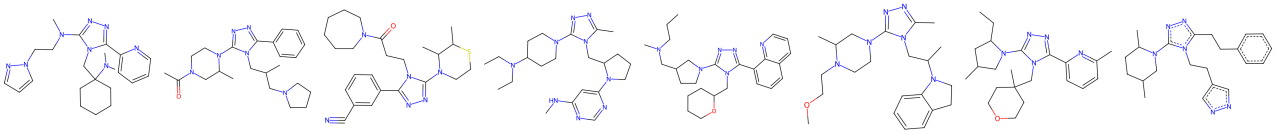
(c) Cluster 2



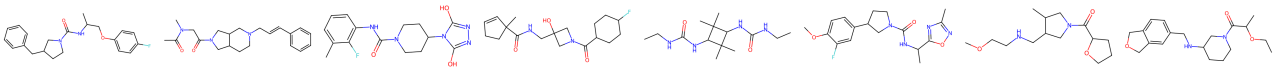
(d) Cluster 3



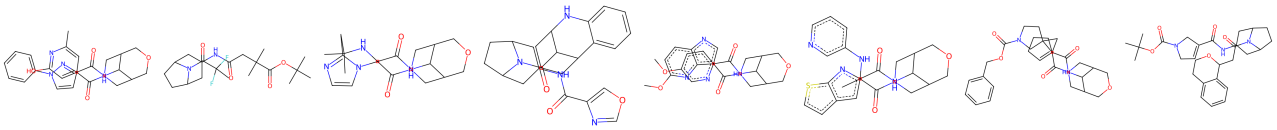
(e) Cluster 4



(f) Cluster 5



(g) Cluster 6



(h) Cluster 7

Figure 8: Visualization of the molecules in each cluster.

C.2. Graph Translation

Graph Interpolation. In exploit of the Euclidean representation of graphs, we explore the continuity of the latent Graph Words using interpolation. Consider a source molecule $\mathcal{G}_s$ and a target molecule $\mathcal{G}_t$. We utilize Graph2Seq to encode them into Graph Words, represented as $\mathcal{W}_s$ and $\mathcal{W}_t$, respectively. We then proceed to conduct a linear interpolation between these two Graph Words, resulting in a series of interpolated Graph Words: $\mathcal{W}'_{\alpha_1}, \mathcal{W}'_{\alpha_2}, \ldots, \mathcal{W}'_{\alpha_k}$, where each interpolated Graph Word is computed as $\mathcal{W}'_{\alpha_i} = (1-\alpha_i)\mathcal{W}_s + \alpha_i \mathcal{W}_t$. These interpolated Graph Words are subsequently decoded back into molecules using GraphGPT.

The interpolation results are depicted in Figure 9. We observe a smooth transition from the source to the target molecule, which demonstrates the model's ability to capture and traverse the continuous latent space of molecular structures effectively. This capability could potentially be exploited for tasks such as molecular optimization and drug discovery.

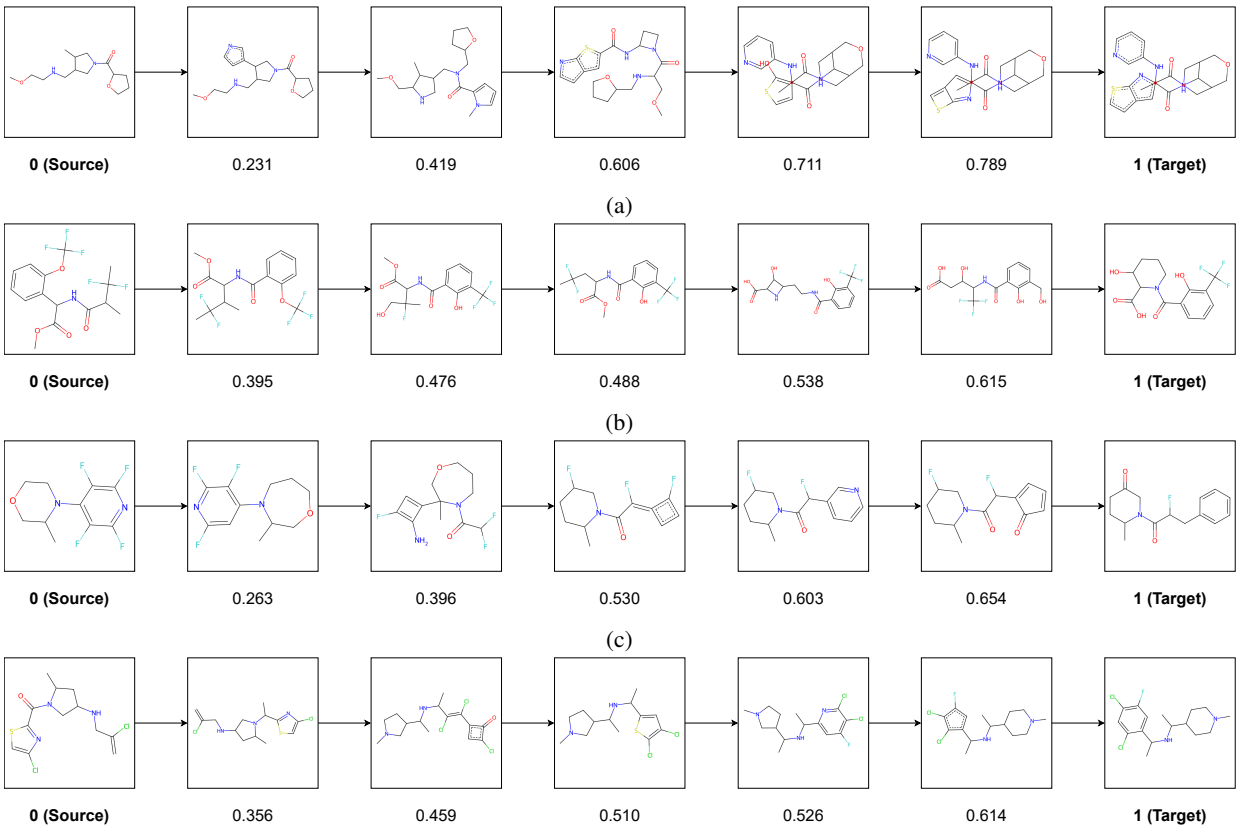


Figure 9: Graph interpolation results with different source and target molecules using GraphsGPT-1W. The numbers denote the values of α for corresponding results.

Graph Hybridization. With Graph2Seq, a graph $\mathcal{G}$ can be transformed into a fixed-length Graph Word sequence $\mathcal{W} = [\boldsymbol{w}_1, \cdots, \boldsymbol{w}_k]$, where each Graph Word is expected to encapsulate distinct semantic information. We investigate the representation of Graph Words by hybridizing them among different inputs.

Specifically, consider a source molecule $\mathcal{G}_s$ and a target molecule $\mathcal{G}_t$, along with their Graph Words $\mathcal{W}_s = [\boldsymbol{w}_{s1}, \cdots, \boldsymbol{w}_{sk}]$ and $\mathcal{W}_t = [\boldsymbol{w}_{t1}, \cdots, \boldsymbol{w}_{tk}]$. Given the indices set I ,we replace a subset of source Graph Words with the corresponding target Graph Words $\boldsymbol{w}_{si} \leftarrow \boldsymbol{w}_{ti}, i \in I$, yielding the hybrid Graph Words $\mathcal{W}_h = [\boldsymbol{w}_{h1}, \cdots, \boldsymbol{w}_{hk}]$, where:

$$\boldsymbol{w}_h = \begin{cases} \boldsymbol{w}_{ti}, & i \in I \\ \boldsymbol{w}_{si}, & i \notin I \end{cases} . \tag{15}$$

We then decode $\mathcal{W}_h$ using GraphGPT back into the graph and observe the changes on the molecules. The results are depicted in Figure 10. From these results, we observe that hybridizing specific Graph Words can lead to the introduction of certain features from the target molecule into the source molecule, such as the Sulfhydryl functional group. This suggests that Graph Words could potentially be used as a tool for manipulating specific features in molecular structures, which could have significant implications for molecular design and optimization tasks.

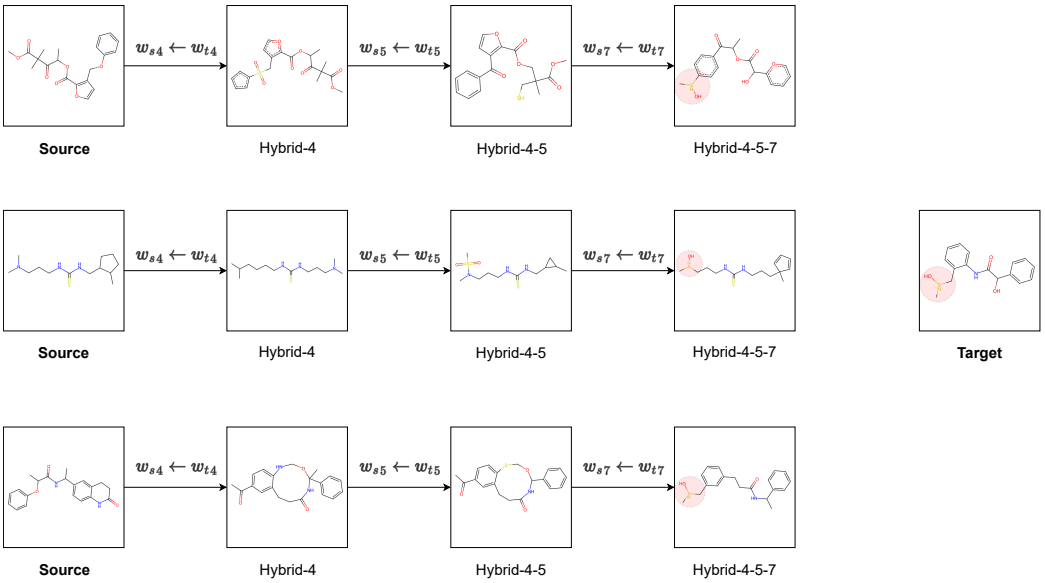


Figure 10: Hybridization results of Graph Words. The figure shows the changes in the source molecule after hybridizing specific Graph Words from the target molecule. We use GraphsGPT-8W which has 8 Graph Words in total.