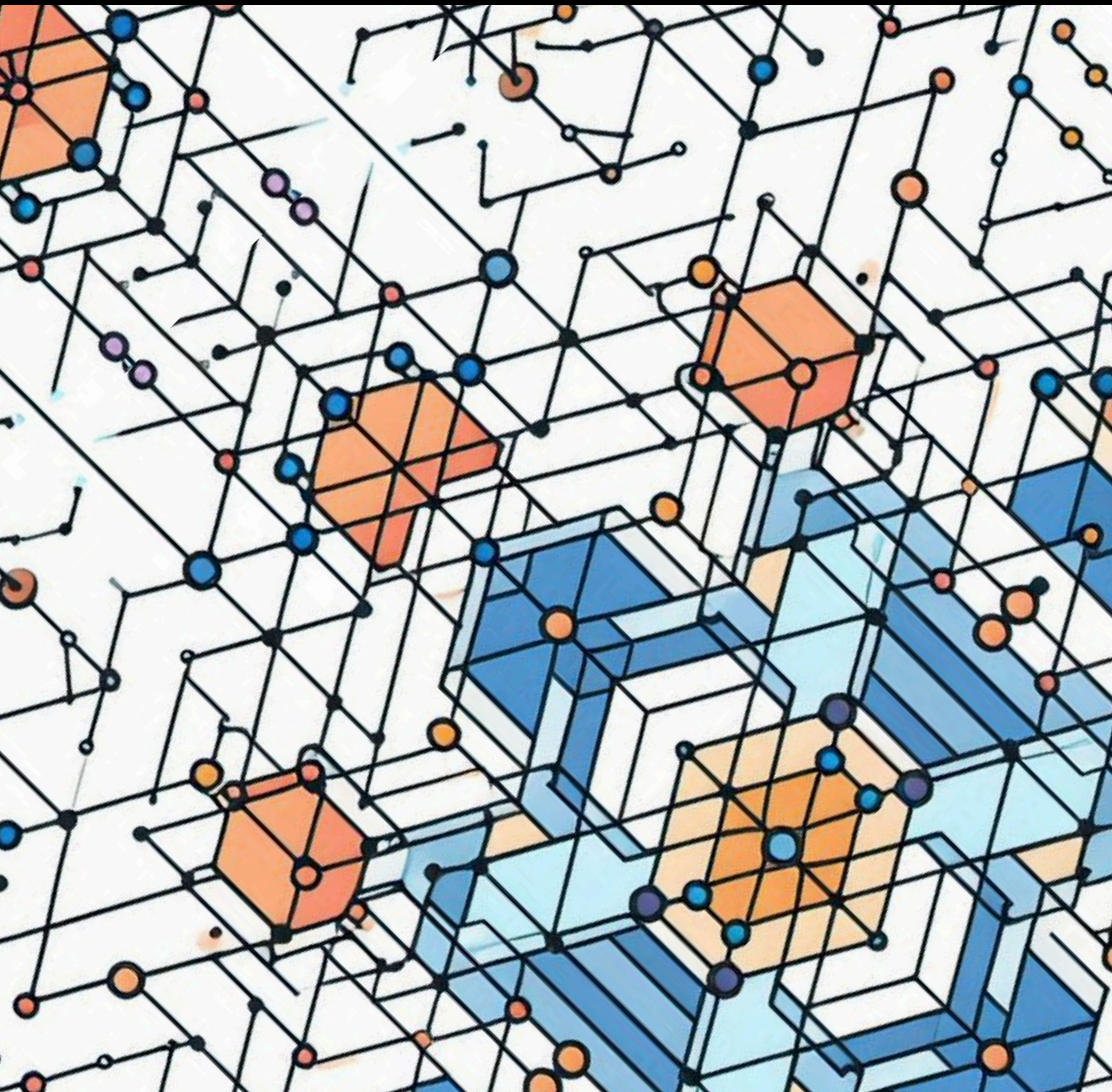
Joint Embedding Predictive Architecture for Self-supervised Pretraining on Polymer Molecular Graphs

Francesco Piccoli

TU Delft



Joint Embedding Predictive Architecture for Self-supervised Pretraining on Polymer Molecular Graphs

by

Francesco Piccoli

to obtain the degree of Master of Science
at Delft University of Technology,
to be defended publicly on
July 1st, 2024 at 10:00.

Student number:	5848474	
Project duration:	November 2023 - July 2024	
Thesis committee:	Prof. dr. ir. M.J.T. Reinders	TU Delft, Responsible advisor
	Dr. J.M. Weber	TU Delft, Daily supervisor
	G. Vogel MSc.	TU Delft, Daily supervisor
	Dr. M. Khosla	TU Delft

TU Delft

Acknowledgements

I would like to thank my supervisor, Gabriel, for his expertise, his kindness, and for supporting me throughout my thesis.

I am also grateful to Jana who gave me the opportunity to work under her supervision and provided me with valuable and expert feedback.

Sono infinitamente grato ai miei genitori e mio fratello, per il loro supporto e per avermi sempre sostenuto in questi anni di studio.

Finally, I want to thank Sirada, for the experiences we shared in this last year of school.

Joint Embedding Predictive Architecture for Self-supervised Pretraining on Polymer Molecular Graphs

Abstract

Recent advancements in machine learning (ML) have shown promise in accelerating polymer discovery by aiding in tasks such as virtual screening via property prediction, and the design of new polymer materials with desired chemical properties. However, progress in polymer ML is hampered by the scarcity of high-quality, labelled datasets, which are necessary for training supervised ML models. In this work, we study the use of the very recent 'Joint Embedding Predictive Architecture' (JEPA) type for self-supervised learning (SSL) on polymer molecular graphs, to understand whether pretraining with the proposed SSL strategy improves downstream performance when labelled data is scarce. By doing so, this study aims to shed light on this new family of architectures in the molecular graph domain and provide insights and directions for future research on JEPAs. Our experimental results indicate that JEPA self-supervised pretraining enhances downstream performance, particularly when labelled data is very scarce, achieving improvements across all tested datasets.

1 Introduction

Synthetic polymers are one of the most widespread classes of materials, constituting an essential component of numerous commodities in everyday life and in industry [2, 36, 37, 84]. The ubiquity of polymers is due to their low-cost processing and chemical stability [25] and to their broad diversity in their chemistry and structure, leading to a wide variety of distinctive properties [12, 36].

The large diversity of the polymer chemical space provides the opportunity for designing polymers whose properties match the application demands [37, 57]. Yet, the high number of combinations of chemical compositions, chain structures, and synthesis methods, bring challenges in the effective browsing of this large search space [37, 84]. In recent years, machine learning (ML) has shown potential in new material (i.e. polymer) discovery [47, 73]. ML methods are increasingly applied in all steps of the materials development cycle: from virtual screening and discovering candidate materials using property prediction, to inverse materials design [12, 52, 57] to create new materials with desired properties.

However, the field of polymer ML is still in its early stages, primarily hindered by the scarcity of high-quality, large, publicly-available, labelled datasets, due to limitations and costs associated with laboratory-derived data [1, 47, 84]. The majority of current studies operate on relatively small labelled datasets, which constrains the potential of the methods developed.

To overcome the labelled data scarcity problem, several strategies have been proposed, via transfer learning [72, 80], multitask learning [25, 36, 49] and self-supervised learning (SSL) [22, 37, 71, 80]. The latter approach is of particular relevance, as it achieved notable successes in other fields, including computer vision [11, 13, 28, 63], natural language processing [17, 50], and more recently, on graphs [34, 45, 70], with successful examples also in the small molecules domain [61]. While SSL has received significant attention for small molecules, particularly in drug discovery, there has been limited exploration for large molecules, especially polymers. The inherent structural characteristics of polymers distinguish them from small molecules, making the direct application of small molecule SSL techniques not well suited. Only recently have some initial works emerged for SSL on polymers (Section 2.2).

In this thesis work, we will study a new architecture family, called Joint Embedding Predictive Architecture (JEPA) [39] for self-supervised pretraining on polymer graphs (for an extensive discussion of JEPAs and their advantages refer to Section 2). In essence, our study focuses on two objectives: firstly, devising

a new self-supervised pretraining strategy based on JEPA, for polymer molecular graphs, to improve the accuracy in downstream tasks (e.g. property prediction). This would be achieved by pretraining the ML model with the proposed method, and then finetuning the model on the labelled data available.
Secondly, the study offers comprehensive insights into the utilization of JEPAs within the graph domain, to shed light on this new architecture type and provide directions for future work with JEPAs. Our research focuses on polymer molecular graphs. While some aspects of our analysis apply broadly to JEPAs across various types of graphs (i.e. in different domains) and extend the study of JEPAs for graphs initiated in [58], other results and experiments are specific to JEPAs in the molecular graph domain, specifically for polymers. This makes our study the first of its kind in this specific area.
The results, discussed in Section 4, show that our pretraining strategy noticeably improves downstream performance, especially in data-scarce scenarios (less than 1000 datapoints), showing significant improvements also when finetuning on a dataset spanning a polymer chemical space very different from the one employed for pretraining. Moreover, our ablation studies on JEPA model design and training strategies, provide new insights into JEPAs for graphs and their effective training.

2 Problem definition

2.1 Self-supervised learning

Defined by Yann LeCun as 'the dark matter of intelligence' [40], SSL is a ML technique where the model learns from the data itself without relying on external labels, by auto-generating labels from the data and hence creating a supervised task. This can be achieved in different ways (Figure 1.A), for instance by predicting missing (masked) parts of the input data [17, 28, 31] or generating data transformations and teaching the model to be invariant to them [14]. In both cases, the model is forced to learn the underlying patterns and structures within the data. Another approach to SSL involves leveraging pseudolabels, namely, some properties of the input that can be automatically derived from the data.
When labelled data is scarce, a model can be pretrained in a SSL fashion, and then finetuned on the limited labelled dataset available, as visible in Figure 1.B. The goal is to improve performance when the model is initialized with the parameters (i.e. knowledge) learned throughout the pretraining stage.
SSL has achieved notable successes in computer vision [11, 13, 28, 63], natural language processing [17, 50], and more recently, on graphs [34, 45, 70], with successful examples also in the small molecules domain [54, 59, 61, 65, 76]. The field is considered one of the most promising for new advancements in ML [40], as it allows to leverage not only the labelled data, but also the unlabelled one, where the available size of the latter is significantly larger compared to the one of the former.

2.2 SSL for polymers

SSL remains largely unexplored for polymers. In [37, 71, 80] the authors exploit Large Language Models (LLMs) for SSL pretraining, obtaining encouraging results. In particular, polyBERT [37] is based on the DeBERTa model [29], and pretrained through masked language modeling (MLM) on approximately 80M polymer strings, generated from about 13,800 original polymers. During finetuning, the pretrained transformer weights are frozen and used to provide polymer embeddings to input to a multi-layer perceptron (MLP) for multitask property prediction. TransPolymer [71] is based on RoBERTa [44] and also utilizes MLM, but on a relatively smaller dataset of around five million polymers augmented from PI1M [46]. Finally, SML-MT [80] is based on BERT [17]. Differently from the previous two models, it exploits also small molecules for pretraining, and not only polymers, leveraging a larger dataset of around one billion molecules. The three aforementioned papers work in a relatively similar manner and obtain similar performance, as nicely reported in [80]. However, these studies suffer from a simplified input polymer representation via PSMILES language, specifying only the repeating unit of the polymer and neglecting important polymer structural information such as chain architecture, stoichiometry, and bonding between monomers' atoms in copolymers, which influence the polymers' properties [80].
In [22] (under review), the authors successfully apply SSL on molecular graphs for polymers for the first time, using the polymer graph representation and dataset from [2] (which we will also use and describe in Sections 3.1 and 3.2.1). In their work, they improve property prediction performance in data-scarce scenarios by pretraining the model in a self-supervised fashion. The authors propose three self-supervised tasks, based on reconstructing masked nodes and masked edges, and a third based on the creation of a pseudolabel, namely the ensemble polymer molecular weight, derived from the polymers' monomers. Nevertheless, the reconstruction-based (generative) tasks

used in [22] have shown limited efficacy for Graph Neural Network (GNN) pretraining over the years [26, p. 71] [43, 64], being outperformed by invariance-based methods [8, 13, 64, 69, 75, 78].

2.3 Joint Embedding Architecture

Invariance-based pretraining is usually associated with Joint Embedding Architectures (JEAs) (Figure 1.C.I), also called Siamese Neural Networks [10, 15], given their joint pair of encoders. JEAs learn to output similar embeddings for compatible inputs x and y, and dissimilar embeddings for incompatible inputs, where compatible inputs are usually obtained via positive data augmentations. Invariance-based methods are trained either in a contrastive [13, 27, 64, 75, 85] or in a non-contrastive fashion [8, 15, 23, 62, 77]. Contrastive methods explicitly push apart embeddings of negative examples and pull together embeddings of positive examples. Non-contrastive training, which uses only positive augmentations, can be done for instance by maximizing the information content of the learned embeddings [8, 77] or via tricks such as stop-gradient [14] momentum update [23], asymmetries between the two encoder architectures [14, 23] and batch-wise or feature-wise normalization [14, 23, 77]. While the end goal of contrastive and non-contrastive training is the same, namely learning meaningful representation invariant to augmentations, they differ in how they prevent representation collapse. When the model collapses, the two encoders output always the same constant regardless of the input, hence bringing the loss to zero, but learning meaningless representations.
These invariance-based approaches have gained more popularity also in the molecular domain [68], exemplified by the work done in [61, 66]. The major limitation of the current invariance-based SSL approaches is the heavy reliance on data augmentations [4, 67], crucial for model performance, as they impose priors on which data transformations maintain the same semantic content. Meaningful data augmentations have been particularly challenging to create for graphs [18, 41, 45, 70, 83]. This difficulty is evident for molecules, where the slightest alteration, such as removing a single node or edge to create a positive augmentation, can result in a different molecule with different properties [61], hence leading to false positives. When trained in a contrastive fashion, this can also lead to the opposite case, false negatives [18]. Current approaches for molecular data augmentations rely on trial and error, extensive domain knowledge, and learning augmentations, all of which are resource-intensive [68]. Furthermore, it is also shown that performance on downstream tasks highly depends on which augmentation techniques were utilized for pretraining [41]. In [67], the authors avoid augmentations by using as one of the two encoders, a perturbed version of the other encoder, to obtain similar embeddings, and they achieve good results. In AFGRL [41], they avoid augmentations by discovering, for each node in the original graph, nodes that can serve as positive samples via k-nearest-neighbour. Another possible way to prevent and ease the need for complex data augmentations, is to utilize a 'Joint Embedding Predictive Architecture' [39] (Figure 1.C.II) to perform a reconstruction task in the latent space. This approach will be the focus of this study.

2.4 Joint Embedding Predictive Architecture

The 'JEPA' term, coined for the first time in [39], includes a spectrum of different models, that share the fundamental feature of having joint encoders processing inputs x and y in parallel, and a predictor that predicts (in the latent space) the embedding $\mathbf{s}_y$ from the embedding $\mathbf{s}_x$ (the outputs of the encoders), potentially, but not necessarily conditioned on a latent variable z. A JEPA model, similarly to JEAs, can be trained to be invariant to a set of hand-crafted augmentations, (with notable examples being SimSiam [15] and BYOL [23], and BGRL [62] for graphs), hence not solving the data augmentation issue. In this scenario, the objective is the same as the one for JEAs, and the difference between the two architecture types lies only in the model design, namely the use of the predictor model component, which creates an asymmetry between the two joint networks. The addition of the predictor module is deemed advantageous for training purposes [15, 23, 62]. However, JEPAs can also be trained in a different fashion, without requiring data augmentations, exemplified by the very recent work in [4, 7, 9, 21, 24, 55, 58] in domains that span from image [4] and video [7] to optic flow estimation [9], audio signal [21], EEG signal processing [24], point clouds [55] and graphs [58]. From now on, in this paper the term JEPA will always refer exclusively to those JEPAs trained with this second modality which does not require data augmentations. In this case, instead of learning invariance to data augmentations, the model learns to reconstruct, in the latent space, a signal y (target) from a compatible signal x (context), using a predictor network conditioned on an additional latent variable z (e.g. positional information about y) to facilitate the reconstruction task [4]. One common approach to obtain compatible signals, that are predictive of each other, is via masking part of the input and learning to reconstruct the masked part y from the unmasked part x. This reconstruction task is conceptually similar to the one utilized in Generative Architectures (GAs) [28, 31] (Figure 1.C.III). However, a crucial difference between JEPAs and GAs, is that in JEPAs (as in JEAs) the loss function is applied in the latent space, not the input space. Differently from JEAs, JEPAs do not seek representations invariant to a set of hand-crafted data augmentations. Instead, JEPAs aim for representations that are predictive of each other in the latent space, when conditioned on additional information [4]. Hence, JEPAs act as a bridge between JEAs and GAs, taking the best from both methods: predicting in the latent space facilitates the learning of semantically-rich representations, avoiding the need to predict and reconstruct every (potentially noisy) detail of the input space, leading often to overfitting [39, 58]. At the same time, JEPAs avoid the need for complex data augmentations, needed for JEAs. Figure 1.C shows the three different architectures. The idea of a reconstruction task in the latent space, used with JEPAs, was introduced in data2vec [5] and later adapted to various domains under the name JEPA, as described earlier.

2.4.1 JEPAs for graphs

The high-level idea of JEPAs for graphs is to mask a part of the graph (i.e. a subgraph) and predict the embedding of the masked part (the target subgraph), from the embedding of another larger part of the graph (the context subgraph), as visible in Figure 1.D. A variation of this strategy has already been implemented in LaGraph [69], where latent graph prediction is also performed via masking. However, differently from our work, in [69] they operate at the node level, they mask the features but not the topology, they do not employ a predictor module, and they optimize an additional objective based on the reconstruction of the input space, similarly to autoencoders. A similar study [18] improves on LaGraph by predicting the masked nodes with extended contexts that enable nodes of smaller importance, to have more weight. In [58], the authors propose for the first time a JEPA tailored for graphs, Graph-JEPA, introducing an initial study. The method builds upon the work done in [4] for images and addresses some of the challenges present when working in the graph domain. However, several parts of their methodologies are left partially unexplored, leaving room for further exploration and experimentation with JEPAs in the graph domain, even more so for molecular graphs. Sections 3.4 and 3.5 of Methodology will detail and motivate the novelty of our proposed JEPA and how it differs from [58]. The changes span most of the components necessary for the architecture, including spatial partitioning techniques, sizes of subgraphs, the specific choice of the encoder types, and training settings (e.g. via regularization). While some changes are more general and regard broadly all JEPAs for graphs, others are tailored for polymer molecular graphs.

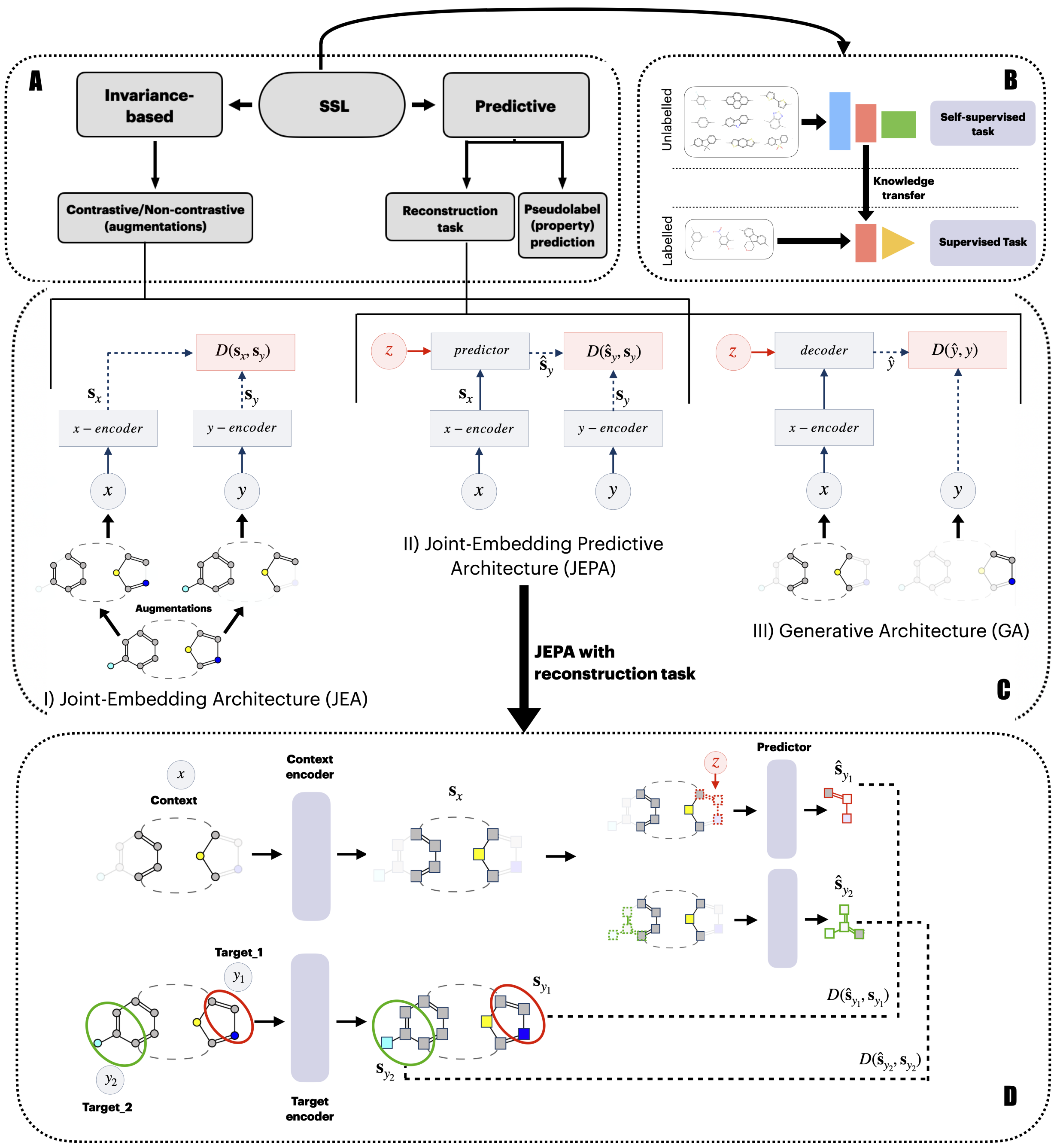


Figure 1: **A:** A simplified overview of different approaches to SSL, for a more complete taxonomy refer to [45, 70]. **B:** A simplified illustration of how SSL is used, and the knowledge transfer step via initializing the finetune model's (red rectangle) weights with those learned throughout pretraining. **C:** Common architectures for SSL and the corresponding inputs in the polymer graph domain (Section 3.1). *I:* Joint Embedding Architectures learn to output similar embeddings (from positive data augmentations) for compatible inputs x, y and dissimilar embeddings for incompatible inputs. *II:* Joint Embedding Predictive Architectures can be used both with invariance-based and reconstruction-based SSL approaches. In the former case, they work similarly to JEAs, in the latter case, they learn to reconstruct the embedding of a signal y from a compatible signal x, using a predictor network that is conditioned on an additional variable z to facilitate prediction. *III:* Generative Architectures learn to directly reconstruct in the input space a signal y from a compatible signal x, using a decoder network, conditioned on an additional latent variable z to facilitate reconstruction. **D:** An overview of JEPA with a reconstruction task, described in Section 2.4 and more thoroughly in Section 3.5. The inputs are polymer graphs. The latent variable z is the positional encoding of the target subgraph (see Section 3.5.3).

3 Methodology

This section introduces the methods employed throughout the work. Firstly, the graph representation of polymers (Section 3.1) and the datasets (Section 3.2) are introduced. Secondly, the graph neural network message passing mechanism utilized is discussed in Section 3.3. Subsequently, the specific pretraining method and the architecture used are described in Sections 3.4 and 3.5. Finally, Section 3.6 details the training procedure.

3.1 Polymer graphs

Polymers are large molecules composed of repeating units called monomers. The material properties of polymers, such as mechanical properties including tensile strength and elasticity, depend on different structural characteristics of polymers. These characteristics include the monomer chemistries, namely the atoms and bonds constituting the monomer, the monomer stoichiometry, representing the ratio in which one monomer is present in the polymer, in case multiple monomers are present, and chain architecture (e.g. alternating, random, block). Figures 2 and 3 provide an intuition of these structural features. Furthermore, the stochastic nature of polymerization reactions usually leads to a polymer material consisting of an ensemble of macromolecules (i.e. polymers) with varying chain lengths.

Given the recurrent chain-like structure of polymers and the stochastic nature of the reactions that create them, representing polymers as graphs constitutes a significant challenge. Merely representing them via their single monomer units, would neglect important structural characteristics, such as their chain architecture, which can significantly alter the material properties of the polymer. In [2], Aldeghi and Coley propose a novel graph representation, capable of capturing both the monomers' chemistry and structural information, exemplified in Figure 2. Specifically, it represents the monomer units' chemistry of a polymer p with a graph G, where the nodes (V) and edges (E) represent the atoms and bonds respectively. Each node v and each edge e_{vu} (connecting node v to node u) are associated with feature vectors, describing the atom and bond characteristics: $\mathbf{f}_v$ and $\mathbf{f}_{e_{vu}}$ where $\mathbf{f}_v \in F^V$ and $\mathbf{f}_{e_{vu}} \in F^E$, with F^V and F^E being the node and edge feature space of the graph, respectively. Tables 1 and 2 show the full list of the features used.

The polymer stoichiometry is represented via node weights: each node v has a weight w_v, where $w_v \in W^V$ and $w_v \in (0, 1]$, that reflects in which ratio the monomer to which the node belongs to, is present in the full polymer. Finally, the chain architecture is represented via stochastic edges, namely edge weights (W^E), describing the average structure of the repeating unit, capable of discriminating between alternating, block, and random chain structures. Hence, each edge e_{vu} is associated with a weight w_{vu} with $w_{vu} \in (0, 1]$ indicating the probability of a bond being present, between node v and node u, in each repeating unit. The edges are directed, to represent a wider range of polymers, including graft copolymers and polymers with terminal ends. From the description provided above we can define the polymer graph as $G(p) = \{V, E, F^V, F^E, W^V, W^E\}$.

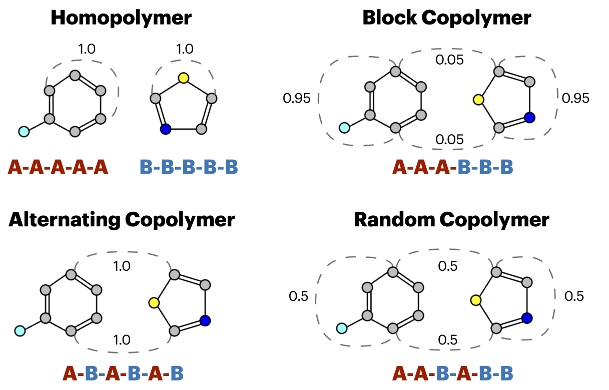


Figure 2: Graph representations of selected polymer topologies are shown, with stochastic edges depicted by dashed lines. Different bead colors indicate various atom types. Below each graph, an example polymer sequence for each topology is provided as text, where A and B represent two distinct monomers. Homopolymers, as well as alternating, random, and block copolymers, can be described as undirected graphs with some stochastic edges, where the probability of the edge reflects the frequency with which the bond is present in the polymer chain. However, following [2], we use directed edges to potentially represent other polymer types (e.g., oligomers with termini, graft copolymers) that are not present in the datasets considered. In the figure, we have omitted directed edges for simplicity.

3.2 Data

3.2.1 Aldeghi and Coley Dataset

The dataset utilized for pretraining is the one from [2], built upon the polymer space defined in [6], depicted in Figure 3, representing conjugated polymers as photocatalysts for hydrogen production. The

Table 1: Node features: all encoded as one-hot vectors except for atom mass. Chiral Tag, Degree, Hybridization, Charge, and #Hs include an additional category for uncommon values.

Node feature	**Description**	**Feature Size**
Atom Type	Type of atom	101
Chiral Tag	Type of chirality	5
Degree	Number of neighbours	7
Is Aromatic	Whether the atom is in an aromatic system	1
Hybridization	Type of hybridization, i.e. sp, sp^2	6
Charge	Formal charge of the atom	6
#Hs	Number of bonded hydrogen atoms	6
Mass	Atom mass	1

Table 2: Edge features: all encoded as one-hot vectors. Stereo has an extra category for uncommon values.

Edge feature	**Description**	**Feature Size**
Bond Type	single, double, triple, or aromatic	4
Conjugated	Whether the bond is conjugated	1
Is in Ring	Whether the bond is part of a ring	1
Stereo	Stereochemistry: none, any, E/Z, or cis/trans	7
No Bond	For robustness, should always be 0	1

dataset contains 42,966 polymers, composed of nine A monomers and 862 B monomers. The polymers are classified according to three distinct chain architectures: alternating, random, and block. Additionally, they are further distinguished by three stoichiometry ratios: 1:1, 1:3, and 3:1. The dataset considers all possible combinations between A and B monomers, and the different stoichiometries and architectures. For random and block copolymers, all stoichiometry ratios were considered, while for alternating copolymers only the 1:1 ratio. The dataset includes electron affinity (EA) and ionization potential (IP) as molecular properties, used as labels for the property prediction task. EA measures a polymer's ability to accept electrons, and IP measures its ability to release electrons. Both are important for using polymers as photocatalysts in hydrogen production. The dataset is divided into three parts: one part (40%) is used for self-supervised pretraining, where the labels are not utilized. The second part (40%) is kept for supervised finetuning (downstream task), although smaller subsets (e.g. 0.4%) are used for testing data-scarce scenarios (Section 4). This split emulates a realistic scenario where pretraining and finetuning do not occur on the same data, as usually pretraining data lacks labels. The remaining 20% of the dataset is used for validation, both for pretraining and finetuning, when finetuning on this dataset. 5-fold cross-validation [60] is used to validate the results.

3.2.2 Diblock Copolymers dataset

Given the constrained polymer chemical space represented by the dataset described in Section 3.2.1, the simplicity of the prediction tasks related to the nature of the predicted properties and the similarities between pretrain and finetune datasets, the model was tested also on a more complex downstream task. For this scope, the Diblock Copolymers dataset [3] was utilized. The dataset, used also in [2], provides the phase behavior of 49 diblock copolymers (Figure 4). It details the observed copolymer phases (lamellae, hexagonal-packed cylinders, body-centred cubic spheres, a cubic gyroid, or disordered) across various relative volume fractions and molar masses, totaling 4780 entries. Since each entry might correspond to multiple phases, the task is framed as a multi-label classification problem with five labels, corresponding to the phases that can be observed. For this scenario, pretraining occurred on 80% of the Aldeghi and Coley dataset (with 5-fold cross-validation), and finetuning was then performed on the Diblock dataset, considering different dataset sizes. Given that some phases are more common than others, resulting in the five labels being imbalanced, we use stratified sampling to maintain the same label distribution across train and test splits, specifically the stratification is based on the distribution of the first phase of each polymer, as done in [2].

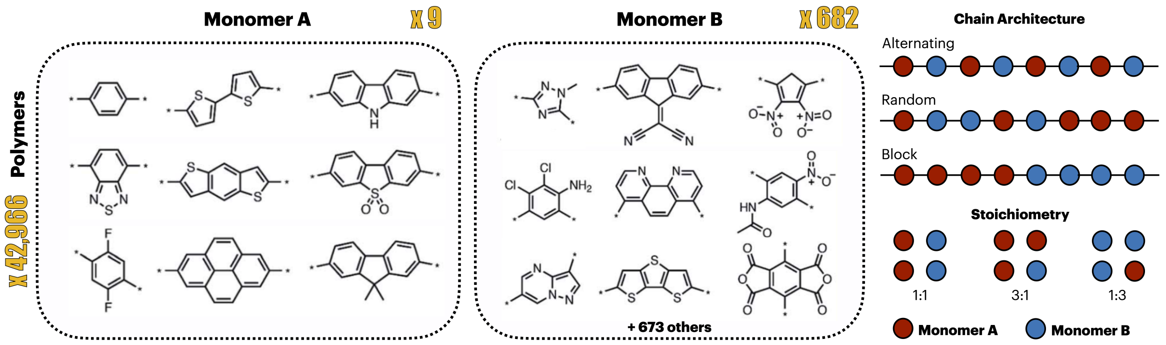


Figure 3: The Aldeghi and Coley dataset comprises 42,966 copolymers characterized by various building blocks, chain architectures, and stoichiometries. There are nine monomers classified into group A, all of which are also included in group B. Each monomer is represented by a distinct bead color in the figure, with each bead being a monomer. The figure is adapted from [2].

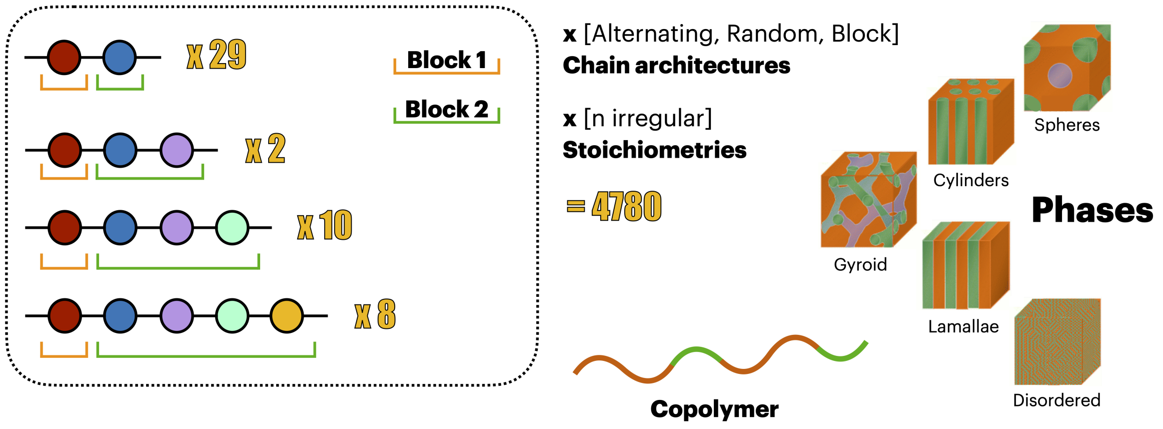


Figure 4: The Diblock Copolymers dataset represents 4780 diblock copolymers, where each block consists of one or multiple monomers, ranging from one to four. Different chain architectures and irregular stoichiometries are considered. The prediction task is a multi-label classification that predicts the self-assembly behavior (phase) of the polymers, specifically how the diblock copolymers fold and shape in certain environmental conditions.

3.3 Node-centred Weighted Directed Message Passing Neural Network

Weighted Directed Message Passing Neural Network (WDMPNN) is a type of graph neural network, introduced by Aldeghi and Coley in [2], based on DMPNN [74], to effectively learn on the polymer graphs described in Section 3.1. In this work, we devise and utilize a variant of the WDMPNN based on node-centred message passing, instead of the original edge-centred one. The node-centred model achieves the same performance as the edge-centred one, but it is more intuitive, more efficient, and less cumbersome to implement and work with. A full comparison and motivation behind this design choice can be found in Appendix A.

Similarly to common GNNs, the core mechanism of the network is based on localized node-centred convolutions, namely message passing between neighbouring nodes, with messages being the outputs of nonlinear functions (transformations) that take the nodes and edges features as input. Hence, the node features are iteratively transformed and updated based on the received messages, resulting in learned node embeddings. A representation of the full graph (i.e. polymer) is then obtained by pooling (e.g. average)

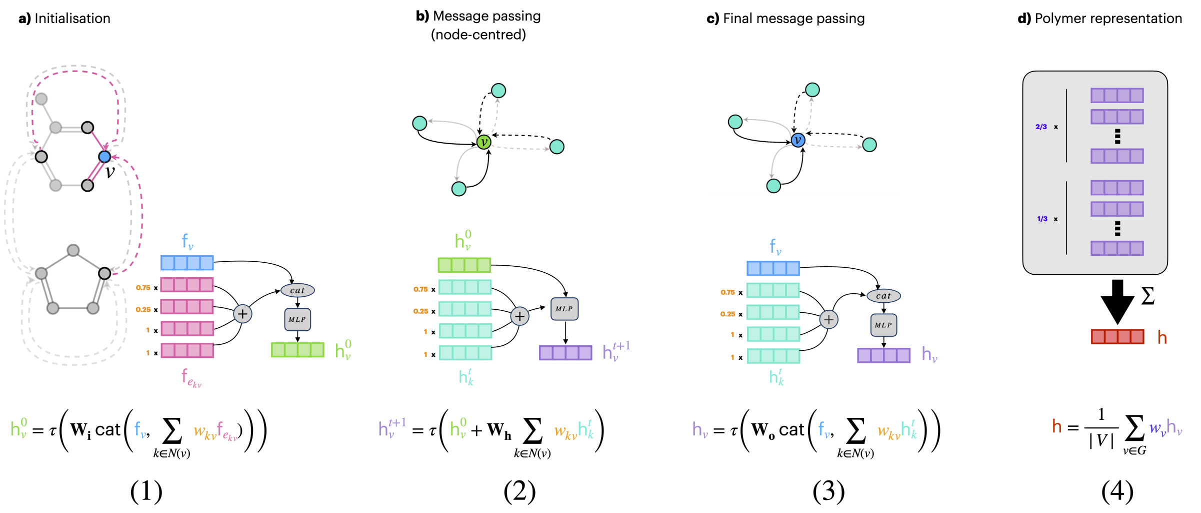

a) Initialisation
b) Message passing (node-centred)
c) Final message passing
d) Polymer representation
v
f_v
$0.75 \times$
$0.25 \times$
$1 \times$
$1 \times$
$f_{e_{kv}}$
cat
MLP
h_v^0
h_k^t
h_v^{t+1}
h_v
$2/3 \times$
$1/3 \times$
Σ
h
$h_v^0 = \tau\left(W_i \operatorname{cat}\left(f_v, \sum_{k \in N(v)} w_{kv} f_{e_{kv}}\right)\right)$
$h_v^{t+1} = \tau\left(h_v^0 + W_h \sum_{k \in N(v)} w_{kv} h_k^t\right)$
$h_v = \tau\left(W_o \operatorname{cat}\left(f_v, \sum_{k \in N(v)} w_{kv} h_k^t\right)\right)$
$h = \frac{1}{|V|} \sum_{v \in G} w_v h_v$
(1)
(2)
(3)
(4)

Figure 5: The four steps to learn polymer representation via the node-centred WDMPNN. Figure adapted from [2].

all nodes (i.e. atoms) embeddings. Figure 5 shows the four steps to learn polymer representations (also called fingerprints) via the node-centred WDMPNN. In the first step, considering a node v, its initial embedding $\mathbf{h}_v^0$ is computed as shown in Equation (1):

$$\mathbf{h}_v^0 = \tau\left(\mathbf{W_i} \operatorname{cat}\left(\mathbf{f}_v, \sum_{k \in N(v)} w_{kv} \mathbf{f}_{e_{kv}}\right)\right) \tag{1}$$

where $N(v)$ represents the neighbours of v, and the $\sum$ summation takes a weighted average of the incoming edge features, based on the edge weights w_{kv} of the edges arriving in v from the respective neighbouring k node. 'cat' denotes feature concatenation, $\mathbf{W_i}$ is a learnable weight matrix, and τ is a non-linear activation function. Concatenating node features and incoming edge features leads to an expressive initial representation $\mathbf{h}_v^0$. In the next step (Equation (2)), the node message passing mechanism is performed:

$$\mathbf{h}_v^{t+1} = \tau\left(\mathbf{h}_v^0 + \mathbf{W_h} \sum_{k \in N(v)} w_{kv} \mathbf{h}_k^t\right) \tag{2}$$

with $\mathbf{W_h}$ being a learnable weight matrix. The $\sum$ summation takes a weighted average of the neighbouring nodes features, based on the edge weights w_{kv} of the edges arriving in v from the respective k node. A residual connection adds the initial embedding ($\mathbf{h}_v^0$) to maintain important original information of the node. The operation is repeated T times, where T represents the number of message passing layers. The final node embedding $\mathbf{h}_v$ is obtained as shown in Equation (3):

$$\mathbf{h}_v = \tau\left(\mathbf{W_o} \operatorname{cat}\left(\mathbf{f}_v, \sum_{k \in N(v)} w_{kv} \mathbf{h}_k^t\right)\right) \tag{3}$$

with $\mathbf{W_o}$ being a learnable weight matrix.
Considering Equation (3) as an additional and final message passing step, the actual total number of message passing layers is $T+1$. Finally, the graph embedding $\mathbf{h}$, representing the polymer in the latent space, is obtained via a weighted mean of the node embeddings, as shown in Equation (4):

$$\mathbf{h} = \frac{1}{|V|} \sum_{v \in G} w_v \mathbf{h}_v \tag{4}$$

where $|V|$ represents the total number of nodes in the graph, and w_v weights each node based on the ratio (stoichiometry) of its monomer in the polymer. Subsequently, the learned polymer representation $\mathbf{h}$, can be utilized as input for any downstream task, for instance as input to a multi-layer perceptron that can be trained in a supervised fashion to predict a property.
In Appendix B, we will describe in full the hyperparameters utilized for the WDMPNN.

3.4 Spatial partitioning for JEPA

To find the input x and y, namely the context and target subgraphs, for the JEPA model, it is necessary to partition the graph G into subgraphs $\{G_1, G_2, \ldots, G_n\}$. Subgraphing is an essential component of JEPAs. We discuss it in detail in this section before describing JEPA itself, along with all other components and design choices, in Section 3.5. Splitting the graph into patches (i.e. subgraphs) can be done in different ways, via a subgraphing function $\mathcal{P}$. In Graph-JEPA [58], the authors propose subgraphing based on METIS [35] (see Section 3.4.2), and random subgraphing. They do not focus on any domain- or graph-specific technique, leaving room for future work. Based on intuition and previous studies, we devise a list of requirements to be respected when subgraphing on polymer graphs:

1. In the case of a copolymer, the context subgraph should include elements from both monomers: predicting a part of monomer B, if monomer B is missing from the context is not possible,
2. Every edge and every node should be in at least one subgraph [30, 58] to include full global information and full input representation,
3. The context patch (subgraph) should be larger, hence more informative, than the targets patches (subgraphs) we are trying to predict from the context [4],
4. The target subgraphs should have minimal overlap with the context subgraph [4, 58] to make the prediction task less trivial,
5. The context and targets subgraphs should change at every training loop to prevent overfitting [4, 58],
6. For every directed edge e_{vu} in a subgraph, include also the edge e_{uv}, to comply with WDMPNN (described in Section 3.3).

Based on these principles the following spatial partitioning methods are proposed:

3.4.1 Motif-based subgraphing

A specialized subgraphing approach tailored to the chemical domain, can generate subgraphs that represent chemical fragments, for example functional groups or small molecular subunits. This partitioning method ensures that the subgraphs carry meaningful chemical information, potentially enhancing the model's understanding of chemical patterns, inter-node relationships, and structural motifs.
Several methods are available to partition a graph into motifs, the most commonly used being BRICS (Breaking of Retrosynthetically Interesting Chemical Substructures) [16]. BRICS is an algorithm designed to decompose chemical compounds into smaller, synthetically relevant fragments by focusing on functional groups and other key substructures. It systematically breaks bonds to generate fragments that retain specific chemical functionalities. Other methods include RECAP [42], various methods based on BRICS to find smaller motifs [76, 79, 82], and methods based on motifs learning [81].
To generate valid motifs, we need to work at the monomer level, as the stochastic edges in the polymer graphs make those graphs invalid molecules and they cannot be processed by software like RDKIT to find the motifs with the aforementioned methods. Given that monomers have a limited number of atoms, it is necessary to utilize a method that fragments the molecule in very small motifs, to avoid having a single motif representing the full monomer, as this would not comply with the subgraphing requirements listed before. After testing different methods, we found that the approach used in [82] is the most suitable for our use case. Their method further improves BRICS by adding two additional rules:

1. breaking the bond where one end atom is in a ring while the other end not
2. selecting non-ring atoms with three or more neighbouring atoms as new motifs and breaking the neighbouring bonds.

The first rule reduces the number of ring variants and the second rule breaks the side chains [82].
Empirical tests showed that the algorithm avoids edge loss by producing subgraphs with minimal overlap. The only case when edge loss occurs is when the algorithm yields subgraphs made of a single node, in that case, a 1-hop expansion is performed. After finding motifs, some motifs are joined to form the context subgraph. In particular, to ensure Requirement 1, we form the context by joining two subgraphs that belong to different monomers and are connected by an intermonomer bond. Then, the context is potentially randomly expanded with other subgraphs, to respect the size requirements. The remaining motifs are available as possible targets. If the remaining number of motifs is smaller than the number of required targets, we create new subgraphs by randomly selecting a non-context node and including it and its 1-hop neighbours in a new subgraph. The frequency

of this operation depends on the sizes of subgraphs, and the number of targets. This is done also for all the other subgraphing methods (Sections 3.4.2 and 3.4.3) if necessary.

3.4.2 METIS subgraphing

METIS is a popular subgraphing algorithm [35], utilized also in [30, 58]. METIS works by partitioning a graph into a predefined number of clusters p while minimizing the edge cut between these clusters and maximizing the within-cluster links. Its widespread use is due to its low computational cost and the quality of the produced subgraphs.

METIS produces non-overlapping partitions. To avoid the loss of edge information, we perform a 1-hop expansion on each partition. This expansion results in larger subgraphs and increased overlap between them. The procedure would also comply with the requirements listed in Section 3.4, so it would require randomly joining subgraphs into a unique context subgraph based on the size requirements, and on the monomer they belong to. The number p of clusters for the METIS algorithm is a hyperparameter, on which the sizes and number of the clusters depend. Through empirical experimentations, we found seven to be an optimal choice for p.

This method, as the one based on random walks (RWs) (Section 3.4.3), potentially produces subgraphs that are not meaningful from a chemical perspective, potentially limiting their semantic content.

3.4.3 Random-walk subgraphing

To compare the usefulness of using domain knowledge (Section 3.4.1) and a proper clustering algorithm (Section 3.4.2), we propose to test a random-walk-based method, suited for the polymer graphs described in Section 3.1. Random walks allow more flexibility, more control over subgraph sizes, and tend to produce more diverse subgraphs at each iteration, given their non-deterministic nature, hence perfectly complying with Requirement 5.

The method begins by creating the context subgraph with an edge connecting atoms from different monomers. In a random-walk fashion, it then iteratively adds elements, alternating between monomers. For the target subgraphs, it starts random walks from the non-context nodes, without constraints on which monomer the nodes added belong to. To avoid edge loss we do a 1-hop expansion of the target subgraphs.

3.5 Joint Embedding Predictive Architecture for graphs

The high-level idea of JEPAs for graphs is to partition the graphs into patches (i.e. subgraphs) and predict (reconstruct) the embedding $\mathbf{s}_y$ of a masked patch (y) of the graph, the target subgraph, from the embedding $\mathbf{s}_x$ of another, larger, part of the graph, the context subgraph (x).

The first step required is to divide the graph into subgraphs, which we thoroughly discussed in Section 3.4. The following steps are described in detail in the next sections, starting from a discussion on the sizes of subgraphs in Section 3.5.1.

3.5.1 Subgraph sizes

In addition to the method by which subgraphs are obtained, it is important to focus on the sizes of the context and target subgraphs. In I-JEPA [4], the authors show the importance of the context and target image patch sizes, showing a significant difference in performance for different sizes. While Graph-JEPA [58] does not consider the subgraph sizes, we explore this aspect of methodology for JEPAs in the graph domain. The importance of subgraph sizes is evident also from an intuitive perspective: for the context to predict the target, the context should contain enough information to make the prediction possible, with additional (i.e. positional) information about the target provided by the latent variable z. This is stated clearly also in the original JEPA paper [39], mentioning how $\mathbf{s}_y$ should be easily predictable from $\mathbf{s}_x$. This cannot be achieved if the context subgraph is too small and consequently non-informative. For instance, if the context contains only three carbon atoms of the full polymer, predicting a specific target (e.g. a functional group) is very complex since three carbon atoms represent very general, non-specific information, that does not allow to precisely predict a missing part. This agrees with Requirements 1 and 3 of Section 3.4. In the experiments, we will extensively investigate the role of subgraph sizes, by testing the model with context subgraph sizes ranging from 20% to 95% of the full graph. For the target subgraphs, sizes will span from 5% to 20%. The results are reported in Sections 4.1.1 and 4.1.2.

3.5.2 Models

We devise two different versions of the model, visible in Figure 6a and Figure 6b. While the end goal of the architectures is the same, they differ in the way the context and target embeddings $\mathbf{s}_x$ and $\mathbf{s}_y$ are obtained. Each of the two architectures presents advantages and disadvantages:

I

The structure of the first model (Figure 6a), is relatively similar to the one utilized in Graph-JEPA [58], itself based on the work of [30], for the way they obtain the graph representation in the latent space. An initial GNN, in our case the WDMPNN, takes separately as input all the subgraphs found by the subgraphing algorithm, plus the context subgraph formed by joining some of those subgraphs. The GNN produces subgraph embeddings by pooling the learned subgraph's node embeddings. All subgraph embeddings (including those of the subgraphs used to form the context subgraph, but excluding the context subgraph), are subsequently fed to the target encoder, E_{target}, implemented with a transformer block. The output of the transformer is new subgraph embeddings, incorporating not only local information, but also global one, obtained from the other subgraphs, via the self-attention mechanism of the transformer. Some of the new subgraph embeddings are used as targets, hence we call them target embeddings.

When the model is trained with momentum update, the E_{target} parameters need to be updated via an exponential moving average (EMA) of the $E_{context}$ parameters. Therefore, the context subgraph embedding is also fed to a transformer context encoder $E_{context}$ to produce a new context subgraph embedding. The context encoder is necessary only to maintain an exact symmetry between the two encoders. This is also the case if we use vicReg with weight sharing (Section 3.5.6). The input of $E_{context}$ is the single context subgraph embedding, hence, the self-attention mechanism of the transformer is not attending any other subgraph embeddings. It is important that the context subgraph does not 'see' the other masked parts (i.e. targets) of the graph, otherwise the prediction task would be trivial.

To maintain positional information while working on subgraphs, we utilize two different positional embeddings, described in detail in Section 3.5.3.

At this stage, we obtained the context and target embeddings, and we can utilize the predictor network h_ϕ, to predict the target embedding $\mathbf{s}_y$ from the context embedding $\mathbf{s}_x$. We will describe this step in Section 3.5.4 since it is the same for both model versions. The target subgraphs are selected randomly from the pool of all subgraphs. The number of targets is arbitrary and is the subject of the experiment in Section 4.1.3. Importantly, the subgraphs utilized to create the context subgraph cannot be picked as targets, to avoid the prediction task to be trivial. Regarding the architecture, it is important to note two key differences between ours and Graph-JEPA. Firstly, the choice of GNN, in which we utilize the WDMPNN for polymer graphs. Secondly, in our model, the target encoder takes as input all subgraphs (spanning the full graph nodes and edges) found by the subgraphing algorithm, including the subgraphs utilized to form the context subgraph. Differently, in Graph-JEPA only the subgraphs selected as targets are given as input. Since the target subgraphs are a (relatively small) subset of all subgraphs, this hinders the exchange of global information, since the target encoder is missing a major part of that global information (i.e. the full graph) from the input it is given. In data2vec [5], one of the seminal works for JEPAs, the authors extensively discuss the importance of the exchange of global information, defined in the paper as 'contextualization' of the targets. Contextualization makes predicting the target from the context more feasible since the targets will contain more unique, polymer-specific global information. As described in Section 3.5.1, making the prediction task feasible, is crucial to effectively train JEPAs [39]. In the successful I-JEPA [4], the authors also contextualize the targets, by feeding all the image patches to the target encoder, not only the image patches selected as targets.

II

The second architecture (Figure 6b) proposed, compared to the first, removes the initial GNN (i.e. WDMPNN) encoder and substitutes the transformer blocks utilized for the context and target encoders, with two GNNs (i.e. WDMPNNs) respectively. The changes make the model less complex and less parameterized, compared to the first one, hence making Model II computationally more efficient. The input of the context encoder remains the context subgraph, while the input of the target encoder is the full graph. This contrasts with Model I, where the target encoder receives as input all the subgraphs found by the partitioning algorithm. Subgraph-level training (e.g. with a transformer) is used mostly to tackle problems such as over-squashing in large graphs [30]. Additionally, in the case of JEPAs, training on subgraphs with a transformer as E_{target}, allows full tar-

get contextualization, since all subgraphs attend each other. However, given the modest size of the polymer graphs (20-30 nodes), we hypothesize that the model can contextualize the targets also via a message passing mechanism (i.e. WDMPNN), which substitutes self-attention (i.e. transformer). This is possible if the number T of message passing layers is large enough, allowing each node to exchange information with nodes T-hop apart. We also consider the over-squashing problem not particularly relevant in the molecular domain, given the small graph sizes. Additionally, training E_{target} on the full graph, allows to maintain positional information more easily. The changes in Model II, remove the need for an intermediate subgraph embedding (the one produced by the initial GNN in Model I), and reduces significantly the number of parameters of the model. As for Model I, $E_{context}$ outputs the context subgraph embedding $\mathbf{s}_x$. To obtain a target subgraph embedding, $\mathbf{s}_y$, we pool the node embeddings learned by E_{target} on the full graph, for the nodes that belong to that target subgraph. Similarly to Model I, the subgraphs utilized to form the context subgraph cannot be used as targets to be predicted.
The next step regards predicting the target subgraph, which we explain in 3.5.4, but first we describe how we include positional information in the next section.

3.5.3 Positional information

We employed two types of positional encoding (PE): at the node level and at the subgraph (patch) level. The node-level PE increases the expressiveness of the model. The goal is to maintain node positional information when working with subgraphs. It is especially useful for Model I where we work with subgraphs for the target encoder. The second type of PE is via subgraph (patch) level positional tokens used by the predictor to ease the prediction of the targets from the context, by providing the context subgraph with positional information of the target.

Node PE. To include node PE, we linearly transform the original node features and sum it with a PE vector, itself obtained via a linear transformation of the original PE features. The original PE features are obtained via random-walk structural encoding (RWSE) [19], as done in [30], originally used by [51]. The RWSE for a node v is defined as:

$$\mathbf{p}_v = (\mathbf{M}_{ii}, \mathbf{M}_{ii}^2, \dots, \mathbf{M}_{ii}^k), \quad \mathbf{p}_v \in \mathbb{R}^k, \tag{5}$$

where $\mathbf{M}^k = (\mathbf{D}^{-1}\mathbf{A})^k$ is the random-walk diffusion matrix of order k and i is the index of node v in the adjacency matrix. $\mathbf{M}_{ii}^k$ represents the probability of node v returning to itself after a k-step random walk. RWSE ensures a unique node representation by considering each node's distinct k-hop topological neighbourhood. It captures how nodes are embedded within the larger graph topology, indicating the likelihood of a node reconnecting with itself over multiple steps, reflecting its position in the overall graph. By doing so, the position is expressed in a global and consistent manner for each node. We set k equal to 20, as done in [30, 58]. The feature vector for v is:

$$\mathbf{h}^v = \mathbf{p}_v\mathbf{T} + \mathbf{f}_v\mathbf{U} \tag{6}$$

with $\mathbf{T} \in \mathbb{R}^{k\times d}$ and $\mathbf{U} \in \mathbb{R}^{f\times d}$ being learnable matrices.

Patch PE. Patch PE is used as the variable z to help the predictor h_ϕ (Section 3.5.4) predict the target embedding from the context embedding by giving to the context information about the target. We compute the patch PE from the original graph adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ (with N the number of nodes) and the subgraphs $\{V_1, \dots, V_P\}$ extracted by the subgraphing algorithm (Section 3.4), as done in [30]. Specifically, we obtain relative positional information via the coarsened adjacency matrix $\mathbf{A}_P \in \mathbb{R}^{P\times P}$ over the subgraphs, with P being the number of subgraphs, and $\mathbf{A}_{P_{ij}}$ defined as:

$$\mathbf{A}_{P_{ij}} = |V_i \cap V_j| = \mathrm{Cut}(V_i, V_j), \tag{7}$$

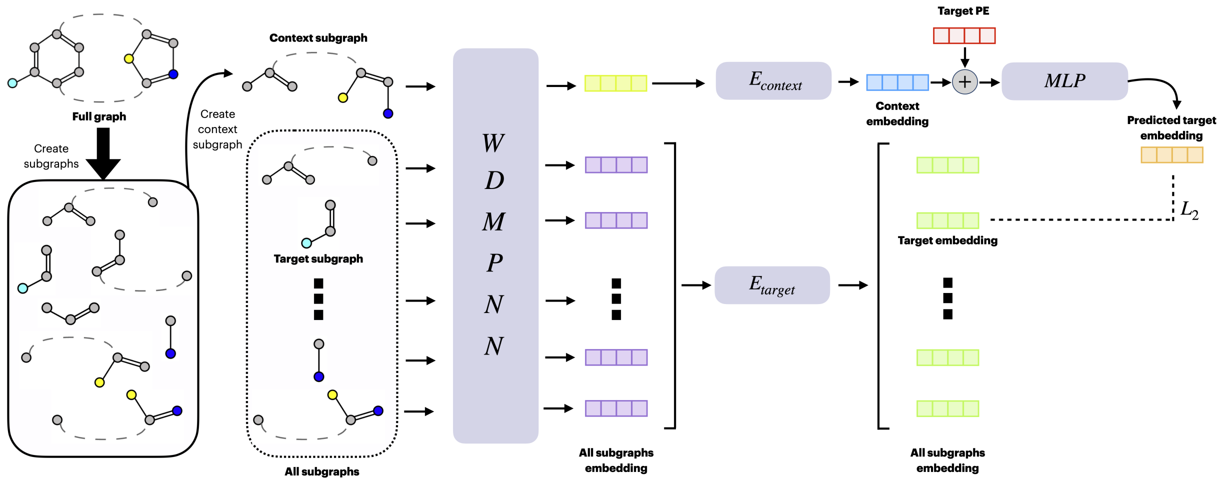
where $\mathrm{Cut}(V_i, V_j) = \sum_{k\in V_i}\sum_{l\in V_j}\mathbf{A}_{kl}$ is the graph cut operator that counts the number of connecting edges between subgraph V_i and subgraph V_j. The positional encoding $\tilde{\mathbf{p}}_i \in \mathbb{R}^{\tilde{k}}$ at the subgraph level, is then extracted similarly to the node-level one, namely via RWSE over the coarsened adjacency matrix $\mathbf{A}_P$. We set $\tilde{k}$ equal to 20, as done in [30].

3.5.4 Predictor

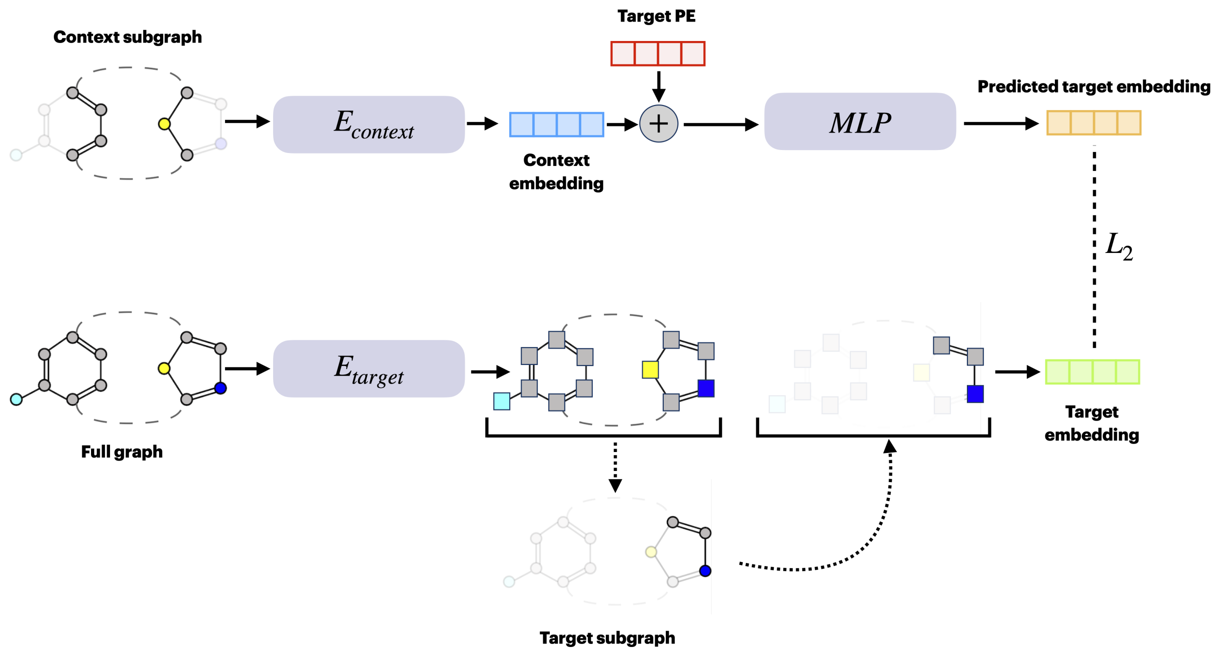
Given the output of the context encoder, $\mathbf{s}_x$, we wish to predict the m target subgraphs representations $\mathbf{s}_y(1), \dots, \mathbf{s}_y(m)$. To that end, for a given target subgraph embedding $\mathbf{s}_y(i)$, the predictor h_ϕ takes as input $\mathbf{s}_x$ summed with the linearly transformed target subgraph positional token $\tilde{\mathbf{p}}_i$:

$$\hat{\mathbf{s}}_y(i) = h_\phi\left(\mathbf{s}_x + \tilde{\mathbf{p}}_i\tilde{\mathbf{T}}\right) \tag{8}$$

with $\tilde{\mathbf{T}} \in \mathbb{R}^{\tilde{k}\times d}$. The predictor outputs the predicted target embedding $\hat{\mathbf{s}}_y(i)$. Since we wish to make predictions for m target blocks, we apply our predictor m times, obtaining predictions $\hat{\mathbf{s}}_y(1), \dots, \hat{\mathbf{s}}_{\mathbf{y}}(m)$. In practice, the predictor h_ϕ is implemented via a MLP (see Appendix B for details on the hyperparameters).



(a) Model version I



(b) Model version II

Figure 6: The two JEPA model versions proposed. An in-depth description of the models can be found from Section 3.5.2 to Section 3.5.5. In Figure (a), we illustrate the subgraphing process on the left. This process also occurs for Model II (Figure (b)), but it is not shown due to space limitations. For Model I, we represent the subgraphs using only the subgraph nodes, whereas for Model II, we depict the nodes and edges not belonging to subgraphs as faded (transparent). Both representations convey the same information and are used according to the available space.

3.5.5 Loss

For each entry, the loss is the average L_2 distance (Mean Square Error (MSE)) between the m predicted target subgraph representations and the m true target subgraph representations:

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{\mathbf{s}}_y(i) - \mathbf{s}_y(i)\right)^2 \tag{9}$$

The parameters of the predictor h_ϕ, and context encoder $E_{context}$, are learned through mini-batch stochastic gradient descent (SGD), via backpropagation. The parameters of the target encoder, E_{target}, are updated via an exponential moving average of the context encoder parameters. This is done to prevent representation collapse, as discussed in Section 2.3. To avoid backpropagation on E_{target} we utilize the stop-gradient operation. Updating E_{target} parameters with $E_{context}$ ones, requires the two encoders to have identical structure. We also experiment with a different learning strategy based on regularization, described in the next section 3.5.6.

3.5.6 Regularization with vicReg

We decided to compare our JEPA trained with stop-gradient and momentum update, with another non-contrastive training strategy based on regularization via vicReg [8] to prevent collapse. VicReg, short for variance-invariance-covariance regularization, is a regularization method that can be utilized to train JEAs and JEPAs to prevent collapse, by maximizing the amount of information in the embeddings produced by the encoders. The way collapse is prevented is more efficient than using a contrastive method, which requires large batch sizes to contain many negative augmentions. It is also more elegant and intuitive than the tricks utilized in other non-contrastive methods like momentum update, stop-gradient, and batch- or layer normalization, whose functioning is still not fully understood. Differently from strategies like momentum update, it does not require the encoders to share weights, nor the inputs to be the same, hence potentially allowing multimodal learning [8]. However, for our work the inputs are both graphs, and we share weights, as it performed better, as well as in [8]. As the name mentions, the loss is composed of three components, the first being invariance $s(\hat{\mathbf{S}}_y, \mathbf{S}_y)$, where we denote $\hat{\mathbf{S}}_y = [\hat{\mathbf{s}}_{y_1}, \dots, \hat{\mathbf{s}}_{y_n}]$ and $\mathbf{S}_y = [\mathbf{s}_{y_1}, \dots, \mathbf{s}_{y_n}]$ as the two batches composed of n vectors of dimension d, representing the predicted and true target embeddings respectively. $\hat{\mathbf{S}}_y$ is obtained as $\hat{\mathbf{S}}_y = h_\phi(\mathbf{S}_x, \tilde{\mathbf{P}}_y)$ where h_ϕ represents the predictor network (Section 3.5.4), $\mathbf{S}_x = [\mathbf{s}_{x_1}, \dots, \mathbf{s}_{x_n}]$ the context embeddings, and $\tilde{\mathbf{P}}_y = [\tilde{\mathbf{p}}_{y_1}, \dots, \tilde{\mathbf{p}}_{y_n}]$ the target patches positional information as described earlier. For simplicity, we assume that m, the number of targets, is one, so that the number of targets and context match, however this can be different (Section 3.5.4). The invariance loss term accounts for the prediction precision in predicting the target embedding, from the context one. Mathematically, it is the MSE loss defined in Equation (9). In addition to the invariance loss, we have a variance loss term, implemented via a hinge loss that allows to maintain the standard deviation (over a batch) of each variable (i.e. feature dimension) of the embedding above a given threshold. This term forces the embedding vectors of samples within a batch to be different. It is important to use the standard deviation and not directly the variance. If we use $S(x) = Var(x)$ in the hinge function, the gradient of S with respect to x becomes close to 0 when x is close to $\bar{x}$ [8]. Mathematically the variance loss v for x is shown in Equation (10):

$$v(\mathbf{Z}_x) = \frac{1}{d}\sum_{j=1}^{d}\max(0, \gamma - S(\mathbf{z}_x^j, \epsilon)), \tag{10}$$

importantly we do not apply the loss directly on $\mathbf{S}_x$ and $\mathbf{S}_y$, but on $\mathbf{Z}_x$ and $\mathbf{Z}_y$ where $\mathbf{Z}_x = f_\theta(\mathbf{S}_x)$, with f_θ representing an expander function (see Figure 7). In Equation (10), $\mathbf{z}_x^j$ represents the vector composed of each value at dimension j in all vectors in $\mathbf{Z}_x$, λ is a constant target value for the standard deviation, we fix it to one as in [8], and S is the regularized standard deviation:

$$S(x, \epsilon) = \sqrt{\text{Var}(x) + \epsilon} \tag{11}$$

where ϵ is a small scalar preventing numerical instabilities. Finally, the covariance loss term attracts the covariances (over a batch) between every pair of (centred) embedding variables (i.e. feature dimensions) towards zero. This term decorrelates the variables of each embedding and prevents an informational collapse in which the variables would vary together or be highly correlated. We define the covariance matrix of $\mathbf{Z}_x$ as:

$$C(\mathbf{Z}_x) = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{z}_{x_i} - \bar{\mathbf{z}}_x)(\mathbf{z}_{x_i} - \bar{\mathbf{z}}_x)^T, \tag{12}$$

where $\bar{\mathbf{z}}_x = \frac{1}{n}\sum_{i=1}^{n}\mathbf{z}_{x_i}$, then the the covariance loss is obtained via:

$$c(\mathbf{Z}_x) = \frac{1}{d}\sum_{i \neq j}[C(\mathbf{Z}_x)]_{i,j}^2 \tag{13}$$

The final loss is the weighted sum of the three terms:

$$L(\hat{\mathbf{S}}_y, \mathbf{S}_y, \mathbf{Z}_x, \mathbf{Z}_y) = \lambda s(\hat{\mathbf{S}}_y, \mathbf{S}_y) + \mu[v(\mathbf{Z}_x) + v(\mathbf{Z}_y)] + \nu[c(\mathbf{Z}_x) + c(\mathbf{Z}_y)] \tag{14}$$

with λ, μ, and ν being hyperparameters set to 25, 25, and one respectively, as suggested in [8].

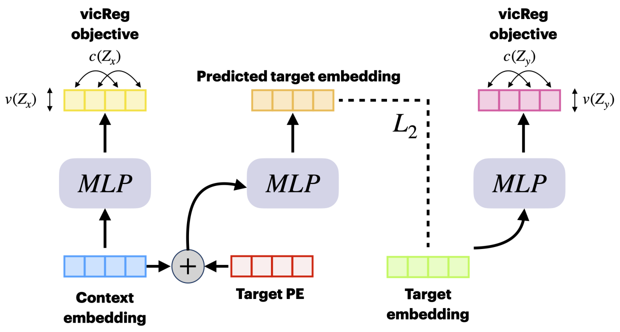


Figure 7: The vicReg objective.

3.5.7 Pseudolabel objective

In [22], namely the first study working on SSL for polymer graphs, as described in Section 2.2, the authors also work with the graph representation introduced in [2] and consequently with the WDMPNN. In their work, they observe significantly better performance when transferring not only the pretrained weights of the WDMPNN encoder but also the weights of the predictor (i.e. MLP) used for predicting the polymer molecular weight in a pseudolabel task. The pseudolabel predictor weights are later transferred to the predictor used for the downstream task, for instance the EA molecular property in the Aldeghi and Coley dataset. Motivated by these results, we decided to experiment with the pseudolabel objective also with our architecture for the Aldeghi and Coley dataset. We achieve that by predicting the molecular weight from the polymer fingerprint learned through the target encoder, as visible in Figure 8. The molecular weight is defined as:

$$M_w = w_{mono1} M_{mono1} + w_{mono2} M_{mono2} \tag{15}$$

with M_w being the polymer molecular weight, M_{mono1} and M_{mono2} being the molecular weights of the monomers, and w_{mono1} and w_{mono2} being the stoichiometry ratios. We utilize the MSE loss between the predicted molecular weight $\hat{M}_w$ and the true one M_w. Differently from [22], the pseudolabel task is done simultaneously with the JEPA training, not subsequently (as they do after the node-level task). We tested different weights α and β for the two losses (target embedding prediction and pseudolabel prediction, respectively), and we found $\alpha = \beta = 1$ to be optimal. We discuss and show the results of the experiment in Section 4.6, specifically in 4.6.1. The idea of utilizing an additional pseudolabel objective is in line with the idea of biasing a JEPA towards learning 'useful' representations, discussed in [39]. The idea is to bias the system towards representations that contain information that is relevant for the downstream task, via predicting a variable that can easily be derived from the data and is relevant to the task.

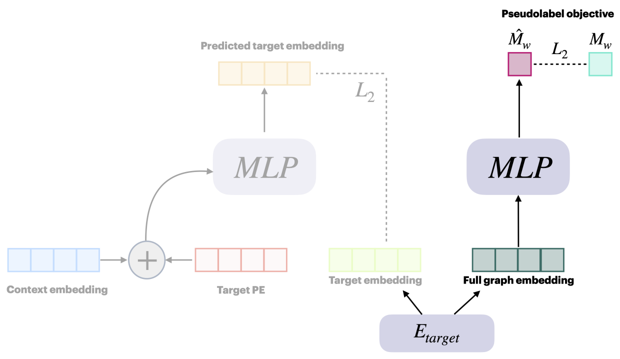


Figure 8: Pretraining with the additional pseudolabel objective that predicts the molecular weight from the polymer representation obtained via E_{target}.

3.6 Training procedure

The pretraining phase entails training the JEPA (Section 3.5). During this phase, the model is trained on 40% (17186 entries) of the Aldeghi and Coley dataset. After pretraining, the target encoder is utilized to obtain the polymer graph representation. This process uses self-supervised representations to capture the structural and chemical features of polymer molecules. For the downstream task (e.g. polymer property prediction), an MLP is employed on top of the target encoder (that outputs the polymer graph representations). Finetuning is done end-to-end, wherein not only the MLP but also the target encoder weights are updated during the optimization process. This allows the polymer graph representations to also finetune to the specific downstream task. For finetuning, we consider the two datasets described in Sections 3.2.1 and 3.2.2. For both datasets, we tested on different finetune data sizes to verify the pretraining effectiveness in different data availability scenarios (see Section 4.5). In Appendix B, we provide the implementation details and the hyperparameters used for JEPA and the finetuning MLP.

4 Results and Discussion

In this section, we discuss the results obtained from our experiments. We begin by examining subgraph sizes (Section 4.1) and subgraphing types (Section 4.2), followed by an analysis of different model architectures (Section 4.3). Subsequently, we compare the performance of various training strategies, specifically momentum update with stop-gradient compared to vicReg (Section 4.4). In Section 4.5, we test our model on the two datasets on the downstream task, testing different finetune data availability scenarios. In Section 4.6, we compare our proposed approach against existing methodologies (i.e. [22]). Finally, we compare our pretrained model performance against the one of a simpler model (i.e. random forest) in low-data scenarios (Section 4.7). We measure the efficacy of the pretraining strategy and certain model components or design choices by assessing the performance in a downstream task, where the model utilized for finetuning is initialized with the weights learned during pretraining. Each result reported is the averaged numerical result obtained by conducting 5-fold cross-validation (i.e. five runs). Each run featured a random finetune dataset, ensuring consistency across experiment settings by using the same five random seeds, hence testing each configuration on the same five finetune datasets. Each experiment regarding the model design, architecture, and subgraphing (Sections 4.1, 4.2, 4.3, 4.4) was conducted on 0.4% of the Aldeghi and Coley dataset, corresponding to 192 datapoints. The rationale behind this choice is that it represents a realistic scenario where labelled lab-derived data is very scarce and expensive to obtain. In addition, the pretraining should show better results in the most data-starving cases, highlighting the impact of certain model design choices more clearly. Moreover, finetuning on a small dataset is computationally more efficient, allowing us to run a greater number of experiments. Finally, 0.4% is one of the dataset sizes used in in [22], allowing us to compare our method to theirs (Section 4.6). We did all the experiments (except the one in Section 4.3), on Model II since it performed better, was computationally cheaper, and more novel compared to Graph-JEPA.

For the Aldeghi and Coley dataset, we report the same metrics as in [2] and [22], namely the R2 (coefficient of determination) and the Root Mean Square Error (RMSE) in predicting the property (e.g. EA). The former metric provides a measure of how well the observed labels are predicted by the model, based on the proportion of total variation of outcomes explained by the model, hence measuring how well the regression model fits the observed data. R2 ranges between zero and one, where higher values correspond to better performance. The RMSE, is calculated by taking the square root of the mean of the squared differences between predicted and observed values, providing an indication of the average deviation of the predicted values from the observed ones. For the Diblock Copolymers dataset, we measure performance as in [2], via the area under the precision-recall curve (AUPRC) [56]. AUPRC is a single scalar value that quantifies the overall performance of a multi-label classification model. Higher AUPRC values indicate better model performance, with a maximum value of one indicating perfect precision-recall balance.

4.1 Subgraph sizes

4.1.1 Context subgraph size

The first study conducted involved an investigation into the impact of context subgraph sizes. Intuitively, as described in the Methodology (Section 3.5.1), we hypothesize that the context should be large (i.e. informative) enough to make the prediction of the target possible, but if too large, it would largely overlap with the target, hence making the prediction task easier. The experiment utilized momentum update and stop-gradient, three target subgraphs, we used random-walk subgraphing, and target subgraphs sized approximately 15% of the total graph. We chose random-walk subgraphing because differently from motif- or metis-based subgraphing, allows us to control more precisely the subgraph sizes.

The results (Table 3), firstly show that pretraining consistently enhanced performance, irrespective of the specific context size scenario. Notably, pretraining contributed to increased stability and robustness of results, evident in the standard deviation metrics, thereby highlighting its efficacy in mitigating the influence of the specific distribution of the finetune training dataset. Furthermore, we find that a context size of 60% (relative to the full graph size) seems to be optimal, confirming our initial hypothesis. However, the model seems to be relatively robust to context sizes, as shown by the small performance gap between different context sizes. The most significant decline in performance is noticeable with a context size of 20%. Notably, a very small context size makes the target prediction task considerably more challenging, hindering the capability of the model to learn. We speculate that this observation is valid not only for polymers but broadly applies also to small molecules and other graph-based structures. Similar trends were observed in the image domain in [4], making the size of the context an important element for JEPA training (see Section 3.5.1), irrespective of the input type. While we cannot assert that 60% is the optimal context size universally for all input types and molecule sizes, we are confident that an interval of 50% to 75% is effective. We hypothesize that larger molecule sizes might require slightly more context information due to the greater number of atoms and substructures and the potentially increased distance between the context and target subgraphs. However, as stated in Section 3.5.2, contextualizing the targets should help ease the prediction task and mitigate the issue of distance between context and target, thereby limiting the necessary increase in context size.

Table 3: Results of the experiments on the context subgraph size, on the Aldeghi and Coley dataset, predicting the EA property. 60% results the optimal size for the context subgraph, corresponding to around 10-15 atoms of the full molecule (20 - 25 atoms).

Context size	**R2 ↑**	**RMSE ↓**
No pretrain	*0.46 ± 0.15*	*0.44 ± 0.06*
20%	0.56 ± 0.07	0.39 ± 0.03
40%	0.60 ± 0.06	0.37 ± 0.03
60%	$\mathbf{0.65 \pm 0.03}$	$\mathbf{0.35 \pm 0.02}$
80%	0.62 ± 0.07	0.37 ± 0.03
95%	0.61 ± 0.05	0.37 ± 0.02

4.1.2 Target subgraph size

Similar to the investigation into context sizes, we also explored the impact of different target subgraph sizes. Employing the same experimental setup as detailed in the context size analysis (Section 4.1.1), we used a context size of 60%, identified as optimal.
The results (Table 4) indicate that the model exhibits robustness to changes in target subgraph sizes, with performance remaining relatively stable across different size scenarios. However, there seems to be an optimal range between 10% to 15%. We speculate that subgraph sizes outside of this range may potentially impede the prediction task, either by oversimplifying or overly complicating it, depending on the extent of overlap between the context and target subgraphs, as well as the specific characteristics of the atom embeddings being predicted. Similarly to the suppositions made in Section 4.1.1, we believe that this conclusion also holds true for small molecules. As previously observed (see also Section 3.5.1), the size of the patches (i.e., subgraphs) is an important factor for effective JEPA training across various domains and similar results were reported in [4] in the image domain. In particular, we hypothesize that a target size range of 10% to 20% should be effective across different molecular sizes and input types. For smaller molecules, the target size might lean towards the higher end of this range, while for larger molecules, it might be closer to 10%. This should avoid the extreme cases where the target is either too small (a single node) or too large (several molecular substructures together).

Table 4: Results of the experiments on the target subgraph size, on the Aldeghi and Coley dataset, predicting the EA property. 10% results as the optimal size for the target subgraph, corresponding to around 2-3 atoms of the full molecule (20 - 25 atoms).

Target size	**R2 ↑**	**RMSE ↓**
No pretrain	*0.46 ± 0.15*	*0.44 ± 0.06*
5%	0.61 ± 0.07	0.37 ± 0.03
10%	$\mathbf{0.66 \pm 0.02}$	$\mathbf{0.35 \pm 0.01}$
15%	0.65 ± 0.03	0.35 ± 0.02
20%	0.63 ± 0.03	0.36 ± 0.02

4.1.3 Number of targets

Finally, we explored the impact of the number of targets being predicted, utilizing the same experimental setup as for the context and target subgraph size experiments, with the optimal configuration found: 60% context size and 10% target size. The results (Table 5) demonstrate that the model is not highly sensitive to the number of targets. While no clear trend is discernible, likely due to the stochastic nature of the training process, we find that a single target yields the best performance and stability.
Having a lower number of targets increases the likelihood of encountering different target subgraphs at each epoch, which can enhance model generalization. Conversely, a higher number of targets may lead to the model predicting the same or very similar targets at each iteration, potentially resulting in overfitting.

Table 5: Experimental results for the number of targets, on the Aldeghi and Coley dataset, for EA.

n. targets	**R2 ↑**	**RMSE ↓**
No pretrain	*0.46 ± 0.15*	*0.44 ± 0.06*
1	$\mathbf{0.67 \pm 0.01}$	$\mathbf{0.34 \pm 0.01}$
2	0.64 ± 0.03	0.36 ± 0.01
3	0.66 ± 0.02	0.35 ± 0.01
4	0.65 ± 0.05	0.35 ± 0.02
5	0.61 ± 0.04	0.37 ± 0.02

4.2 Subgraphing type

Utilizing the same experimental settings, with the optimal configuration identified as 60% context size, 10% target size, and a single target predicted, we evaluated the impact of different subgraphing strategies, described in detail in Section 3.4.
We observed (Table 6) that they all perform similarly, with a slight advantage for the random-walk one, demonstrating the best performance and stability. Interestingly, the motif-based method, which leverages domain knowledge to produce chemically meaningful subgraphs, exhibits slightly lower performance. We hypothesize that each subgraphing method carries its own set of advantages. While motif-based subgraphing generates chemically meaningful subgraphs, it tends to produce a relatively small number of subgraphs, in a deterministic fashion, potentially limiting model generalization by increasing the likelihood of encountering similar or identical subgraphs (both context and target ones) throughout training. On the other hand, the METIS algorithm, while also producing consistent subgraphs at each epoch, generates on average a higher number of subgraphs compared to the motif-based approach, introducing more variability across epochs. Finally, random-walk subgraphing generates different subgraphs at every epoch, thanks to the stochastic nature of the subgraphing process. Moreover, it allows for greater control over subgraph sizes, a factor previously highlighted as relatively influential in model performance. In conclusion, we find that variations in subgraphs and subgraph sizes play a more crucial role in determining model performance than the chemical meaningfulness of the subgraphs themselves.

Table 6: Results of the experiments on the subgraphing type, on the Aldeghi and Coley dataset, predicting the EA property.

Subgraphing	R2 ↑	RMSE ↓
No pretrain	*0.46 ± 0.15*	*0.44 ± 0.06*
Motif-based	0.63 ± 0.05	0.36 ± 0.02
Metis	0.67 ± 0.04	0.34 ± 0.02
RW	**0.67 ± 0.01**	**0.34 ± 0.01**

4.3 Model type

We compared the two model versions proposed, utilizing the best configuration identified: 60% context size, 10% target size, one target subgraph predicted, and random-walk subgraphing. The key finding (Table 7) is that pretraining significantly enhances performance and stability for both model versions, as evidenced by an increase of more than 0.20 in the R2 metric value in both cases. Model I demonstrates notably poorer performance compared to Model II. There are several potential reasons for this discrepancy. Firstly, all previous experiments were optimized for Model II, including subgraph sizes, the number of targets, and subgraphing type, potentially giving it an advantage. Additionally, Model I is inherently more complex, featuring a combination of a WDMPNN initial encoder and transformers encoders (see Section 3.5.2). This complexity may require more data to effectively learn, particularly in the context of the very data-scarce scenario (192 graphs) tested here. Another factor to consider is that Model I trains the target encoder on small subgraphs, potentially not aligning well with the polymer dataset utilized, which consists of copolymers made up of different monomer units. This approach could lead to a loss of positional information crucial for accurate predictions. In contrast, Model II does not encounter this issue as it trains the target encoder on the full original polymer graph.

Table 7: Results of the experiments on the model type, on the Aldeghi and Coley dataset, predicting the EA property.

Model	R2 - No pretrain ↑	R2 - Pretrain ↑
I	0.27 ± 0.22	**0.52 ± 0.03**
II	0.46 ± 0.15	**0.67 ± 0.01**

4.4 Effects of regularization

We conducted a comparison of different training strategies aimed at preventing representation collapse, specifically exploring momentum update with stop-gradient (i.e. EMA) and vicReg regularization. These evaluations were conducted under the optimal settings of 60% context size, 10% target size, one target subgraph predicted, and random-walk subgraphing. Our findings, summarized in Table 8 and Table 9, indicate that the two strategies perform similarly. In particular, on the Aldeghy and Coley dataset, EMA demonstrates slightly better performance and stability. For the Diblock dataset, vicReg shows a slight advantage. Both approaches prove to be very effective in preventing representation collapse and improving performance. One point against EMA is that all hyperparameter optimizations conducted thus far were carried out with EMA, suggesting that the model was finetuned specifically for EMA training. By examining the representation space of the polymers (Figure 9), we observe that vicReg is able to achieve better clustering, especially when coloring by monomer A. This improvement in clustering can be attributed to vicReg's regularization objective, which aims to maximize the information content of the learned polymer representations. However, the improved clustering in the representation space does not directly translate into significantly better downstream performance for the datasets considered, indicating that there is no simple direct correspondence between clustering quality and predictive performance.

Table 8: Results of the experiments on the training strategy, comparing regularization (with vicReg) and momentum update (i.e. EMA). Comparison on the Aldeghi and Coley dataset, predicting the EA property.

Training	**R2** $\uparrow$	**RMSE** $\downarrow$
No pretrain	*0.46 ± 0.15*	*0.44 ± 0.06*
VicReg	0.65 ± 0.05	0.35 ± 0.02
EMA	$\mathbf{0.67 \pm 0.01}$	$\mathbf{0.34 \pm 0.01}$

Table 9: Results of the experiments on the training strategy, comparing regularization (with vicReg) and momentum update (i.e. EMA). Comparison on the Diblock dataset (4% of the dataset (191 graphs)).

Training	**AUPRC** $\uparrow$
No pretrain	*0.36 ± 0.03*
VicReg	$\mathbf{0.41 \pm 0.04}$
EMA	0.40 ± 0.02

4.5 Downstream tasks

Thus far, our experiments focused on finding optimal design choices, and model components. With the best configuration found, we evaluated the pretraining effectiveness in different finetune data availability scenarios. We hypothesize that the utility of pretraining diminishes as supervised training data increases, although this relationship is contingent upon the dataset and downstream task. Figures 10 and 11 present the performance results for the Aldeghi and Coley datasets, respectively, concerning the prediction tasks of Electron Affinity and Ionization Potential. We test from a data scenario of 0.4% (192 datapoints) to a scenario of 24% (10.311 datapoints). Consistent with our hypothesis, the pretrained model demonstrated performance improvements in scenarios with the most limited data availability. This enhancement was particularly evident up to a data size of 2.4% (1,031 datapoints). However, beyond this threshold, the benefits of pretraining plateaued and, in some cases, led to negative transfer, as observed in EA prediction for data sizes of 3.2% and 4%. For this reason and for clearer visualization, in Figures 10 and 11 we report results only up to the 4% scenario. This suggests that the supervised data available is sufficient for learning, rendering the transferred pretraining knowledge redundant or potentially noisy. In practice, a small change in the R2 value (e.g. ±0.01) does not significantly impact molecular design tasks, such as property prediction. However, in cases where the R2 increase is higher (i.e. 0.4% and 0.8% scenarios), the improvement does have a meaningful effect on the task.
We also conducted experiments on different data scenarios for the Diblock dataset, as visible in Figure 12. We tested from a training data scenario of 191 datapoints (4%) to 3.824 datapoints (80%). The latter scenario corresponds to training on the full dataset, retaining 20% for testing, as done in [2]. The results consistently demonstrated an improvement in performance in the pretrained scenario, even for the more data-rich scenarios. This underscores the effectiveness of the proposed pretraining strategy and its capability to generalize outside of the training distribution. In fact, the pretraining dataset represents a different polymer chemical space compared to the finetuning dataset, suggesting that the knowledge acquired during pretraining is not overfitting or memorizing the training distribution (i.e. chemical space) but rather learning valuable chemical knowledge. Furthermore, the observation that pretraining enhances performance even in the full dataset scenario indicates the strategy's potential utility for knowledge transfer purposes in scenarios where data is more abundant.

4.6 Comparison to other SSL tasks

We compared the effectiveness of our pretraining strategy to the only other SSL pretraining method [22] designed for the polymer graphs we use, working on the same dataset [2]. In their work [22], they employ two SSL tasks: one at the node level, masking nodes and edges and learning to predict them, and the other at the graph level, predicting a pseudolabel corresponding to the molecular weight of the polymer, derived from the monomers' weights. They test both tasks separately and together, and they discover that pretraining via both tasks proves to be the most

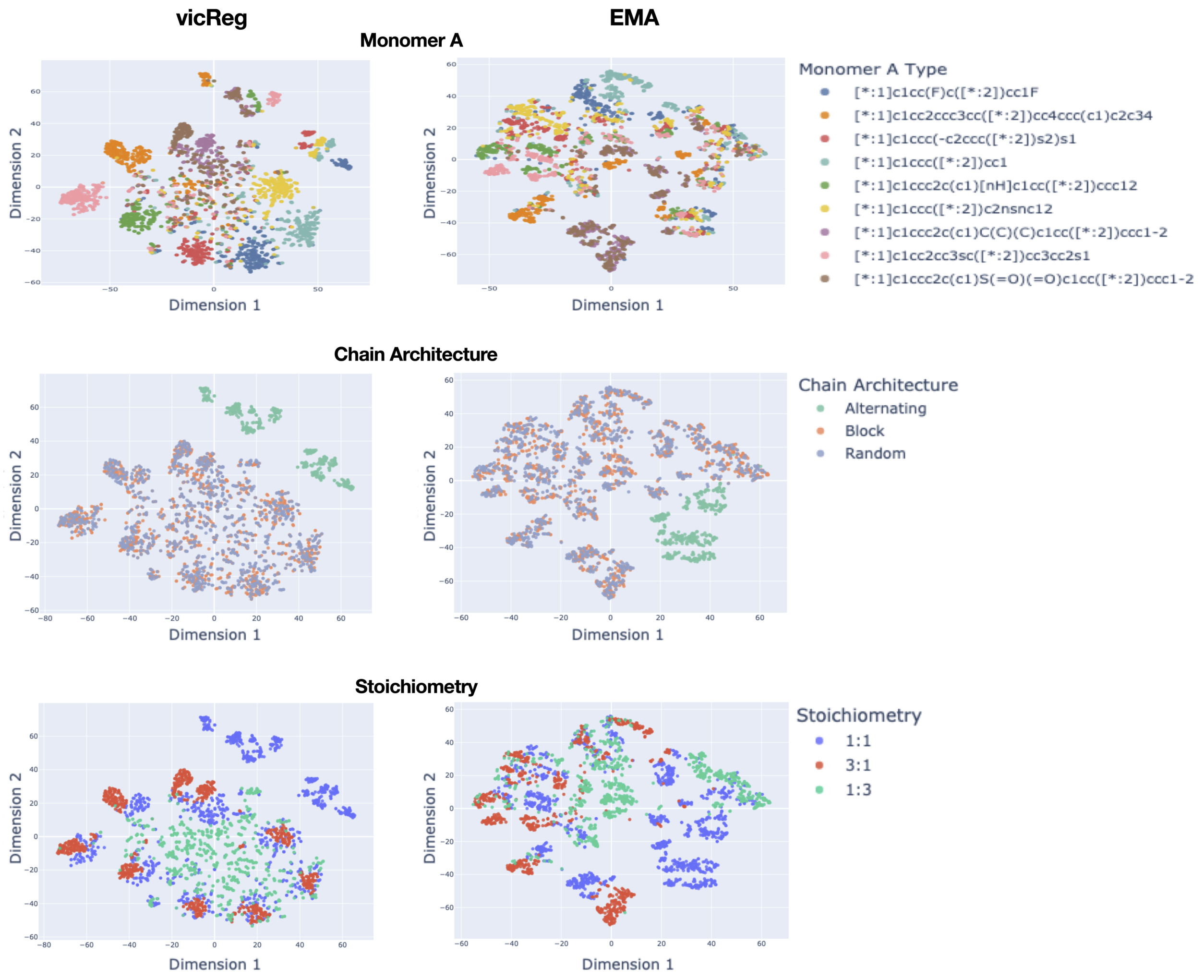


Figure 9: The t-SNE visualization of the embedding space learned for the polymers via vicReg (left) and EMA (right), colored (from top to bottom) by monomer A, chain architecture, and stoichiometry. Despite both approaches achieve good clustering, vicReg clustering is stronger, especially for monomer A. For Chain Architecture and Stoichiometry, both training strategies cluster effectively. The clustering by ’Alternating’ chain architecture is very strong, given that it is only associated to the 1:1 stoichiometry, while block and random chain architectures have a larger variety of three different stoichiometries.

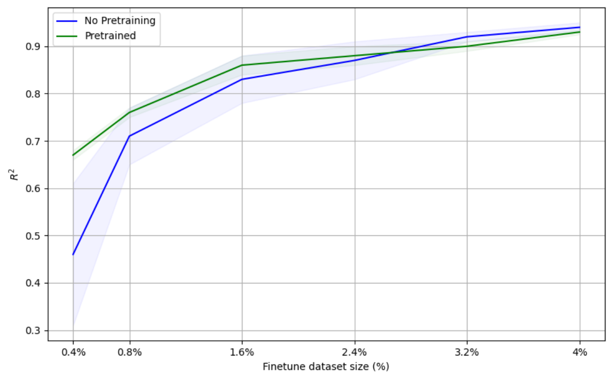


Figure 10: Effectiveness of our pretraining strategy on the Aldeghi and Coley dataset, for different finetune dataset sizes, predicting the EA property.

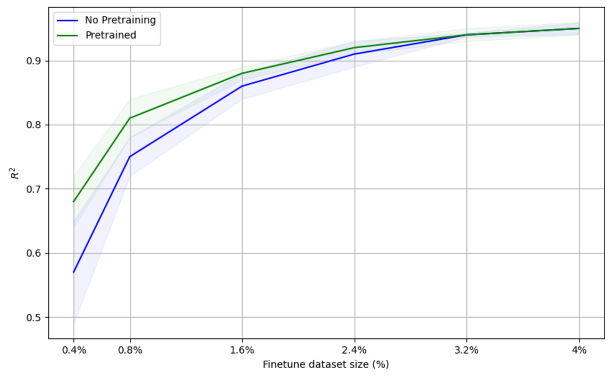


Figure 11: Effectiveness of our pretraining strategy on the Aldeghi and Coley dataset, for different finetune dataset sizes, predicting the IP property.

effective. This result aligns with findings in the literature [32] that SSL on graphs works better when using both node-level and graph-level tasks together. Figure 13 shows the result of the experiment. As visible, their pretraining strategy seems more effective, yielding better performance. However, some experimental factors differed between our experiments and theirs, particularly in our use of different pretraining and finetuning datasets at every run, utilizing cross-validation. In contrast, the other study always uses the same datasets across runs. Additionally, we use node-centred message passing (Section 3.3), while they utilize the original edge-centred method. Interestingly, their method achieves a significant performance boost when also transferring the weights of the fully connected layers used to predict the molecular weight. This suggests that the knowledge obtained while learning to predict the weight is particularly relevant. However, this relevance might be specific to the downstream task considered (i.e. the properties being predicted), in our case the Aldeghi and Coley dataset for EA prediction.

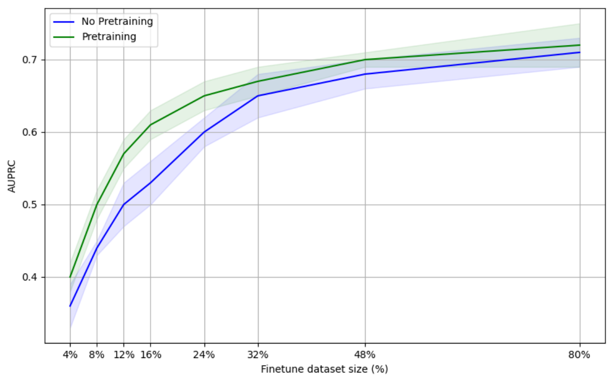


Figure 12: Effectiveness of our pretraining strategy on the Diblock dataset for different finetune dataset sizes.

4.6.1 Pseudolabel objective

We decided to experiment with the pseudolabel objective also with our architecture, by predicting the molecular weight from the polymer fingerprint learned through the target encoder, as described in Section 3.5.7. The pseudolabel objective significantly improves performance, similarly to what was shown in [22]. We included the results of our model performance with the pseudolabel objective in Figure 13. The improved result via the pseudolabel objective, is in line with the idea of biasing a JEPA towards learning 'useful' representations, discussed in [39]. The idea is to bias the system towards representations that contain information that are relevant for the downstream task, via predicting a variable that can easily be derived from the data.

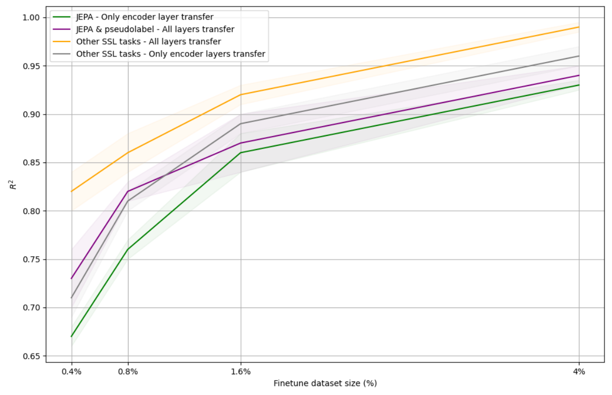


Figure 13: Comparison between our pretraining strategy and the SSL tasks from [22] on the Aldeghi and Coley dataset, for different finetune dataset sizes, predicting the EA property. We include both the scenarios when only the WDMPNN encoder weights are transferred, and the scenario when also the pseudolabel predictor weights are transferred.

4.7 Comparison to a simpler model in low-data regimes

As our pretraining is effective only in low-data regimes, we compared the performance of the pretrained WDMPNN with that of a simpler model, a random forest. Typically, simpler ML models outperform complex ones in data-scarce scenarios. The WDMPNN generates a polymer representation (fingerprint) through pooling node embeddings learned during training, which is then used as input for the MLP predictor. In contrast, the random forest model utilizes Extended-Connectivity Fingerprints (ECFP) [53] of size 2048 and radius 2, computed with RDKit [38]. Although this comparison is not entirely fair since the WDMPNN must learn the representation from the limited available data, ECFPs remain an easily accessible tool that can be utilized effectively. Furthermore, this experiment evaluates the pretraining strategy's capability to learn polymer fingerprints. The results show that the random forest model consistently outperforms the pretrained WDMPNN in the low-data regimes. This trend is evident in Figure 14, where the random forest model exhibited a clear advantage over the pretrained WDMPNN. Notably, in the smallest data regime (0.4%), the random forest model achieved a notable increase of 0.20 in the R2 score. This advantage diminishes as the dataset size increases.
Similar trends were observed for the Diblock dataset (Figure 15). Given that our pretraining strategy demonstrates effectiveness primarily in low-data regimes, and considering the superior performance of random forest models in such scenarios, we advise against the adoption of our proposed method for the specific datasets analyzed. This recommendation is based also on the significantly higher training costs associated with the WDMPNN and JEPA models compared to the random forest.

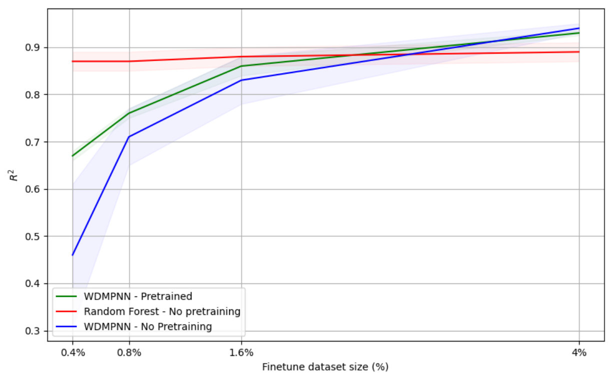


Figure 14: Comparison between our pretraining strategy and a random forest model on the Aldeghi and Coley dataset, predicting the EA property, for different finetune dataset sizes.

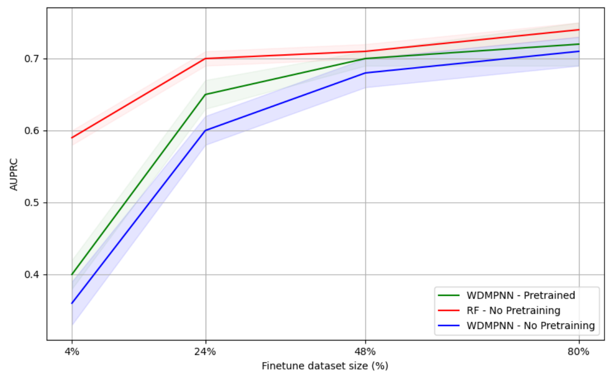


Figure 15: Comparison between our pretraining strategy and a random forest model on the Diblock dataset, for different finetune dataset sizes.

5 Conclusion and limitations

In conclusion, we introduced a novel self-supervised pretraining strategy for polymer molecular graphs, leveraging JEPAs. Our study stands as one of the initial efforts in exploring JEPAs for molecular graph-related tasks, thus enriching the understanding and analysis of this new architectural family in this domain, specifically for polymers. Through our experiments, we demonstrated the efficacy of our pretraining approach in significantly enhancing performance, particularly in scenarios where finetune data is very limited. However, we also showed how other SSL approaches [22] or a simpler model (Section 4.7) outperform our method, in the two downstream tasks considered. Interestingly, our pretraining strategy exhibits generalization capabilities, as evidenced by its effectiveness on a dataset representing a different chemical space from the one utilized for pretraining (Section 4.5). While our findings show considerable progress, our work has several limitations. Firstly, we acknowledge the absence of comparisons with a broader range of other self-supervised approaches (benchmarks), which limits the understanding of the effectiveness of our approach. Related to this, our approach was tested on only two finetune datasets. These limitations stem from our decision to work on polymers and utilize the graph representation introduced by Aldeghi and Coley [2]. Most self-supervised approaches in the literature require to be adapted to work with this specific polymer graph representation. Additionally, very few polymer datasets provide all the necessary information (i.e. stoichiometry and chain architectures) required for the graph representation. Currently, the only SSL study working on the same dataset is [22], which we discussed in Section 4.6. For future work, we propose extending our investigation to include other polymer datasets for finetuning, for instance by using a simpler graph representation not requiring chain and stoichiometry information. It would also be interesting to pretrain the model on a larger and more diverse dataset, compared to the approximately 17,000 polymers used in our study. This dataset is not only relatively small but also limited in chemical diversity, comprising only nine different atom types and polymers with only nine different A monomers. Additionally, future work could test the devised methodology for small molecules, for instance on the ZINC dataset [33], to explore its performance in different chemical contexts. We could also extend testing to other benchmark datasets representing various domains, providing a more comprehensive evaluation of the proposed JEPA as a self-supervised approach for graphs.

Another limitation of our work is the lack of a systematic and thorough hyperparameter optimization search, such as a grid search. We empirically tested various hyperparameter settings and used the best settings found as a starting point for our experiments. Some of our experiments, such as those on subgraph sizes and the number of targets, optimized parameters related to model components and design choices. However, this greedy optimization led to a finetuning of the model choices for Model II and the other settings being gradually decided. In the future, more work, optimization, and testing could be done also for Model I, which can effectively contextualize the targets via self-attention and remains a promising alternative. Finally, for subgraphing, we encourage future work to explore subgraphing on line graphs [20], for instance, using METIS. This approach could prevent edge loss while minimizing subgraph overlap, thus better satisfying the subgraphing requirements.

References

[1] Roshan A. Patel, Carlos H. Borca, and Michael A. Webb. Featurization strategies for polymer sequence or composition design by machine learning. *Molecular Systems Design & Engineering*, 7(6):661–676, 2022.

[2] Matteo Aldeghi and Connor W. Coley. A graph representation of molecular ensembles for polymer property prediction. *Chemical Science*, 13(35):10486–10498, 2022.

[3] Akash Arora, Tzyy-Shyang Lin, Nathan J Rebello, Sarah HM Av-Ron, Hidenobu Mochigase, and Bradley D Olsen. Random forest predictor for diblock copolymer phase behavior. *ACS Macro Letters*, 10(11):1339–1345, 2021.

[4] Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-Supervised Learning from Images with a Joint-Embedding Predictive Architecture, April 2023.

[5] Alexei Baevski, Wei-Ning Hsu, Qiantong Xu, Arun Babu, Jiatao Gu, and Michael Auli. Data2vec: A general framework for self-supervised learning in speech, vision and language. In *International Conference on Machine Learning*, pages 1298–1312. PMLR, 2022.

[6] Yang Bai, Liam Wilbraham, Benjamin J Slater, Martijn A Zwijnenburg, Reiner Sebastian Sprick, and Andrew I Cooper. Accelerated discovery of organic polymer photocatalysts for hydrogen evolution from water through the integration of experiment and theory. *Journal of the American Chemical Society*, 141(22):9063–9071, 2019.

[7] Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mido Assran, and Nicolas Ballas. V-jepa: Latent video prediction for visual representation learning. 2023.

[8] Adrien Bardes, Jean Ponce, and Yann LeCun. VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning, January 2022.

[9] Adrien Bardes, Jean Ponce, and Yann LeCun. Mc-jepa: A joint-embedding predictive architecture for self-supervised learning of motion and content features. *arXiv preprint arXiv:2307.12698*, 2023.

[10] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, 6, 1993.

[11] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

[12] Lihua Chen, Ghanshyam Pilania, Rohit Batra, Tran Doan Huan, Chiho Kim, Christopher Kuenneth, and Rampi Ramprasad. Polymer informatics: Current status and critical next steps. *Materials Science and Engineering: R: Reports*, 144:100595, April 2021.

[13] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1597–1607. PMLR, November 2020.

[14] Xinlei Chen and Kaiming He. Exploring Simple Siamese Representation Learning, November 2020.

[15] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15750–15758, 2021.

[16] Jörg Degen, Christof Wegscheid-Gerlach, Andrea Zaliani, and Matthias Rarey. On the Art of Compiling and Using 'Drug-Like' Chemical Fragment Spaces. *ChemMedChem*, 3(10):1503–1507, 2008.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Haoran Duan, Cheng Xie, Peng Tang, and Beibei Yu. Contextual features online prediction for self-supervised graph representation. *Expert Systems with Applications*, 238:122075, 2024.

[19] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021.

[20] Tim S Evans and Renaud Lambiotte. Line graphs, link partitions, and overlapping communities. *Physical review E*, 80(1):016105, 2009.

[21] Zhengcong Fei, Mingyuan Fan, and Junshi Huang. A-jepa: Joint-embedding predictive architecture can listen. *arXiv preprint arXiv:2311.15830*, 2023.

[22] Qinghe Gao, Tammo Dukker, Artur M. Schweidtmann, and Jana M. Weber. Self-supervised graph neural networks for polymer property prediction. 2024.

[23] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, koray kavukcuoglu, Remi Munos, and Michal Valko. Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 21271–21284. Curran Associates, Inc., 2020.

[24] Pierre Guetschel, Thomas Moreau, and Michael Tangermann. S-jepa: towards seamless cross-dataset transfer through dynamic spatial attention. *arXiv preprint arXiv:2403.11772*, 2024.

[25] Rishi Gurnani, Christopher Kuenneth, Aubrey Toland, and Rampi Ramprasad. Polymer Informatics at Scale with Multitask Graph Neural Networks. *Chemistry of Materials*, 35(4):1560–1567, February 2023.

[26] William L Hamilton. *Graph representation learning.* Morgan & Claypool Publishers, 2020.

[27] Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *International conference on machine learning*, pages 4116–4126. PMLR, 2020.

[28] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

[29] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*, 2020.

[30] Xiaoxin He, Bryan Hooi, Thomas Laurent, Adam Perold, Yann Lecun, and Xavier Bresson. A Generalization of ViT/MLP-Mixer to Graphs. In *Proceedings of the 40th International Conference on Machine Learning*, pages 12724–12745. PMLR, July 2023.

[31] Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. Graphmae: Self-supervised masked graph autoencoders. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 594–604, 2022.

[32] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

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

[34] Wei Jin, Tyler Derr, Haochen Liu, Yiqi Wang, Suhang Wang, Zitao Liu, and Jiliang Tang. Self-supervised Learning on Graphs: Deep Insights and New Direction, June 2020.

[35] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[36] Christopher Kuenneth, Arunkumar Chitteth Rajan, Huan Tran, Lihua Chen, Chiho Kim, and Rampi Ramprasad. Polymer informatics with multi-task learning. *Patterns*, 2(4):100238, April 2021.

[37] Christopher Kuenneth and Rampi Ramprasad. polyBERT: A chemical language model to enable fully machine-driven ultrafast polymer informatics. *Nature Communications*, 14(1):4099, July 2023.

[38] Greg Landrum et al. Rdkit: Open-source cheminformatics, 2006.

[39] Yann LeCun. A Path Towards Autonomous Machine Intelligence Version 0.9.2, 2022-06-27.

[40] Yann LeCun and Ishan Misra. Self-supervised learning: The dark matter of intelligence. `https://ai.meta.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/`, 2021. Accessed: December 13, 2023.

[41] Namkyeong Lee, Junseok Lee, and Chanyoung Park. Augmentation-free self-supervised learning on graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 7372–7380, 2022.

[42] Xiao Qing Lewell, Duncan B Judd, Stephen P Watson, and Michael M Hann. Recap retrosynthetic combinatorial analysis procedure: a powerful new technique for identifying privileged molecular fragments with useful applications in combinatorial chemistry. *Journal of chemical information and computer sciences*, 38(3):511–522, 1998.

[43] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Zhaoyu Wang, Li Mian, Jing Zhang, and Jie Tang. Self-supervised Learning: Generative or Contrastive. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021.

[44] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[45] Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip S. Yu. Graph Self-Supervised Learning: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(6):5879–5900, June 2023.

[46] Ruimin Ma and Tengfei Luo. Pi1m: a benchmark database for polymer informatics. *Journal of Chemical Information and Modeling*, 60(10):4684–4690, 2020.

[47] Tyler B. Martin and Debra J. Audus. Emerging Trends in Machine Learning: A Polymer Perspective. *ACS Polymers Au*, 3(3):239–258, June 2023.

[48] Gary C McDonald. Ridge regression. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(1):93–100, 2009.

[49] Owen Queen, Gavin A. McCarver, Saitheeraj Thatigotla, Brendan P. Abolins, Cameron L. Brown, Vasileios Maroulas, and Konstantinos D. Vogiatzis. Polymer graph neural networks for multitask property learning. *npj Computational Materials*, 9(1):1–10, May 2023.

[50] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[51] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

[52] Patrick Reiser, Marlen Neubert, André Eberhard, Luca Torresi, Chen Zhou, Chen Shao, Houssam Metni, Clint van Hoesel, Henrik Schopmans, Timo Sommer, and Pascal Friederich. Graph neural networks for materials science and chemistry. *Communications Materials*, 3(1):1–18, November 2022.

[53] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50(5):742–754, 2010.

[54] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying WEI, Wenbing Huang, and Junzhou Huang. Self-Supervised Graph Transformer on Large-Scale Molecular Data. In *Advances in Neural Information Processing Systems*, volume 33, pages 12559–12571. Curran Associates, Inc., 2020.

[55] Ayumu Saito and Jiju Poovvancheri. Point-jepa: A joint embedding predictive architecture for self-supervised learning on point cloud. *arXiv preprint arXiv:2404.16432*, 2024.

[56] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

[57] Kianoosh Sattari, Yunchao Xie, and Jian Lin. Data-driven algorithms for inverse design of polymers. *Soft Matter*, 17(33):7607–7622, August 2021.

[58] Geri Skenderi, Hang Li, Jiliang Tang, and Marco Cristani. Graph-level Representation Learning with Joint-Embedding Predictive Architectures, September 2023.

[59] Hannes Stärk. Self-Supervised learning for small Molecular Graphs.

[60] Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society: Series B (Methodological)*, 36(2):111–133, 1974.

[61] Mengying Sun, Jing Xing, Huijun Wang, Bin Chen, and Jiayu Zhou. MoCL: Data-driven Molecular Fingerprint via Knowledge-aware Contrastive Learning from Molecular Graph. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 3585–3594, Virtual Event Singapore, August 2021. ACM.

[62] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Rémi Munos, Petar Veličković, and Michal Valko. Bootstrapped representation learning on graphs. In *ICLR 2021 Workshop on Geometrical and Topological Representation Learning*, 2021.

[63] Tobias Uelwer, Jan Robine, Stefan Sylvius Wagner, Marc Höftmann, Eric Upschulte, Sebastian Konietzny, Maike Behrendt, and Stefan Harmeling. A Survey on Self-Supervised Representation Learning, August 2023.

[64] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

[65] Yuyang Wang, Zijie Li, and Amir Barati Farimani. Graph Neural Networks for Molecules. volume 36, pages 21–66. 2023.

[66] Yuyang Wang, Jianren Wang, Zhonglin Cao, and Amir Barati Farimani. Molecular contrastive learning of representations via graph neural networks. *Nature Machine Intelligence*, 4(3):279–287, 2022.

[67] Jun Xia, Lirong Wu, Jintao Chen, Bozhen Hu, and Stan Z. Li. SimGRACE: A Simple Framework for Graph Contrastive Learning without Data Augmentation. In *Proceedings of the ACM Web Conference 2022*, pages 1070–1079, Virtual Event, Lyon France, April 2022. ACM.

[68] Jun Xia, Yanqiao Zhu, Yuanqi Du, and Stan Z Li. Pre-training graph neural networks for molecular representations: retrospect and prospect. In *ICML 2022 2nd AI for Science Workshop*, 2022.

[69] Yaochen Xie, Zhao Xu, and Shuiwang Ji. Self-supervised representation learning via latent graph prediction. In *International Conference on Machine Learning*, pages 24460–24477. PMLR, 2022.

[70] Yaochen Xie, Zhao Xu, Jingtun Zhang, Zhengyang Wang, and Shuiwang Ji. Self-Supervised Learning of Graph Neural Networks: A Unified Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):2412–2429, February 2023.

[71] Changwen Xu, Yuyang Wang, and Amir Barati Farimani. TransPolymer: A Transformer-based language model for polymer property predictions. *npj Computational Materials*, 9(1):1–14, April 2023.

[72] Hironao Yamada, Chang Liu, Stephen Wu, Yukinori Koyama, Shenghong Ju, Junichiro Shiomi, Junko Morikawa, and Ryo Yoshida. Predicting Materials Properties with Little Data Using Shotgun Transfer Learning. *ACS Central Science*, 5(10):1717–1730, October 2019.

[73] Cheng Yan and Guoqiang Li. The Rise of Machine Learning in Polymer Discovery. *Advanced Intelligent Systems*, 5(4):2200243, 2023.

[74] Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, Andrew Palmer, Volker Settels, Tommi Jaakkola, Klavs Jensen, and Regina Barzilay. Analyzing Learned Molecular Representations for Property Prediction. *Journal of Chemical Information and Modeling*, 59(8):3370–3388, August 2019.

[75] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph Contrastive Learning with Augmentations. In *Advances in Neural Information Processing Systems*, volume 33, pages 5812–5823. Curran Associates, Inc., 2020.

[76] Xuan Zang, Xianbing Zhao, and Buzhou Tang. Hierarchical Molecular Graph Self-Supervised Learning for property prediction. *Communications Chemistry*, 6(1):1–10, February 2023.

[77] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stephane Deny. Barlow Twins: Self-Supervised Learning via Redundancy Reduction. In *Proceedings of the 38th International Conference on Machine Learning*, pages 12310–12320. PMLR, July 2021.

[78] Hengrui Zhang, Qitian Wu, Junchi Yan, David Wipf, and Philip S Yu. From Canonical Correlation Analysis to Self-supervised Graph Neural Networks. In *Advances in Neural Information Processing Systems*, volume 34, pages 76–89. Curran Associates, Inc., 2021.

[79] Leili Zhang, Vasumitra Rao, and Wendy Cornell. r-brics–a revised brics module that breaks ring structures and carbon chains. *ChemMedChem*, page e202300202.

[80] Pei Zhang, Logan Kearney, Debsindhu Bhowmik, Zachary Fox, Amit K. Naskar, and John Gounley. Transferring a molecular foundation model for polymer property predictions, October 2023.

[81] Shichang Zhang, Ziniu Hu, Arjun Subramonian, and Yizhou Sun. Motif-Driven Contrastive Learning of Graph Representations, April 2021.

[82] Zaixi Zhang, Qi Liu, Hao Wang, Chengqiang Lu, and Chee-Kong Lee. Motif-based Graph Self-Supervised Learning for Molecular Property Prediction. In *Advances in Neural Information Processing Systems*, volume 34, pages 15870–15882. Curran Associates, Inc., 2021.

[83] Tong Zhao, Wei Jin, Yozen Liu, Yingheng Wang, Gang Liu, Stephan Günnemann, Neil Shah, and Meng Jiang. Graph Data Augmentation for Graph Machine Learning: A Survey, January 2023.

[84] Yuankai Zhao, Roger J. Mulder, Shadi Houshyar, and Tu C. Le. A review on the application of molecular descriptors and machine learning in polymer design. *Polymer Chemistry*, 14(29):3325–3346, 2023.

[85] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph Contrastive Learning with Adaptive Augmentation. In *Proceedings of the Web Conference 2021*, pages 2069–2080, Ljubljana Slovenia, April 2021. ACM.

Appendices

A Node-centred and Edge-centred message passing with WDMPNN

The choice of edge-centred convolutions differs from most common GNNs, which use node-centred message passing. The motivation behind this design, explained in [74] is to prevent totters, that is, to avoid messages being passed along any path of the form $v_1 v_2 ... v_n$ where $v_i = v_{i+2}$ for some i, which are thought to introduce noise into the graph representation by creating unnecessary loops in the message passing trajectory. However, we argue that such loops are not really relevant for the dataset considered as the polymer graphs utilized are unlikely to contain such small cyclic patterns. Working with edge-centred convolutions requires big and sparse adjacency matrices of shape $A_{e \times e}$ and $A_{e \times n}$, which makes the process more cumbersome, complex, and unintuitive. This is also due to the fact that the edge-centred operation is not well supported in *Pytorch Geometric*, requiring extra work to precompute the adjacency matrices. This affects the capability to test on different datasets (i.e. with cross-validation) as the precomputing stage is computationally expensive. The performance of the two methods on EA and IP on the Aldeghi and Coley dataset can be seen in Tables A1 and A2 respectively. The results were obtained in the scenario where we train on 80% of the data, and test on 20%. The results refer to the test set. As visible in the tables, both methods achieve the same results, making the node-centred convolution a good solution, given the ease of implementation.

Table A1: Comparing the node-centred and the edge-centred message passing on the EA property.

WDMPNN	**R2** $\uparrow$	**RMSE** $\downarrow$
Edge-centred	0.998 ± 0.0003	0.029 ± 0.002
Node-centred	0.998 ± 0.0002	0.027 ± 0.001

Table A2: Comparing the node-centred and the edge-centred message passing on the IP property.

WDMPNN	**R2** $\uparrow$	**RMSE** $\downarrow$
Edge-centred	0.998 ± 0.0007	0.022 ± 0.004
Node-centred	0.997 ± 0.0004	0.025 ± 0.003

B Models hyperparameters

In this section, we list all the hyperparameters utilized for Model I, Model II, the finetune predictor, and vicReg. We do this to allow other researchers to be able to reproduce our results.

B.1 JEPA Models

Table B1: Hyperparameters for Model I.

Hyperparamater	**Value**
Batch size	128
Epochs	10
Learning rate	0.0005
WDMPNN Dropout	0.1
WDMPNN hidden size	128
WDMPNN layers	2
MLP (predictor) layers	3
MLP (predictor) hidden size	300
MLP (predictor) normalization	Batch Norm
Transformer Dropout	0.35
Transformer hidden size	128
Transformer blocks	2
activation (WDMPNN & MLP & Transf)	Relu
Attention type	Hadamard
Optimization algorithm	Adam
Pooling	Mean
Node PE size	20
Patch PE size	20

Table B2: Hyperparameters for Model II.

Hyperparamater	**Value**
Batch size	128
Epochs	10
Learning rate	0.0005
WDMPNN Dropout	0.1
WDMPNN hidden size	300
WDMPNN layers	4
activation (WDMPNN & MLP)	Relu
MLP (predictor) layers	3
MLP (predictor) hidden size	300
MLP (predictor) normalization	Batch Norm
Optimization algorithm	Adam
Pooling	Mean
Node PE size	20
Patch PE size	20

B.2 Finetune predictor

Here we list the parameters of the MLP trained on top of the target encoder for finetuning the model during the downstream task. The pseudolabel predictor (Section 3.5.7), has the same hidden size and number of layers as the finetune predictor.

Table B3: Hyperparameters for the MLP predictor.

Hyperparamater	**Value**
Batch size	64
Epochs	100
Learning rate	0.001
Hidden size	50
Layers	3
Optimization algorithm	Adam

B.3 VicReg

Here we list the parameters of the MLP trained on top of the target encoder for finetuning the model during the downstream task.

Table B4: Hyperparameters for the MLP predictor.

Hyperparamater	**Value**
Share encoders weights	True
Invariance loss weight	25
Variance loss weight	25
Covariance loss weight	1
MLP (expander) layers	2
MLP (expander) hidden size	256
MLP (expander) activation function	Relu
MLP (expander) normalization	Batch Norm

C Linear finetune

In addition to the finetune procedure listed in Section 3.6, we tested the effectiveness of our pretraining strategy in a linear finetune setting. In this case, the finetune dataset size utilized is larger, namely the full 40% for the Aldeghi and Coley finetune dataset. This finetune setting is relatively common when testing SSL approaches, for instance it was done in [58] for Graph-JEPA. Usually, the pretrain and finetune data is exactly the same, but in our case it is not. The goal is to assess the representation learning capabilities of the SSL method. In fact, when training a complex, non-linear model on top of the pretrained model, as we did with a MLP, it is relatively more difficult to assess the effectiveness of the pretrained model, as the complex finetune predictor can potentially already learn effectively.
We utilize as linear finetune predictor, the Ridge regressor [48], as done in [58].
In Table C1 we report the results observed for the Aldeghi and Coley dataset, predicting the EA property. The pretrain JEPA model is the best performing model obtained (Model II). As evidenced by the results, pretraining effectively improves performance also in this linear finetune scenario, with considerably more data available for finetuning.

Table C1: Comparing the non pretrained and the pretrained model in a linear finetune setting.

	R2 (Train) $\uparrow$	**MAE (Test)** $\downarrow$
Non pretrained	0.594 ± 0.03	0.287 ± 0.013
Pretrained	0.69 ± 0.02	0.244 ± 0.008