Deep Surrogate Docking: Accelerating Automated Drug Discovery with Graph Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

The process of screening molecules for desirable properties is a key step in several applications, ranging from drug discovery to material design. During the process of drug discovery specifically, protein-ligand docking, or *chemical docking*, is a standard *in-silico* scoring technique that estimates the binding affinity of molecules with a specific protein target. Recently, however, as the number of virtual molecules available to test has rapidly grown, these classical docking algorithms have created a significant computational bottleneck. We address this problem by introducing *Deep Surrogate Docking (DSD)*, a framework that applies deep learning-based *surrogate modeling* to accelerate the docking process substantially. DSD can be interpreted as a formalism of several earlier surrogate prefiltering techniques. Specifically, we show that graph neural networks (GNNs) can serve as fast and accurate estimators of classical docking algorithms. Additionally, we introduce *FiLMv2*, a novel GNN architecture which we show outperforms existing state-of-the-art GNN architectures, attaining more accurate and stable performance by allowing the model to filter out irrelevant information from data more efficiently. Through extensive experimentation and analysis, we show that the DSD workflow combined with the FiLMv2 architecture provides a 9.496x speedup in molecule screening with a $< 3\%$ recall error rate on an example docking task. Our open-source code is available at [*hidden for anonymous review*].

1 Introduction

The discovery of new chemical compounds is critical to many scientific fields, such as the development of new medicines, environmental energy sources, and sustainable agriculture. Previously, discovery of such molecules involved costly *in-vitro* synthesis and testing of several candidate compounds [42]. However, in recent years, advancement in computational hardware has led to *in-silico* screening of new chemical compounds. Concretely, representations of chemicals stored in virtual libraries are screened for desirability using computational scoring functions, and top candidates are selected for *in-vitro* synthesis, testing, and analysis [19; 28; 42].One of the primary applications of this process is new drug discovery. A particularly popular method for *in-silico* screening of potential compounds for new drug desirability is *protein-ligand docking*. At a high level, docking software analyzes the conformation and orientation (known together as *pose*) of a candidate molecule into the binding site of a particular *target* [19]. By simulating the *pose* of the candidate molecule several times, the algorithm assigns the candidate a *docking score*, which represents the estimated binding affinity of the molecule to the target [42]. Top docking candidates [1] (those with the *most negative* docking scores)

[1] Oftentimes, the top candidates are first selected and then narrowed down further using domain specific methods such as scaffold analysis. These details are outside of the scope of this work, which is focused on accelerating the *docking* phase of filtering.

Submitted to NeurIPS 2021 AI for Science Workshop.

are then selected for wet-lab synthesis and testing [42]. Recently, [28] demonstrated the viability of this docking-enabled drug discovery at scale. Specifically, they demonstrate docking a virtual library of over 170 million molecules against two target receptors, and, by selecting only the top 44 and 549 chemicals, respectively, yielded effective new potential drugs that target each of these receptors. However, this docking process is computationally expensive, and generally CPU bound [46; 42]. Indeed, the approach used by [28] to dock 138 million molecule against a single target required 43,563 CPU core hours to compute.

In order to solve this problem, recent research has focused on using deep learning to accelerate the docking process [19]. Several methods attempt to circumvent the classical docking algorithm altogether, either by using generative models to produce *de-novo* candidates with desirable properties [3; 49; 17; 35; 51], or by using surrogate models to predict docking scores [19; 9]. Recently, graph neural networks (GNNs) have attracted increasing attention from this field, as molecules can be concisely represented as graphs [27; 51; 19]. GNNs are generalizations of common deep learning architectures, such as CNNs and attention networks, and are able to directly operate on graph structured data such as molecules [50]. While current methods show significant promise, they suffer from issues such as candidate drug synthesizability issues [3] and model inaccuracy [19].

In this work, we attempt to address the issues with current deep-learning enabled drug discovery workflows. We do so by (i) proposing a novel *Deep Surrogate Docking* (DSD) workflow, which simplifies and improves the hit-rate of current methods by combining ML-based surrogate docking with classical docking, (ii) proposing a novel GNN architecture, *FiLMv2*, which we show outperforms existing state of the art GNN architectures in an example DSD workflow, and (iii) providing additional auxiliary improvements to the surrogate training process that increase the overall hit rate of the model. Theoretical discussion and extensive experimentation demonstrates the viability of both DSD and the FiLMv2 architecture in accurately accelerating the throughput of the chemical docking task. Specifically, we demonstrate a 9.496x computational speedup with $< 3\%$ error rate compared to classical docking on the ZINC virtual library when targeting the D_4 dopamine receptor.

Section 2.1 describes the DSD workflow. Concretely, a small randomly sampled subset of the virtual molecule library is run through a classical docking algorithm to generate labels (i.e. ground truth docking scores), which are in turn used to train the surrogate model to predict docking scores given a molecule. The trained surrogate model is then used to perform inference on the full molecule library, and the top scoring molecule candidates are run through the classical docking algorithm. The novelties of the DSD formalism with respect to earlier pre-filtering methods include the introduction of refined evaluation metrics and training strategies. Our analysis framework indicates our workflow is nearly 10x as fast as the traditional-only docking method with less than 0.1% error of detecting top hits.

Section 2.2 describes the FiLMv2 architecture, a simple iterative improvement to the existing FiLM architecture which improves model performance in our application. Specifically, the affine transformation used in the FiLM architecture is added to the node representation *after*, rather than before, the non-linearity. This increases the expressiveness of the model by preventing collapse to a single affine transformation. We experimentally demonstrate the improvement of the FiLMv2 architecture compared to other GNN architectures in the DSD surrogate model task.

2 Methods

Here, we present the key methods of our paper, including the details of our surrogate model workflow, the FiLMv2 architecture, the Exponentially Weighted Mean Squared Error Loss and Virtual Node.

2.1 Deep Surrogate Docking

At a high level, surrogate models aim to emulate, and in many cases, replace [13] [31], classical docking simulation software described in Sections 1 and 4. These classical docking methods are generally computationally intensive and limited to CPU [28; 42]. Thus, surrogate models can enable an exciting opportunity to extend docking over to the ever-increasing size [38; 42] of virtual libraries. However, previous analysis [42; 13] has shown that predicting exact docking scores as a regression task can be infeasible. In order to address this issue, we propose the *Deep Surrogate Docking* (DSD) framework. In this workflow, *surrogate models* are trained to select top candidate molecules which

are then docked using a classical algorithm in order to determine the exact docking score and pose information as standard docking workflow. Existing prefiltering techniques such as those proposed in [39] and [54] can be interpreted as special cases of our DSD framework. Specifically, we define a molecule as a graph $m = (V, E)$ where V is a set of vertices and E a set of edges. We define a given virtual library of molecules as $M = \{m_1, m_2, \ldots, m_N\}$ with $|M| = N$. We then define a classical docking algorithm as a function $f_D : (V, E) \to \mathbb{R}$ which maps a given molecule to a scalar docking score [2]. Similarly, we can define a surrogate docking model as a function $f_S : (V, E) \to \mathbb{R}$, which attempts to estimate the output of f_D. We implement this surrogate model as a novel GNN architecture described in Section 2.2.

In our workflow, we first randomly sample a (typically small) subset of M: $D \subseteq M$, with $|D| = \rho N$ determined by hyperparameter ρ (See Section 3.2 for more details). We then run this subset D through the docking algorithm in order to generate labels (docking scores) for each $m \in D$: $Y = \{y_i = f_D(m_i)\}_{i=1}^{|D|}$, where $\forall i\ m_i \in D$. We can then define the training set used for training f_S as $T = \{(m^{(i)}, y^{(i)})\}_{i=1}^{|D|}$, where $\forall i\ y_i \in Y$. This train set can optionally further be split into typical train-val-test sets for the purposes of model evaluation (See 3.1 for details on our data splits). We emphasize here that in chemical docking, molecules with the *lowest* docking scores correspond to the *most desirable*, and we thus use *lowest* and *best* scores interchangeably. This thus coverts the problem of training f_S into a fully supervised machine learning task. f_S is then trained on T using typical gradient descent methods (See Section 3 and Appendix for more details on our training methods).

Next, given a fully trained surrogate model f_S, we can perform inference on the full virtual library M in order to generate estimated docking scores $y_S \in Y_S = \{f_S(m_i)\}_{i=1}^{N}$, where $\forall i\ m_i \in M$. We then define the set $P \subset M$ selecting the top $|P| = \sigma N$ molecules with the lowest estimated docking scores. The *Data Reduction Factor* $\sigma \ll 1$ is the hyperparamter that determines the fraction of most promising (top scoring) molecules that are docked by f_S. These $|P|$ molecules are then input to f_D in order to determine the exact docking score and pose information as standard docking workflow: $Y_P = \{f_D(m_i)\}_{i=1}^{|P|}$, where $\forall i\ m_i \in P$.

We now define, for the purposes of evaluation, the *Hit Threshold*, ζ, a third hyperparameter which determines the fraction of molecules which should be recalled from the model. For example, when $\zeta = 0.001$, we are interested in correctly selecting the top 0.1% of molecules in a given virtual library. Our analysis in Section 3 uses $\zeta = 0.001$ and $\zeta = 0.01$, which is consistent with typical practices in standard docking [42; 28].

One benefit of such a workflow is that it easily allows introduction of inductive bias into f_S so as to properly rank *only top performing* molecule candidates. Typically, only molecules on the order of the top 0.1% or less of the total virtual library are selected for additional domain-specific analysis and possible synthesis [28; 42; 46]. Thus, biases that aid the model in classifying these top molecules, perhaps at the expense of total performance across all docking scores, is useful to the task. One example of such an inductive bias is the exponentially weighted MSE loss described in Section 2.3. Another benefit of this model is the ability to customize the runtime performance vs. accuracy tradeoff. Specifically, as hyperparameter σ is varied, the total error rate with respect to traditional docking and time required for the full DSD is intuitively traded off. Section 3.2 shows that the intuitive notion that σ and model accuracy are positively correlated, holds empirically.

2.2 FiLMv2 Architecture

Our proposed model to identify molecules with the lowest docking scores is based on a new GNN architecture called FiLMv2. FiLMv2 is a variant of the GNN-FiLM architecture [4], which has been shown to outperform other state-of-the-art architectures, like Graph Convolutional Networks (GCN) [24] and Graph Attention Networks (GAT) [43], on regression tasks on molecular graphs. The idea behind the original GNN-FiLM model is to perform feature-wise linear modulation of a message's source representation, $\boldsymbol{h}_u^t$, using a function of the target representation, $\boldsymbol{h}_v^t$:

$$\boldsymbol{h}_v^{t+1} = l\left(\sum_{u \in \mathcal{N}_v} \sigma\left((\boldsymbol{W}_\gamma \boldsymbol{h}_v^t) \odot (\boldsymbol{W}_\alpha \boldsymbol{h}_u^t) + (\boldsymbol{W}_\beta \boldsymbol{h}_v^t)\right)\right) \tag{1}$$

[2] For simplicity, pose estimations are ignored in this discussion, though in practice a pose estimate can also be output from the model. See Appendix for more details.

where $\boldsymbol{W}_\alpha, \boldsymbol{W}_\beta, \boldsymbol{W}_\gamma$ are learned matrices, $\odot$ is the Hadamard element-wise product, $\mathcal{N}_v$ denotes the neighbors of the target node v, and σ and l are ReLU nonlinearities. The GNN-FiLM architecture of Eq. 1 allows to build dynamic filters based on the target node representation to decide which features of the source representation should be discarded or selected in the updated state of the target node. The possibility to apply element-wise modulation of a source's representation based on the target's representation is absent in GCN and GAT architectures, where the target representation is either used as an additional incoming message (GCN) or used to modulate the entire source representation (GAT). Our FiLMv2 architecture introduces a simple modification to the original GNN-FiLM Eq. 1, consisting in applying a ReLU nonlinearity to the target representations on both the multiplicative and additive terms:

$$\boldsymbol{h}_v^{t+1} = l\left(\sum_{u \in \mathcal{N}_v} \left(\text{ReLU}(\boldsymbol{W}_\gamma \boldsymbol{h}_v^t) \odot (\boldsymbol{W}_\alpha \boldsymbol{h}_u^t) + \text{ReLU}(\boldsymbol{W}_\beta \boldsymbol{h}_v^t)\right)\right). \tag{2}$$

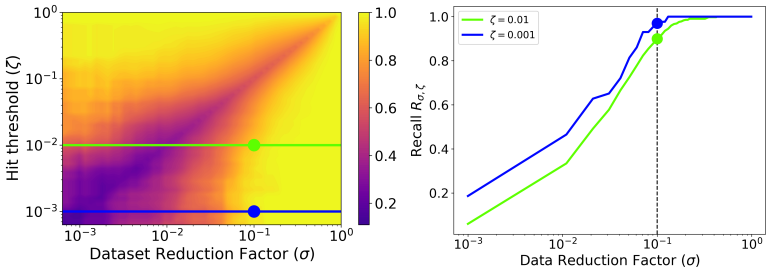
The addition of the ReLU nonlinearities enables to perform a more effective selection of the most relevant information from the source representation. Indeed, target features that are mapped by the linear transformation to any negative number are zeroed out by the ReLU activation function and hence can effectively filter out the corresponding entries of the source representation via the element-wise products. This filtering mechanism is considerably more accurate and stable than the original GNN-FiLM, where a fine-tuned calibration of the linear transformation's parameters is necessary in order to produce an output close to zero. Moreover, without the activation function, for each different target representation a different set of parameter values must be learnt in order to zero-out a particular entry of the source representation, which is highly inefficient and unstable to changes of the target representation. These insights are confirmed by a series of tests performed to assess the effectiveness of various combinations of nonlinearities. In particular, we observe that (see Appendix for details of the experiments): (i) the nonlinearity σ in Eq. 1 can be removed as it does not improve the model's performance; (ii) adding a nonlinearity to the source term $(\boldsymbol{W}_\alpha \boldsymbol{h}_u^t)$ does not further improve the model performance; (iii) using a tanh instead of ReLU nonlinearity does not bring any improvement with respect to the original GNN-FiLMv1. The last result in particular crucially demonstrates that only a nonlinearity that can map to zero a large range of input values can effectively act as a filter, and for this reason the tanh function does not work well as a filter because its asymptotic values are -1 and 1. However, it is important to note that the above analysis of FiLMv2 is largely empirically driven, and further work is needed to assess whether the improvements of FiLMv2 compared to FiLM hold more generally.

2.3 Exponentially Weighted Mean Squared Error Loss (W-MSE)

As described in Section 2.1, the goal of the surrogate model in the DSD framework is to select top molecule candidates which are then run through a classical docking algorithm. Thus, correct classification of top scoring data are far more important than correct classification of comparatively worse scoring data. Additionally, perhaps due to the presence of inductive biases in their algorithm design, current docking methods tend to produce much noisier scores for comparatively lower scoring molecules than higher scoring ones [28]. In order to address these two points, we introduce a weighted mean squared error (W-MSE) loss for use during training. This incentivizes our model to properly score the training samples with good docking scores. Additionally, the model is penalized less for bad predictions of bad scoring chemicals, as these labels may be comparatively noisier. Here, the loss of a single example is weighted by a negative exponential function of the label (docking score). Thus, labels associated with better docking scores (smaller scores) give a higher contribution to the overall loss. Explicitly, for a scalar valued model predicted score z_i for molecule m_i, associated label y_i, mean squared error (MSE) loss function $l : \mathbb{R} \to \mathbb{R}$, and *exponential coefficient* α, we have: $loss = \sum_i e^{-\alpha y_i} \cdot l(z_i, y_i)$ Given that the most desirable (i.e. "best scoring") ligands correspond to those with the most negative scores, α hyperparameter in the range $[0, \infty)$, with $\alpha = 0$ corresponding to the case where no weighing occurs. In practice, we find that $\alpha = 0.8$ works well and thus use it for all models unless otherwise noted, though the effect of this hyperparameter is explored in more detail in the Appendix.

2.4 Virtual Node

First introduced by [14], a so called *virtual node* is a single node added to a graph with a connection to every other node in the given graph. The edge connections of the virtual node are given a special edge



(a) (b)

Figure 1: (a) shows a RES plot of the FiLMv2 model, and superimposes the locations of $\text{AURTC}_{0.01}$ (green line), $\text{AURTC}_{0.001}$ (blue line), $R_{0.1,0.01}$ (green dot), and $R_{0.1,0.001}$ (blue dot). (b) shows the $\text{AURTC}_{0.01}$ and $\text{AURTC}_{0.001}$, and the locations of $R_{0.1,0.01}$, $R_{0.1,0.001}$ with respect to these curves.

type to differentiate them from other nodes in the graph. It has been shown that the use of a virtual node improves performance by facilitating long distance connections between faraway nodes during the propogation phase of a GNN architecture[14]. We introduce this feature to this application, adding a virtual node to each graph as a preprocessing step. Empirically we find that performance improves across all architectures tested. Further, the use of virtual nodes reduces reliance on computationally complex global pooling operations, thus reducing both train time and costly hyperaparamter tuning. The effects of the virtual node on this experiement are explored in ablation (Section 3.4).

3 Experiments

We now describe and justify our experimental setup, defining the dataset, models, and metrics. We apply DSD to docking a virtual library of 128 million molecules with a single receptor target and we compare results of the FiLMv2 architecture with other state-of-the-art GNN models on a small held out test dataset, demonstrating the scalability and performance of the DSD framework. Finally, we ablate the important design elements of the work.

3.1 Experimental Details

Datasets For experimentation, we use the ZINC dataset [38] of 128 million virtual molecules with docking scores against the single D_4 dopamine receptor target, generated using the method described by [28]. Though to the best of our knowledge, no standard benchmark datasets yet exist for chemical docking, the ZINC dataset is one of the largest in size, diverse in molecule *scaffold* representation, and contains other optimizations not found in earlier virtual libraries [28; 38]. Additional details about the dataset description and preprocessing can be found in the Appendix. For the majority of our experiments, we randomly sample 500,000 molecules with ground truth labels from the full dataset, corresponding to $\rho = 0.00313$ (see Section 2.1), which are then split into 80-10-10 train-validation-test partitions. Models are tuned using the validation set and final results (in Section 3.2) are reported on the held out test partition. In Section 3.2 we explicitly vary the ρ parameter in order to demonstrate its effect on the overall framework. In Section 3.3, labels for the full ZINC dataset are generated and used to evaluate model inference over the entire virtual library.

Models We experiment implementing the surrogate model f_S with four GNN convolution operations to use as a backbone for our architecture: the graph isomorphism operator (GIN) [53], graph attention convolution operation (GATv2) [5], the feature-wise linear modulation operation (FiLM) [4], and our novel FiLMv2 operation, which is discussed in detail in Section 2.2. Our full architecture consists of a backbone containing a hyperparameterized number of these graph operations, followed by a global mean pool layer and a MLP mapping the pooled features to a single regression score. Additionally, a single dropout layer and layer normalization layer are added between each graph convolution operation. Each of the four models were trained using the Adam optimizer in order to minimize the weighted mean squared error (W-MSE) loss presented in Section 2.3 and tuned using standard hyperparameter tuning practices. Full details regarding this process can be found in the Appendix.

Metrics Given that the overall task of fitting a model with respect to the DSD framework is novel to the best of our knowledge, careful consideration was given to metrics used to evaluate f_S. For the purposes of evaluation, we can apply a hard threshold to both the ground truth scores and model predictions in order to convert the regression task into a binary classification problem. That is, following the logic in Section 2.1, we can label the top ζ fraction of molecules scored by the classical docking algorithm f_D as positive and otherwise negative. Similarly, we can threshold the top σ fraction of molecules scored by the surrogate model f_S as positive and otherwise negative. Thus, we can treat these labels as y_{true} and y_{pred} for the purposes of evaluating f_S and use traditional binary classification metrics, like precision and recall. However, being functions of ζ and σ, the traditional metrics vary when different values of the two thresholds are selected and thus cannot offer a fair and comprehensive performance evaluation of the models. Thus, we introduce the *RES Score* metric to measure the overall performance of the models for all combinations of threshold values, the AURTC$_\zeta$ metric to measure the performance for a fixed ζ, and provide the parametrized Recall values $R_{\sigma,\zeta}$ for specific values of ζ and σ. The next three paragraphs describe the RES Score, AURTC$_\zeta$, and R$_{\sigma,\zeta}$, respectively. Figure 1 visualizes the relationship between the three metrics. We also provide other typical classification metrics (e.g. AUROC, F-1, etc) for specific values of ζ and σ in the Appendix.

Regression Enrichment Surfaces and the *RES Score* Regression Enrichment Surfaces (RES) have been introduced as a tool to mitigate some of the issues discussed above [8]. RES can be conceptualized as a 3D surface parameterized by σ and ζ. For each $\sigma, \zeta \in (0, 1)$, the value of RES is the recall of y_{true} and y_{pred}, that is the fraction of all top-scoring molecules that were identified by the model. RES can be visualized using a heatmap, such as in Figure 2, where axes are log scaled in order to highlight performance where σ and ζ are close to zero. Additionally, a RES score can be calculated by taking the volume under the RES surface with log-scaled axes. This scalar summarizes the quality of the RES in a single number.

Area Under Recall Threshold Curve (AURTC$_\zeta$) We define the *Recall Threshold Curve* (RTC) as a single dimensional slice of RES obtained by fixing a ζ value and expressing the recall as a function of σ. We define the area under this curve as AURTC$_\zeta$, which allows assessment of performance at particular values of ζ. We report AURTC$_\zeta$ for all of our experiments at $\zeta = 0.01$ and $\zeta = 0.001$.

Parameterized Recall (R$_{\sigma,\zeta}$) Finally, we can follow the logic above and once again reduce the dimensionality of the AURTC$_\zeta$ to a single point by fixing σ and obtaining the corresponding recall. We thus define parameterized recall, R$_{\sigma,\zeta}$, which reports a single recall value for a given σ and ζ. For all experiments, we report R$_{0.1,0.01}$ and R$_{0.1,0.001}$, which we believe represent reasonable typical parameters for most docking tasks.

3.2 Main Results

All four models were trained and hyperparameters tuned according to the details in Section 3.1. The performance on a held out test partition for each model is shown in Table 1. We note that our novel FiLMv2 architecture outperforms all other models in all but one of the selected metrics, where perfomance is equivalent to the original FiLM architecture. Additionally, we find that AURTC$_\zeta$ *improves* as ζ decreases (see Table 1 and Figure 1) demonstrating the model's success at accurately recalling molecules with the best scoring labels. Figure 2 shows the associated RES plots for FiLM and FiLMv2, as well as a heatmap showing the their difference. We note from this Figure that FiLMv2 additionally shows performance improvements in regions where both σ and ζ are very small. As discussed previously, this is especially useful to the overall task and is indicative of FiLMv2's ability to adapt to the W-MSE loss function. Details about the final hyperparameters of each model, total number of model parameters, etc. can be found in the Appendix.

While the majority of our experiments focus on a 500,000 molecule train-val-test dataset (i.e. $\rho = 0.00313$), we vary the size of the dataset D in order to observe its effect of performance. Thus, hyperparameter tuning on FiLMv2 architecture was conducted for several datset sizes. Table 2 in the Appendix shows the intuitive result that for each size tested, RES score and model size increase monotonically with the size of D.

Table 1: Performance of FiLMv2 and other models from the literature as surrogate models on the ZINC dataset with labels generated from docking against the D_4 dopamine receptor. All results except FiLMv2 (128M inf.) are from a single held out test partition.

Model	W-MSE	RES Score	$AURTC_{0.01}$	$AURTC_{0.001}$	$R_{0.1,0.01}$	$R_{0.1,0.001}$
GIN	0.402	0.742	0.938	0.951	0.798	0.804
GATv2	0.396	0.763	0.944	0.959	0.866	0.884
FiLM	0.389	0.768	0.946	**0.965**	0.890	0.968
FiLMv2 (ours)	**0.383**	**0.773**	**0.950**	**0.965**	**0.898**	**0.977**
FiLMv2 (128M inf.)	*0.381*	*0.775*	*0.947*	*0.967*	*0.899*	*0.972*
FiLMv2 (no W-MSE)	N/A	0.758	0.932	0.960	0.838	0.954
FiLMv2 (no virt. node)	0.402	0.760	0.944	0.963	0.878	0.976

3.3 Inference on 128 million virtual molecules

Here, we demonstrate the runtime and performance benefits of the DSD framework at scale by performing inference of the same 128 million ZINC dataset docked by [28]. To evaluate performance, the entire dataset (including the training samples) were given predictions by the final FiLMv2 model. The results of this inference is shown in Table 1 as *FiLMv2 (128M inf.)* and demonstrates that the results of evaluation on a small held out test set generalize well to the entire virtual library. We evaluate inference speed (i.e. *screening throughput*) compared to traditional docking. We define t_D to be the time required to dock this dataset using the traditional method in [28] and $t_{DSD} = t_{inf} + \sigma t_D$ to be the total time required for inference using our DSD framework. Here t_{inf} is the time required to perform model inference on the full dataset and σt_D time required to dock the σN top candidates. For the given virtual library docked against the D_4 receptor, t_{inf} was reported as 1,728 hours on 1,500 CPU cores. t_{inf} for FiLMv2 architecture on a single A100 GPU is 9.167 hours. Thus, for $\sigma = 0.1$ the DSD method provides a $\frac{t_D}{t_{inf}+\sigma t_D} = \frac{1{,}728}{9.167+0.1\cdot 1{,}728}$ = 9.496x speedup, while recalling 97.7% of the top 0.1% of molecules. We note that this speed performance discussion is limited to inference time (known as *docking throughput* in the literature [42]), and does not include the one-time cost of generating a typically small labeled subset of the virtual library and training the model.

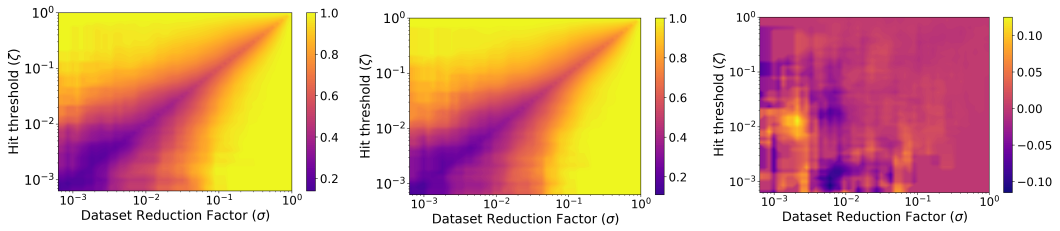


Figure 2: RES plots of FiLM (a) and FiLMv2 (b) on a heldout test set. (c) shows the difference in recall values between FiLM and FiLMv2.

3.4 Ablation Studies

We ablate two major components of our model, the W-MSE loss presented in Section 2.3 and the virtual node presented in 2.4. We thus train a FiLMv2 model with the same hyperparameters as the analogous FiLMv2 model in Section 3.2, but trained with vanilla MSE loss. We also train another FiLMv2 model without use of a virtual node in both training and inference. Table 1 shows that both features result in improvement of the model across all reported metrics. W-MSE is especially effective in metrics that deal with the ability of the model to recall top scoring molecules, such as $R_{0.1,0.01}$ and $R_{0.1,0.001}$, improving performance by 0.05 and 0.023, respectively. This is consistent with our earlier discussions of W-MSE. The virtual node improves performance across metrics, for example increasing the RES Score and $AUROC_{0.01}$ by 0.013 and 0.006, respectively. However, it is important to note that the use of the virtual node can substantially increase training time, as a

virtual node with edge connections to every other node increases the density of each graph. Thus, we additionally experiment with increasing model size when training on data without virtual node augmentation and find that, at a model size such that overall training time is similar to our final FiLMv2 model (with virtual node), the effect of the virtual node is less pronounced when compared to a FiLMv2 model of same size but trained with the virtual node augmentation. However, the overall performance of such models are worse than our final model. This substantiates our claim that use of virtual node both improves overall performance and allows use of smaller model sizes.

4 Related Work

There are generally two high level approaches to automated drug discovery, *de-novo* design of molecules with desirable properties [17; 3; 35; 41; 49] and screening based approaches [42]. While *de-novo* drug design, wherein generative models directly produce novel molecules with useful properties, shows significant promise, current approaches suffer from issues such as lack of molecule synthesizability, algorithmic instability, reduced molecule sizes, etc. [30; 3]. Indeed, even *de-novo* generated drugs may require or benefit from virtual screening, albeit at a comparatively smaller scale. Conversely, screening based approaches attempt to use a *docking algorithm* in order to score molecules from a generally large virtual library of compounds for desirability by evaluating its binding affinity with a particular *target* (also called *receptor*, or *ligand*) [42]. Recently, substantial progress has been made to generate larger and more robust virtual libraries. Notably, [28] demonstrate successfully docking on a virtual library of 170 million molecules, and ultimately discovering a new class of inhibitors for the AMPC receptor. Crucially, they also show that hit rates of molecules decrease monotonically with docking score. This serves as one of the justifications for biasing our loss function to emphasize prediction of the best scoring molecules (See Section 2.3). Recently, substantial work has focused on development of machine learning based surrogate models to accelerate docking. Some approaches focus on traditional feature based methods [9; 20], while others apply language models directly on SMILES strings [47; 18] or GNNs on graph representations of molecules [27; 19]. Of these GNN based approaches, most use trained models to infer docking scores directly (e.g. [27]). Some existing screening methods [39; 54; 9] do use a surrogate prefiltering approach, though they lack several of the key training and evaluation strategies proposed in our DSD workflow. Graph neural networks (GNNs) are generalizations of neural network architectures such as convolution that operate on non-euclidean data (i.e. graphs) [50]. At a high level, GNNs operate through *message passing*, wherein the state of each node is iteratively updated using a function of the state of its neighbor nodes. Recently, substantial progress has been made in designing GNN general architectures that are analogous to common DL architectures such as convolution [4; 24; 7; 16; 29; 36; 10], VAEs [23; 33], attention [43; 5; 1; 40; 6; 22; 32; 45], and others [44; 15; 21; 12]. Additionally, substantial work has been made in creating GNN architectures suitable for specific applications such as for modeling quantum interactions [37] and molecular property prediction [25; 52], though we generally avoid these models in our experimentation as they may inadvertently introduce implicit biases to our framework.

5 Conclusions

In this work, we present *Deep Surrogate Docking* (DSD), a workflow which accelerates traditional chemical docking through the use of surrogate models. We also demonstrate the viability of DSD in the drug discovery domain by using it to dock 128 million molecules in the ZINC dataset against the D_4 dopamine receptor. We show that DSD substantially accelerates docking throughput with minimial error. For example, in one such setup, we are able to speedup a standard docking algorithm by 9.496x, while recalling $> 97\%$ of top 0.1% molecules. Moreover, we show that the DSD workflow allows scientists flexibility to tradeoff inference time and hit accuracy by varying a single data reduction parameter σ. We also present a novel GNN architecture, FiLMv2, which we show outperforms other models from the literature in the example docking task above. FiLMv2 is implemented as a simple modification to the existing Feature-wise linear modulation (FiLM) architecture and attains considerably more accurate and stable performance by allowing the model to more easily filter out irrelevant information from data. This work demonstrates the viability of the DSD workflow with a single large scale example. We leave the analysis of generalizability to other virtual molecule libraries and receptors to future work.

References

[1] S. Bai, F. Zhang, and P. H. Torr. Hypergraph convolution and hypergraph attention. *Pattern Recognition*, 110:107637, 2021.

[2] L. Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[3] J. Bradshaw, B. Paige, M. J. Kusner, M. Segler, and J. M. Hernández-Lobato. Barking up the right tree: an approach to search over molecule synthesis dags. *Advances in Neural Information Processing Systems*, 33:6852–6866, 2020.

[4] M. Brockschmidt. Gnn-film: Graph neural networks with feature-wise linear modulation. In *International Conference on Machine Learning*, pages 1144–1152. PMLR, 2020.

[5] S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? *International Conference on Learning Representations*, 2021.

[6] D. Busbridge, D. Sherburn, P. Cavallo, and N. Y. Hammerla. Relational graph attention networks. *arXiv preprint arXiv:1904.05811*, 2019.

[7] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pages 1725–1735. PMLR, 2020.

[8] A. Clyde, X. Duan, and R. Stevens. Regression enrichment surfaces: a simple analysis technique for virtual drug screening models. *arXiv preprint arXiv:2006.01171*, 2020.

[9] A. Clyde, T. Brettin, A. Partin, H. Yoo, Y. Babuji, B. Blaiszik, A. Merzky, M. Turilli, S. Jha, A. Ramanathan, et al. Protein-ligand docking surrogate models: A sars-cov-2 benchmark for deep learning accelerated virtual screening. *arXiv preprint arXiv:2106.07036*, 2021.

[10] T. Derr, Y. Ma, and J. Tang. Signed graph convolutional networks. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 929–934. IEEE, 2018.

[11] M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[12] H. Gao and S. Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.

[13] F. Gentile, V. Agrawal, M. Hsing, A.-T. Ton, F. Ban, U. Norinder, M. E. Gleave, and A. Cherkasov. Deep docking: a deep learning platform for augmentation of structure based drug discovery. *ACS central science*, 6(6):939–949, 2020.

[14] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

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

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

[17] M. Hartenfeller and G. Schneider. Enabling future drug discovery by de novo design. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(5):742–759, 2011.

[18] S. Honda, S. Shi, and H. R. Ueda. Smiles transformer: Pre-trained molecular fingerprint for low data drug discovery. *arXiv preprint arXiv:1911.04738*, 2019.

[19] S. Jastrzębski, M. Szymczak, A. Pocha, S. Mordalski, J. Tabor, A. J. Bojarski, and S. Podlewska. Emulating docking results using a deep neural network: a new perspective for virtual screening. *Journal of Chemical Information and Modeling*, 60(9):4246–4262, 2020.

[20] J. Jiménez-Luna, F. Grisoni, and G. Schneider. Drug discovery with explainable artificial intelligence. *Nature Machine Intelligence*, 2(10):573–584, 2020.

[21] W. Jin, M. Qu, X. Jin, and X. Ren. Recurrent event network: Autoregressive structure inferenceover temporal knowledge graphs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6669–6683, 2020.

[22] D. Kim and A. H. Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *The Ninth International Conference on Learning Representations (ICLR 2021)*. International Conference on Learning Representations (ICLR 2021), 2021.

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

[24] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*, 2017.

[25] J. Klicpera, J. Groß, and S. Günnemann. Directional message passing for molecular graphs. In *International Conference on Learning Representations*, 2019.

[26] G. Landrum. Rdkit documentation. *Release*, 1(1-79):4, 2013.

[27] J. Lim, S. Ryu, K. Park, Y. J. Choe, J. Ham, and W. Y. Kim. Predicting drug–target interaction using a novel graph neural network with 3d structure-embedded graph representation. *Journal of chemical information and modeling*, 59(9):3981–3988, 2019.

[28] J. Lyu, S. Wang, T. E. Balius, I. Singh, A. Levit, Y. S. Moroz, M. J. O'Meara, T. Che, E. Algaa, K. Tolmachova, et al. Ultra-large library docking for discovering new chemotypes. *Nature*, 566 (7743):224–229, 2019.

[29] Z. Ma, J. Xuan, Y. G. Wang, M. Li, and P. Liò. Path integral based convolution and pooling for graph neural networks. *Advances in Neural Information Processing Systems*, 33:16421–16433, 2020.

[30] D. Martinelli. Generative machine learning for de novo drug discovery: A systematic review. *Computers in Biology and Medicine*, page 105403, 2022.

[31] S. Mehta, S. Laghuvarapu, Y. Pathak, A. Sethi, M. Alvala, and U. D. Priyakumar. Memes: Machine learning framework for enhanced molecular screening. *Chemical science*, 12(35): 11710–11721, 2021.

[32] X. Mo, Y. Xing, and C. Lv. Heterogeneous edge-enhanced graph attention network for multi-agent trajectory prediction. *arXiv preprint arXiv:2106.07161*, 2021.

[33] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang. Adversarially regularized graph autoencoder for graph embedding. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 2609–2615, 2018.

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[35] E. Putin, A. Asadulaev, Y. Ivanenkov, V. Aladinskiy, B. Sanchez-Lengeling, A. Aspuru-Guzik, and A. Zhavoronkov. Reinforced adversarial neural computer for de novo molecular design. *Journal of chemical information and modeling*, 58(6):1194–1204, 2018.

[36] B. Rozemberczki, P. Englert, A. Kapoor, M. Blais, and B. Perozzi. Pathfinder discovery networks for neural message passing. In *Proceedings of the Web Conference 2021*, pages 2547–2558, 2021.

[37] K. Schütt, P.-J. Kindermans, H. E. Sauceda Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *Advances in neural information processing systems*, 30, 2017.

[38] T. Sterling and J. J. Irwin. Zinc 15–ligand discovery for everyone. *Journal of chemical information and modeling*, 55(11):2324–2337, 2015.

[39] E. Tejera, C. R. Munteanu, A. López-Cortés, A. Cabrera-Andrade, and Y. Pérez-Castillo. Drugs repurposing using qsar, docking and molecular dynamics for possible inhibitors of the sars-cov-2 mpro protease. *Molecules*, 25(21):5172, 2020.

[40] K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

[41] X. Tong, X. Liu, X. Tan, X. Li, J. Jiang, Z. Xiong, T. Xu, H. Jiang, N. Qiao, and M. Zheng. Generative models for de novo drug design. *Journal of Medicinal Chemistry*, 64(19):14011–14027, 2021.

[42] P. H. Torres, A. C. Sodero, P. Jofily, and F. P. Silva-Jr. Key topics in molecular docking for drug design. *International journal of molecular sciences*, 20(18):4574, 2019.

[43] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[44] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm. Deep graph infomax. *ICLR (Poster)*, 2(3):4, 2019.

[45] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu. Heterogeneous graph attention network. In *The world wide web conference*, pages 2022–2032, 2019.

[46] Z. Wang, H. Sun, X. Yao, D. Li, L. Xu, Y. Li, S. Tian, and T. Hou. Comprehensive evaluation of ten docking programs on a diverse set of protein–ligand complexes: the prediction accuracy of sampling power and scoring power. *Physical Chemistry Chemical Physics*, 18(18):12964–12975, 2016.

[47] Z. Wang, M. Liu, Y. Luo, Z. Xu, Y. Xie, L. Wang, L. Cai, Q. Qi, Z. Yuan, T. Yang, et al. Advanced graph and sequence neural networks for molecular property prediction and drug discovery. *Bioinformatics*, 38(9):2579–2586, 2022.

[48] D. Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

[49] Y. Wu, N. Choma, A. Chen, M. Cashman, É. T. Prates, M. Shah, V. G. M. Vergara, A. Clyde, T. S. Brettin, W. A. de Jong, et al. Spatial graph attention and curiosity-driven policy for antiviral drug discovery. *arXiv preprint arXiv:2106.02190*, 2021.

[50] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[51] J. Xiong, Z. Xiong, K. Chen, H. Jiang, and M. Zheng. Graph neural networks for automated de novo drug design. *Drug Discovery Today*, 26(6):1382–1393, 2021.

[52] Z. Xiong, D. Wang, X. Liu, F. Zhong, X. Wan, X. Li, Z. Li, X. Luo, K. Chen, H. Jiang, et al. Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism. *Journal of medicinal chemistry*, 63(16):8749–8760, 2019.

[53] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[54] Y. Yang, K. Yao, M. P. Repasky, K. Leswing, R. Abel, B. K. Shoichet, and S. V. Jerome. Efficient exploration of chemical space with docking and deep learning. *Journal of Chemical Theory and Computation*, 17(11):7106–7119, 2021.

Appendix

A Additional Details Regarding Chemical Docking for Drug Discovery

Here, we discuss and clarify details about classical docking algorithms which are relevant to this work.

Overview of Classical Docking Algorithms At a high level, docking algorithms are functions that (1) sample and (2) score molecules against a ligand/receptor. This scoring function estimates the binding affinity of a candidate molecule against the receptor. As discussed in the main text, they are thus often used to screen molecules (i.e. potential drug candidates) against a specific ligand. The binding affinity calculation is acheived by simulating the interaction of a given molecule with the ligand at several orientations (that is, *poses*) [46]. However, most docking algorithms only succeed in ranking the relative binding affinities of a set of molecules and either do not aim to, or are not successful in, predicting the true binding affinity of the molecule-lingand pair[46]. It is also important to note that recently [28] showed that the *hit-rate*, or quality of docking prediction, decreases monotonically with the docking score. Thus, many modern screening workflows use docking as a prefilter, and then use domain specific heuristics to further narrow down the set of molecules for *in-vitro* synthesis [28; 42; 9]. This further motivates the use of surrogate models as drop-in prefilters in the automated drug discovery pipeline.

Pose Estimations As discussed above, *pose estimations* are calculated by docking algorithms, and are thus output by most docking software [46]. However, for simplicity, we define the docking function f_D in the main text as only outputting a docking score. In practice, both the score and pose can be recovered from f_D. However, the pose is not relevant to the DSD workflow.

Lack of Parallelizability Due to the sequencial nature of the docking process, there is currently no good way to parallelize the docking of a single molecule, and thus docking algorithms are generally CPU bound [28; 46]. In our opinion, this further motivates the need for accurate surrogate models in order to accelerate the automated drug discovery process.

B Experimental Details

Here, we detail our preprocessing and training setup, hyperparameter tuning strategy, provide details about the hardware system used, and provide additional metrics for the performance of the top models for each of the four tested architectures listed in Table 1 one of the main text.

Preprocessing Details ZINC data is provided in *Simplified Molecular Input Line Entry System*, or *SMILES*, format, a common string representation of molecules [48]. We use standard techniques [19] to convert each SMILES string into a (two-dimensional) graph representation for use by a GNN. Specifically, we create a graph representation of each molecule where the node represents a single atom in a molecule and contains nine features: atomic number, chirality, degree, formal charge, number of associated hydrogen bonds, number of radical electrons, hybridization, whether it is aromatic, and whether it is contained in a ring. Edges represent the bond between atoms in a molecule and are encoded with three features: the bond type, stereochemical information, and whether it is conjugated. More information about the technical details of these features, as well as their *in-silico* representations can be found in the RDKit documentation [26], which was the tool used to perform this preprocessing. Additionally, as described in Section 2.4 of the main text, a virtual node with a distinct *virtual* edge type to every other node in the graph was added to each molecule during preprocessing. Molecules with NaN docking values where dropped, as the docking algorithm used by [28] represents either noisy or poor performing molecules with NaNs. Finally, all docking labels were standardized by the mean and standard deviation of the training set label distribution.

Model Training and Hyperparameter Tuning As described in Section 3 of the main text, we experiment with implementing four GNN convolution operations to use as a backbone for our architecture: the graph isomorphism operator (GIN)[53], graph attention convolution operation (GATv2) [5], the feature-wise linear modulation operation (FiLM) [4], and our novel FiLMv2

operation, which is discussed in detail in Section 2.2 of the main text. For each graph operation, we stack a hyperparameterized number of layers of said operation in order to create the backbone of the model. A layer normalization layer, dropout layer, and ReLU non-linearity are applied between each of these layers. After this backbone, a global mean pool aggregates the node representations and a single MLP layer maps this aggregated representation to a single regression score. In order to simplify the hyperparameter tuning process, a single hidden dimension size and dropout amount is chosen for each of the layers. Thus, the hyperparameters are the size of this hidden layer, the number of layers in the backbone, and the amount of dropout. For all models, hyperparameter tuning was run, via grid search, on hidden layer sizes of $\{2, 4, 8, 16, 32, 64, 128, 256\}$, number of layers of $\{1, 2, 4, 5, 6\}$, and dropout values of $\{0.0, 0.1, 0.15, 0.2, 0.3\}$. For hidden layer sizes, additional tuning was done in multiples of four at a local neighborhood around the initial hidden layer size chosen. For example, if the best performing hidden layer size in the initial grid search was 64, additional testing was done for $\{52, 56, 60, 64, 68, 72, 76, 80\}$. Each model was trained for 1,000 epochs, and the epoch where validation loss was best was selected as the final model. All models were trained with the Adam optimizer with a learning rate of 0.001 and a batch size of 512. The final hyperparameter values for the best model for each of the four architectures (corresponding to the first four rows of Table 1 in the main text) are shown in Table 2.

Hardware Details Training was conducted using an internal cluster of 8 NVIDIA DGX A100 GPUs (40GB memory), using Pytorch's built-in distributed data parallel (DDP) tool. Inference was conducted using a single NVIDIA DGX A100 GPU (40GB memory). Total computation required to train the best performing model is 29.421 GPU hours. Total computation used to perform inference on the total dataset of 128 million molecules is 9.167 GPU hours.

Additional Metrics for Final Models As discussed in Section 3.1 of the main text, it is possible to convert the DSD task to a classification problem by fixing the parameters σ and ζ. Table 2 also reports the AUROC and F-1 scores for each of the models for $\sigma = 0.1$ and $\zeta = 0.001$. Note that as AUROC by nature varies the classification threshold (i.e. σ), a fixed σ is not used to calculate the AUROC.

Table 2: Hyperparameter values and additional metrics for the best performing models of each architecture. These models correspond to the first four rows of Table 1 in the main text. RES Score is repeated in this table to facilitate easier reference.

Model	Hidden Layer Size	#Layers	Dropout	#Params.	RES Score	AUROC	F-1
GIN	120	4	0.1	132,961	0.742	0.979	0.998
GATv2	64	4	0.1	105,537	0.763	0.986	**0.999**
FiLM	64	4	0.1	**102,977**	0.768	0.985	**0.999**
FiLMv2 (ours)	64	4	0.1	**102,977**	**0.773**	**0.987**	**0.999**

Error Bars As part of our experimentation, we retrained the best performing models for all four architectures 5 times, varying the random seeds used each time. For all models, the range of W-MSE was < 0.0002 and the range of the RES Score was < 0.0004. Thus, we conclude that the results obtained in this work are not significantly affected by stochasticity in the training process.

C Additional Details on the Effects of Dataset Size

Section 3.3 describes the phenomenon that as the dataset size used for training is increased, so does the RES Score and number of parameters of the best tested model. In our experiments, we performed grid search on the FiLMv2 model, the best performing architecture from Section 3.2. Specifically, we used the same assumptions for model architecture as in B and tested hidden dimension sizes of $\{4, 8, 16, 32, 64, 128\}$, number of convolutional layers of $\{1, 2, 4, 5, 6\}$, and dropout of $\{0.1, 0.15, 0.2\}$. After finding the optimal model associated with the combination the hyperparameters above, we performed a smaller search around the chosen hidden dimension in order to further tune this hyperparameter. Each model was trained for 1,000 epochs, and the epoch where validation loss was best was selected as the final model. Table 3 shows the RES score and number of parameters for the final model chosen for testing after hyperparameter tuning. Please note that

the dataset size refers to the total dataset size and, as described in 3.1, this number is further broken down into a 80-10-10, train-val-test split. For example, the first row of Table 3 has a dataset of 5,000, of which $5,000 \cdot 0.8 = 4,000$ examples were used for training. Table 4 shows the chosen hyperparameters for each of these models.

Table 3: RES and model sizes for different sizes of D.

Dataset Size	RES Score	#Params.
5,000	0.594	7,313
50,000	0.724	26,913
100,000	0.756	58,801
500,000	0.773	102,977
2,000,000	0.781	600,193

Table 4: Hyperparameters of each of the models in Table 3

Dataset Size	Hidden Layer Size	#Layers	Dropout
5,000	16	4	0.1
50,000	32	4	0.1
100,000	48	4	0.1
500,000	64	4	0.1
2,000,000	128	6	0.1

D Additional FiLMv2 Experiments

Section 2.2 of the main text describes the architecture of our novel FiLMv2 convolution in detail. There, three main empirical claims are used for discussion:

1. The nonlinearity $sigma$ in Eq. 1 of the main text can be removed as it does not improve the model's performance.
2. Adding a nonlinearity to the source term ($(\boldsymbol{W}_\alpha \boldsymbol{h}_u^t)$) does not further improve the model performance.
3. Using a tanh instead of ReLU nonlinearity does not bring any improvement with respect to the original FiLMv1.

The experiments conducted that led to these conclusions are detailed below for completeness. We define *FiLMv2 (tanh)* as the FiLMv2 model defined in Section 2.2 of the main text, but with all instances of ReLU non-linearity replaced with tanh non-linearity. We also define *FiLMv2 (source act.)* as the FiLMv2 model defined in Section 2.2 of the main text, but with an additional ReLU non-linearity added to source term ($(\boldsymbol{W}_\alpha \boldsymbol{h}_u^t)$). *FiLM* and *FiLMv2* are as defined in the main text.

Table 5 shows the results of testing each of these models on the 500k train-val-test set used for training the main models. Each model was created by using one of the four convolution operations described above (FiLM, FiLMv2, FiLMv2 (tanh), FiLMv2 (source act.)) and building a backbone of four layers with a hidden dimension size of 64. All other details, such as dropout and layer norm, global pooling, and MLP, etc. are the same as in B. Thus, all models have the same number of parameters, and are trained with the same seed on the same dataset.

Table 5: Performance comparison of FiLMv2 with FiLM and FiLMv2 variants.

Model	W-MSE	RES Score	$\text{AURTC}_{0.01}$	$\text{AURTC}_{0.001}$	$\text{R}_{0.1,0.01}$	$\text{R}_{0.1,0.001}$
FiLM	0.389	0.768	0.946	0.965	0.890	0.968
FiLMv2	0.383	0.773	0.950	0.965	0.898	0.977
FiLMv2 (tanh)	0.388	0.765	0.947	0.966	0.892	0.967
FiLMv2 (source act.)	0.385	0.770	0.949	0.965	0.897	0.974

E W-MSE Loss and the *alpha* Hyperparameter

Section 2.3 in the main text describes the weighted mean square error (W-MSE) used in order to increase the recall of top scoring molecules. Here, we show the results for training the top performing FiLMv2 model from Section 3.2 of the paper at several values of α, the hyperparameter responsible for the degree to which each molecule is weighted (See Section 2.3 for more details). Table 6 shows the results of such experiments. Since varying α changes the value of the overall loss for a constant dataset and model, the absolute W-MSE loss when varying α may not necessarily be indicative of downstream performance. Thus, Table 6 also shows performance of each model for all the metrics used for evaluation in Table 1 of the main text. We see that $\alpha = 0.8$ matches or beats performance on all but one of these metrics and thus use $\alpha = 0.8$ for all other experiments.

Table 6: W-MSE and several metrics for the best peforming FiLMv2 model trained with different W-MSE weighing schemes (i.e. different α vales).

α	W-MSE	RES Score	$\text{AURTC}_{0.01}$	$\text{AURTC}_{0.001}$	$\text{R}_{0.1,0.01}$	$\text{R}_{0.1,0.001}$
0	0.401	0.758	0.932	0.960	0.838	0.954
0.2	0.358	0.764	0.942	0.964	0.869	0.953
0.5	**0.348**	**0.773**	0.947	0.965	0.893	**0.977**
0.8	0.383	**0.773**	**0.950**	0.965	**0.900**	**0.977**
1.0	0.433	0.770	0.949	0.966	0.893	**0.977**
1.2	0.499	0.762	**0.950**	**0.970**	0.893	0.953

F Tools

Software Tools Substantial portions of this project were implemented in the Pytorch [34] and Pytorch Geometric [11] libraries, both of which are open source (BSD and MiT licenses, respectively) software tools. Each tool was utilized with adherence to their licensing conditions. We used Weights & Biases [2], which is free for academic use (MiT license) for experiment tracking and hyperparamter sweeps. Additionally, several other Python libraries were used for various auxiliary tasks ranging from preprocessing to utility tasks. A full list of these tools can be found in our code's `requirements.txt` file. Again, each tool was used according to their respective library's licensing conditions.

Data As discussed in the main text, data from the ZINC [38] dataset was used. ZINC is a free and publicly available dataset, and was used according to their licensing policies. Additionally, labels for this data was provided by the docking conducted by [28], and was used in this project with permission from the authors.