# A Supplementary information

## A.1 Dataset curation & statistics

Unlabeled training data was downloaded from the ChEMBL database to represent small molecules with drug-like properties. At the time of data download, the ChEMBL database contained 2,399,743 unique compounds. We constrained the compound category to "small molecules" and enforced adherence to the Lipinsky rule of five (Lipinski et al., 1997), specifically setting violations to "0," resulting in a refined set of 1,348,115 compounds available for download. The molecules were acquired in the form of Simplified Molecular Input Line Entry System (SMILES) (Weininger, 1988) strings. Subsequent to data retrieval, we conducted preprocessing using the database cleaning functionalities of the Chemical Data Processing Toolkit (*CDPKit* 2024). This process involved the removal of solvents and counter ions, adjustment of protonation states to a physiological pH value, and elimination of duplicate structures, where compounds differing only in their stereo configuration were regarded as duplicates. To prevent data leakage, we carefully removed all structures from the training data that would occur in one of the test sets we used for our benchmark experiments. The final set was comprised of 1,221,098 compounds. For each compound within the dataset, a 3D conformation was generated using the CONFORGE (Seidel, Permann, et al., 2023) conformer generator from the CDPKit, which was successful for 1,220,104 compounds. To enhance batch diversity, we generated only one conformation per compound for contrastive training. Subsequently, 3D pharmacophores were computed for each conformation, with removal of pharmacophores containing less than four pharmacophoric points. The ultimate dataset comprised 1,217,361 distinct pharmacophores. The data pipeline is illustrated in Figure 6.

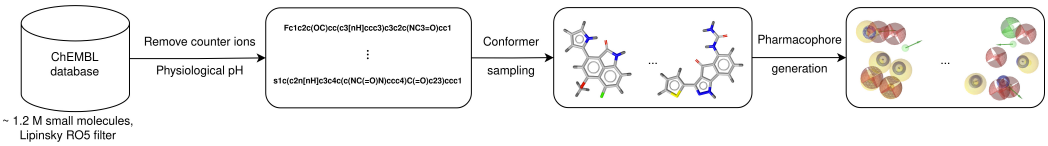


Figure 6: Training data generation from the ChEMBL database (*ChEMBL* 2024). To resemble ligands with drug-like properties, unlabeled training data was derived from approximately 1.2 M small molecules from the ChEMBL database.

Figure 7 shows the frequency of pharmacophores with a specific pharmacophoric point count in the training data. On average, a pharmacophore consists of 13 pharmacophoric points, with the largest pharmacophore in the dataset containing 32 points. Pharmacophores with fewer than four points were omitted during data clean-up. Hydrophobic pharmacophoric points and hydrogen bond acceptors are the most prominent, while hydrogen bond donors and aromatics occur less frequently. Ionizable pharmacophoric points and halogen bond donors are comparatively rare.

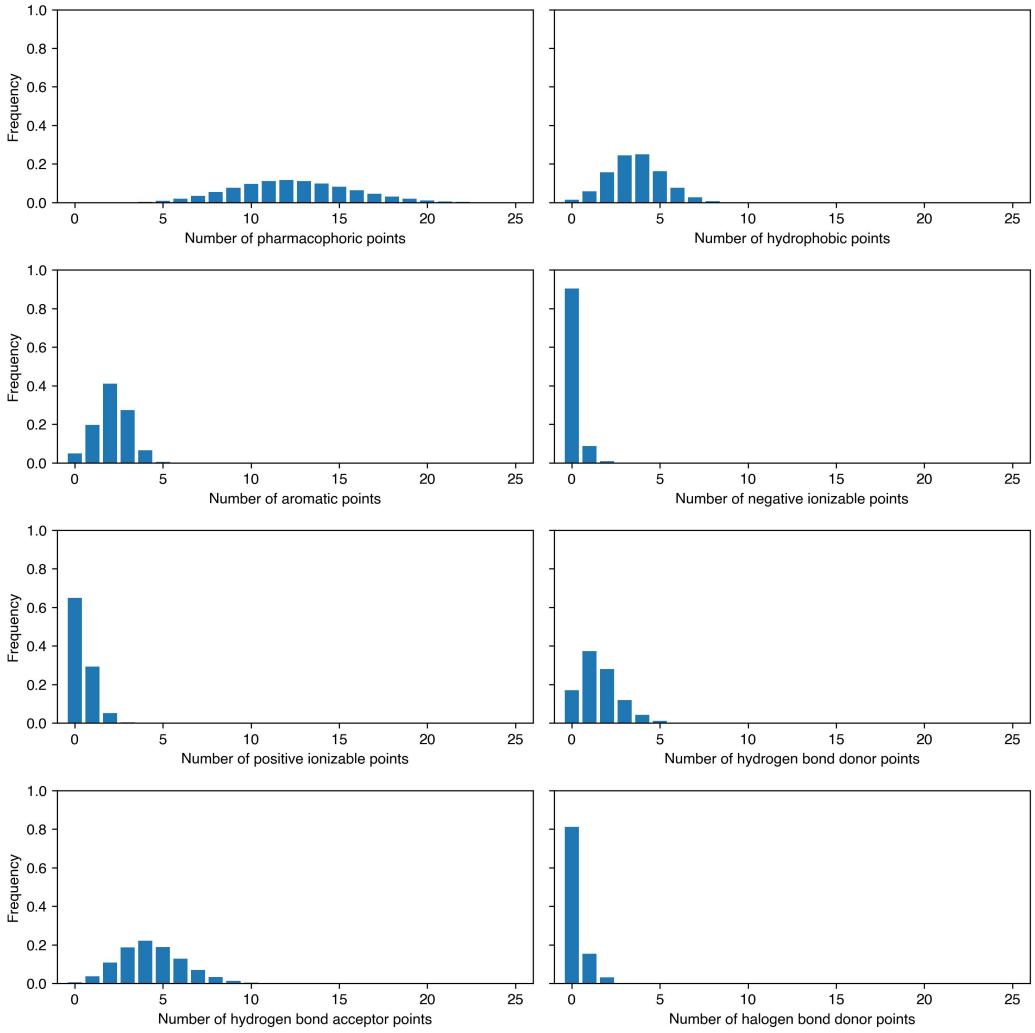


Figure 7: Pharmacophoric point statistics of the training data. The respective histograms display the total number of pharmacophoric points and the number of points of specific types per pharmacophore in the training data. The complete training dataset contains 1,217,361 distinct pharmacophores.

### A.2 Augmentation module

The augmentation module receives the initial pharmacophore $\mathbf{x_0} = [\mathbf{h_0}, \mathbf{r_0}]$, with the initial OHE feature matrix $\mathbf{h_0}$ and the Cartesian coordinates $\mathbf{r_0}$. Edge attributes of the complete graph were calculated from the pair-wise distances between nodes after modifying the input according to the augmentation strategy, which combines random node deletion and random node displacement. The module outputs the modified tuple $\mathbf{x} = [\mathbf{h}, \mathbf{e}]$ with the feature matrix $\mathbf{h}$ and the edge attributes $\mathbf{e}$.

**Node deletion** Random node deletion involved removing at least one node, with the upper bound determined by the cardinality of the set of nodes $V_i$ of graph $G_i$. To ensure the output graph retained at least three nodes, the maximum number of deletable nodes was $|V_i| - 3$. The number of nodes to delete was drawn uniformly at random.

**Node displacement** There are two modes for the displacement of pharmacophoric points. Positive pairs were constructed by displacing the pharmacophoric points within the tolerance sphere of the initial pharmacophore. For simplicity, we assumed the same tolerance sphere radius $r_T$ across different pharmacophoric types. The coordinate displacement $(\Delta x, \Delta y, \Delta z)$ was created from spherical coordinates $\phi \sim \mathcal{U}(0, 2\pi)$ and $\cos\theta \sim \mathcal{U}(-1, 1)$, which were drawn at random from a uniform distribution. The coordinate displacement was calculated as

$$\Delta x = \Delta r \sin\theta \cos\phi,\ \Delta y = \Delta r \sin\theta \sin\phi,\ \Delta z = \Delta r \cos\theta \tag{6}$$

where $\Delta r = r_T \sqrt[3]{u}$ and $u \sim \mathcal{U}(0, 1)$. Negative pairs were created by displacement of the nodes at the border of the tolerance sphere. This was achieved by random sampling from a sphere surface, *i.e.* with $u = 1$.

### A.3 Message passing neural network

Convolution on irregular domains like graphs is formulated as message passing, which can generally be described as:

$$\mathbf{h}_i^{(k)} = \gamma^{(k)}(\mathbf{h}_i^{(k-1)}, \bigoplus_{j \in \mathcal{N}(i)} \phi^{(k)}(\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)}, \mathbf{e}_{ij})) \tag{7}$$

where $\mathbf{h}_i^{(k)} \in \mathbb{R}^{F'}$ denotes the node features of node $i$ at layer $k$, $\mathbf{h}_i^{(k-1)} \in \mathbb{R}^{F}$ denotes the node features of node $i$ at layer $k-1$, $\mathbf{e}_{ij} \in \mathbb{R}^{D}$ the edge features of the edge from node $i$ to node $j$, $\gamma^{(k)}$ and $\phi^{(k)}$ are parameterized, differentiable functions, and $\bigoplus$ is an aggregation operator like, *e. g.*, the summation operator (Fey et al., 2019). In our encoder architecture, we employed the following edge-conditioned convolution operator, which was proposed both by Gilmer et al. (2017) and Simonovsky et al. (2017):

$$\mathbf{h}_i^{(k)} = \Theta \mathbf{h}_i^{(k-1)} + \sum_{j \in \mathcal{N}(i)} \mathbf{h}_j^{(k-1)} \cdot \psi_\Theta(\mathbf{e}_{ij}) \tag{8}$$

where $\Theta \in \mathbb{R}^{F \times F'}$ denotes learnable weights and $\psi_\Theta(\cdot) : \mathbb{R}^D \to \mathbb{R}^{F \times F'}$ denotes a neural network, in our case an MLP with one hidden layer. These affine transformations map node features $\mathbf{h}$ into a latent representation that combines pharmacophoric types with distance encodings.

### A.4 Encoder implementation

The encoder was implemented as a GNN $f_\Theta : \mathcal{G} \to \mathbb{R}_+^D$ that maps a given graph $G$ to the abstract representation vector $\mathbf{z} \in \mathbb{R}_+^D$. The architecture is comprised of an initial embedding block, three subsequent convolution blocks, followed by a pooling layer, and a projection block.

**Embedding block** The embedding block receives the pharmacophore graph $G_i$ as the tuple $\mathbf{x}_i = [\mathbf{h}_i, \mathbf{e}_i]$, with the OHE feature matrix $\mathbf{h}_i$ and the edge attributes $\mathbf{e}_i$. Initial node feature embeddings are created from the OHE features with a fully-connected (FC) dense layer with learnable weights $\mathbf{W}$ and bias $\mathbf{b}$:

$$\mathbf{h}_i \leftarrow \mathbf{W}\mathbf{h}_i + \mathbf{b} \tag{9}$$

**Convolution block** The convolution block consists of a graph convolution layer, which is implemented as edge-conditioned convolution operator (NNConv), the update rule is described in Section A.3. The network further consists of batch normalization layers (BN), GELU activation functions, and dropout layers. The hidden representation $\mathbf{h}_i^l$ of graph $G_i$ is updated at block $l$ as follows:

$$[\mathbf{h}_i^l, \mathbf{e}_i] \rightarrow \{\text{NNConv} \rightarrow \text{BN} \rightarrow \text{GELU} \rightarrow \text{concat}(\mathbf{h}_i^{l'}, \mathbf{h}_i^l) \rightarrow \text{dropout}\} \rightarrow \mathbf{h}_i^{l+1} \tag{10}$$

where $\mathbf{h}_i^{l'}$ represents the latent representation after activation. Updating the feature matrix $l$ times yields the final node representations of the pharmacophoric points.

**Pooling layer** We employed additive pooling for graph-level read-out $\mathbf{r}_i$, which aggregates the set of $|V|$ node representations $\{\mathbf{h}_1, ..., \mathbf{h}_{|V|}\}_i$ of a Graph $G_i$ by element-wise summation:

$$\mathbf{q}_i = \sum_{k=1}^{|V|} \mathbf{h}_k \tag{11}$$

**Projection block** The projection block maps the graph-level read-out to the positive real number space and is implemented as a multi-layer perceptron $\text{MLP} : \mathbb{R}^d \rightarrow \mathbb{R}_+^D$, where $d$ is the dimension of the vector representation before and $D$ the dimension after the projection. The block consists of $k$ sequential layers of FC layers, BN, ReLU activation, and dropout:

$$\mathbf{q}_i^k \rightarrow \{\text{FC} \rightarrow \text{BN} \rightarrow \text{ReLU} \rightarrow \text{Dropout}\} \rightarrow \mathbf{q}_i^{k+1} \tag{12}$$

The final layer is a FC layer without bias and with positive weights, only:

$$\mathbf{z}_i \leftarrow \text{abs}(\mathbf{W})\mathbf{q}_i \tag{13}$$

Matrix multiplication of the positive learnable weights $\mathbf{W}$ and the output of the last ReLU activation function produces the final representation $\mathbf{z}_i \in \mathbb{R}_+^D$.

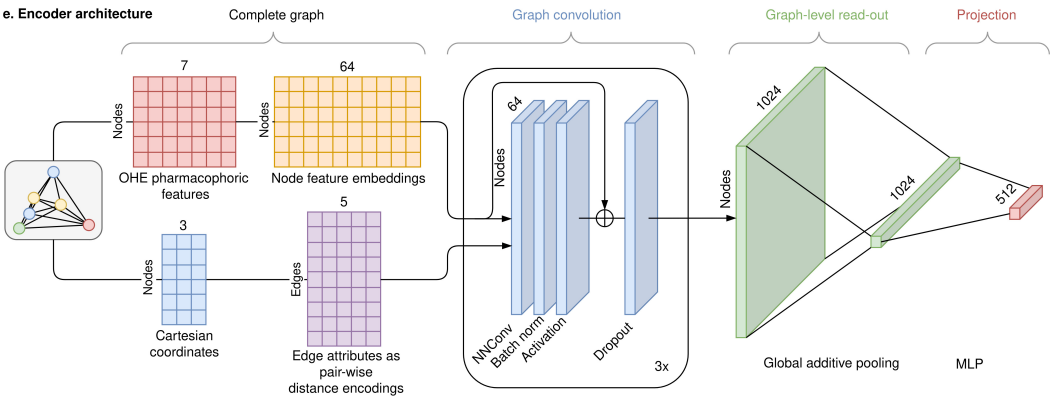


Figure 8: Architecture of the GNN encoder model.

### A.5 Model implementation and training

**Implementation dependencies** The GNN was implemented in Python 3.10 with PyTorch (v2.0.1) and the PyTorch Geometric library (v2.3.1) (Fey et al., 2019). Both, model and dataset, were implemented within the PyTorch Lightning (Falcon et al., 2019) framework (v2.1.0). Model training was monitored with Tensorboard (v2.13.0). CDPKit (v1.1.1) was employed for chemical data processing. Software was installed and executed on a Rocky Linux (v9.4) system with x86-64 architecture.

**Model training** Training was performed on a single NVIDIA GeForce 3090 RTX graphics unit with 24 GB GDDR6X. Training runs were performed for a maximum of 500 epochs with a batch size of 256 pharmacophore graphs. Curriculum learning was applied by gradual enrichment of the dataset with increasingly larger pharmacophore graphs. At training start, only pharmacophore graphs with 4 nodes were considered. After 10 subsequent epochs without considerable minimization of the loss function, pharmacophore graphs with one additional node were added to the training data. The loss function was minimized with the Adam optimizer, we further applied gradient clipping. A training run on the full dataset took approximately 48 hours with the above hardware specifications.

**Hyper parameter tuning & model selection** Hyper parameters were optimized through random parameter selection, the tested ranges are summarized in Table 3. Unlabeled data was split into training and validation data with a 98:2 ratio. Training runs were compared using the AUROC value on the validation data. This was calculated by treating the positive and negative pairs as binary labels, and the predictions were based on their respective order embedding penalty, which was calculated with Equation (3). Hyper parameter optimization was performed on a reduced dataset with 100,000 graphs, which took approximately 5 hours per run. The best performing models were retrained on the full dataset. The hyper parameters of the final encoder model are summarized in Table 2. After model selection, the final model performance was tested on virtual screening datasets.

Table 2: Hyper parameters of the best performing encoder model

| Hyper parameter | |
|---|---|
| batch size | 256 |
| dropout convolution block | 0.2 |
| dropout projection block | 0.2 |
| max. epochs | 500 |
| hidden dimension convolution block | 64 |
| hidden dimension projection block | 1024 |
| output dimension convolution block | 1024 |
| output dimension projection block | 512 |
| learning rate optimizer | 0.001 |
| margin for negative pairs | 100.0 |
| number of convolution blocks | 3 |
| depth of the projector MLP | 3 |
| edge attributes dimension | 5 |
| sampling sphere radius positive pairs | 1.5 |
| sampling surface radius negative pairs | 1.5 |

Table 3: Tested hyper parameter ranges for model training.

| Hyper parameter | |
|---|---|
| dropout | [0.2, 0.3, 0.4, 0.5] |
| margin for negative pairs | [0.1, 0.5, 1, 2, 5, 10, 100] |
| output dimension projection block | [64, 128, 256, 512, 1024] |
| displacement sphere radius $r_T$ of positive pairs | [0.25, 0.5, 1.0, 1.5] |

### A.6 Virtual screening

**DUD-E dataset details** General information about the DUD-E targets is summarized in Table 4. For each target we downloaded the receptor structure from the PDB and created the corresponding interaction pharmacophore with the CDPKit. Vector features were converted into undirected pharmacophoric points with LigandScout (Wolber and Langer, 2005). The resulting pharmacophore queries (Figure 9) were used in our virtual screening experiments.

Table 4: DUD-E targets that were selected for bechmarking experiments in this study.

| Target | PDB code | Ligand ID | Active Ligands | Active Conformations | Decoy Ligands | Decoy Conformations | Query Features |
|---|---|---|---|---|---|---|---|
| ACES | 1e66 | HUX | 451 | 10048 | 26198 | 567122 | 6 |
| ADA | 2e1w | FR6 | 90 | 2166 | 5448 | 125035 | 7 |
| ANDR | 2am9 | TES | 269 | 3039 | 14333 | 211968 | 6 |
| EGFR | 2rgp | HYZ | 541 | 12468 | 35001 | 755017 | 7 |
| FA10 | 3kl6 | 443 | 537 | 13343 | 28149 | 638831 | 5 |
| KIT | 3g0e | B49 | 166 | 3703 | 10438 | 224364 | 5 |
| PLK1 | 2owb | 626 | 107 | 2531 | 6794 | 152999 | 6 |
| SRC | 3el8 | PD5 | 523 | 11868 | 34407 | 737864 | 6 |
| THRB | 1ype | UIP | 461 | 11494 | 26894 | 626722 | 7 |
| UROK | 1sqt | UI3 | 162 | 3450 | 9837 | 199204 | 6 |

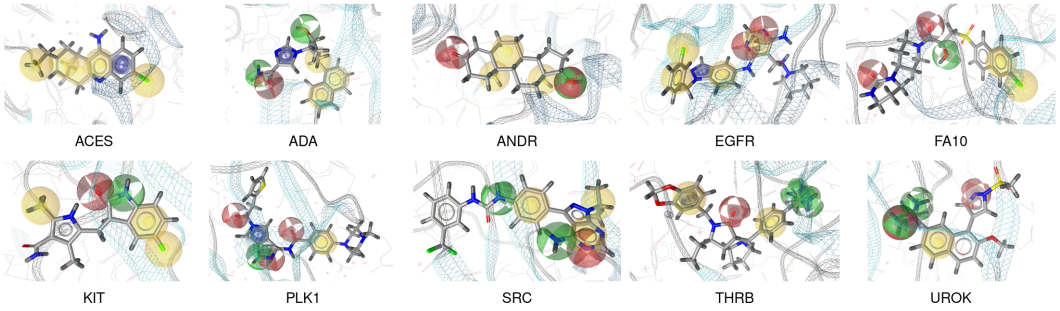


Figure 9: Structure-based pharmacophore queries of ten targets of the DUD-E benchmark dataset.

**CDPKit alignment scoring function** The CDPKit implements alignment as a clique-detection algorithm and computes a rigid-body transformation *via* Kabsch's algorithm to align the pharmacophore query $P_Q$ to the pharmacophore target $P_T$. The goodness of fit is evaluated with a geometric scoring function $S : \mathcal{P} \times \mathcal{P} \to \mathbb{R}_+$:

$$S(P_Q, P_T) = S_{MFP}(P_Q, P_T) + S_{Geom}(P_Q, P_T) \tag{14}$$

where $S_{MFP} : \mathcal{P} \times \mathcal{P} \to \mathbb{Z}_+$ counts the number of matched feature pairs and $S_{Geom} : \mathcal{P} \times \mathcal{P} \to [0, 1)$ evaluates their geometric fit.

**Runtime measurement** We measured alignment runtimes using the psdscreen tool from the CDPKit with 128 threads on an AMD EPYC 7713 64-Core Processor, while embedding and matching runtimes with PharmacoMatch were recorded using an NVIDIA GeForce RTX 3090 GPU with 24 GB GDDR6X. Runtime per pharmacophore was estimated by dividing the total runtime by the number of pharmacophores in each dataset, with the final estimate taken as the mean of ten runs. The results report the mean and standard deviation of these estimates across all ten datasets.

**ROC curves** The performance metrics of our virtual screening experiments are derived from the ROC curves presented in Figures 10 and Figure 11.

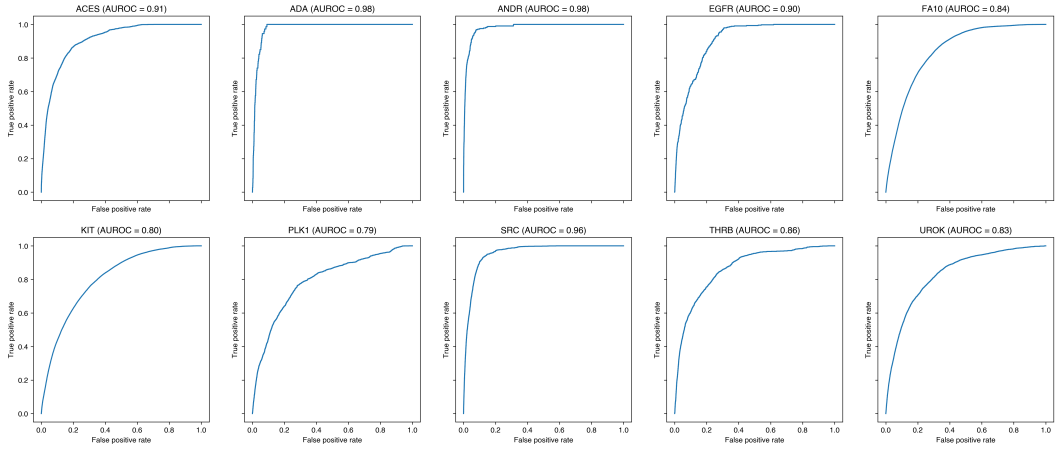


Figure 10: Performance comparison of PharmacoMatch and the alignment algorithm. The ROC-curves display the agreement of the hitlist ranking of the two algorithms for ten targets of the DUD-E benchmark dataset.

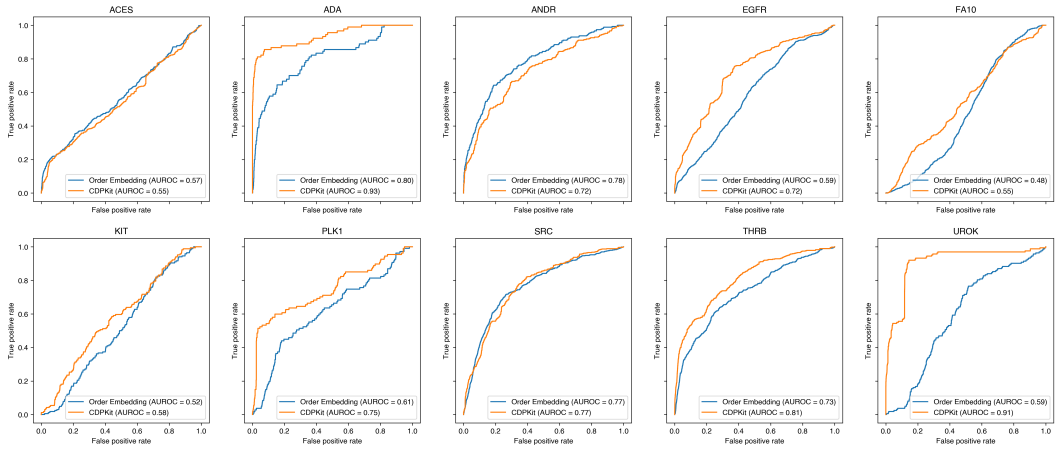


Figure 11: Absolute screening performance of PharmacoMatch and the alignment algorithm performance for ten targets of the DUD-E benchmark dataset. The pharmacophore queries were generated from the respective PDB ligand-receptor structures.

### A.7 Embedding space visualization

**UMAP visualization** UMAP embeddings for visualization plots were calculated with the *UMAP* Python library. The ‘metric‘ parameter was set to Manhattan distance, all other parameters are the default settings of the implementation. We tested a range of hyper parameters to ensure that the visualization results are not sensitive to parameter selection.