Few-Shot In-Context Imitation Learning via Implicit Graph Alignment –Supplementary Material–

Anonymous Author(s)
Affiliation
Address
`email`

1 Local Encoder Network

The representation of object alignment $\mathcal{A}(O_A, O_B)$ as a heterogeneous graph is a crucial step in effectively capturing the relationship between the two objects. To achieve this, we begin by encoding the segmented point clouds of the objects as sets of feature and position pairs. The underlying assumption is that each feature vector can effectively represent the local geometry of a specific part of the object. By treating these feature vectors as nodes in a graph and connecting them with edges that represent their relative positions, we create a graph representation that enables the network to focus on the specific parts of the objects. By adopting this graph-based approach, we are able to shift the network's attention towards local information and individual parts of the objects, rather than relying solely on the global geometry of the entire objects. This localised representation facilitates more precise and targeted reasoning about the alignment between the objects, leading to improved performance in capturing the complex relationships and relative positions between the object parts.

To construct the local features, we follow a step-by-step process. Firstly, we utilise the Furthest Point Sampling (FPS) algorithm to sample K points on the surface of the point cloud (8 in our implementation). These sampled points serve as the centre positions p_i for the subsequent calculation of local embeddings. Next, we group all the points in the original point cloud according to their closest centroid and re-centre them around their respective centroids. This grouping process results in K different point clouds, each representing a distinct part of the object. To encode these local point clouds, we employ a shared PointNet model. This model takes each local point cloud as input and generates a feature vector $\mathcal{F}_i$ that describes the local neighbourhood around each centroid. Our PointNet model consists of an eight-layer MLP (Multi-Layer Perceptron) with skip connections, serving as the backbone for our local encoder. To introduce $\mathbb{SO}(3)$-equivariance to the features, we incorporate Vector Neurons [1] into the linear layer of our network. This approach, as described in the Vector Neurons paper, helps ensure that the features maintain equivariance with respect to rotations in three-dimensional space. Additionally, we include the mean distance to neighbouring points as an additional feature for each point, which helps break the linear dependence between the points. Overall, the local encoder comprises approximately 1.7 million trainable parameters, allowing it to capture and encode the relevant local information from the point clouds.

To enforce, that the local embeddings indeed encode the local geometry, we pre-train them as an implicit occupancy network [2], where a decoder is given a query point and a local feature embedding and is asked to determine whether a query point lies on the surface of the encoded part of the object $D_\theta(\mathcal{F}^i, q) \rightarrow [0, 1]$. Decoder is implemented as a PointNet Model [3] with GeLU activation functions [4] (without Vector Neurons).

We utilise positional encoding [5] and express edge features as $q = (sin(2^0\pi p_q), cos(2^0\pi p_q), ..., sin(2^{L-1}\pi p_q), cos(2^{L-1}\pi p_q))$, where p_q is the position of the query point, and L is the number of frequencies used. In our experiments, we set $L = 10$.

To train the occupancy network as an auto-encoder (as presented in Figure 1), a synthetic dataset is generated, consisting of point clouds for randomly sampled ShapeNet objects and corresponding labelled query points obtained using the PySDF library. This dataset comprises a total of 100,000

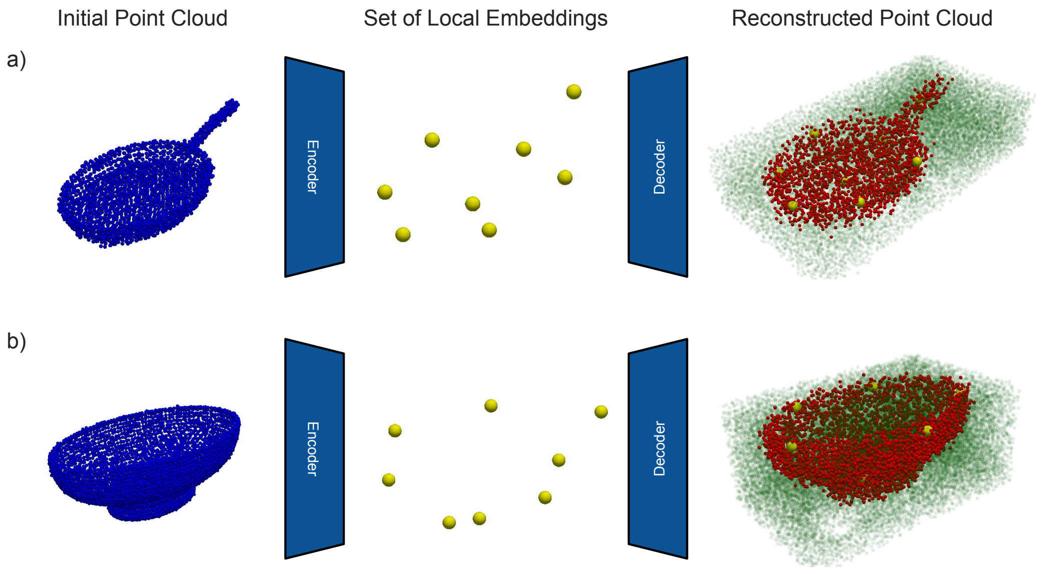

Figure 1: Examples of of our trained auto-encoder when reconstructing a pan (a), and a bowl (b). Blue point clouds represent initial point cloud observations, yellow points represent sampled centroids, and red and green points represent network prediction made for that point, occupied (red) and not occupied (green).

samples. During the training process, two NVIDIA RTX 2080ti GPUs were utilised for computational acceleration. The training duration spanned a period of approximately 3 days. We used AdamW [6] optimiser and scheduler our learning rate using the Cosine Annealing scheduler.

2 Energy Based Model

To learn our proposed alignment distribution $p_\theta(\mathcal{A}^t_{test}|\mathcal{A}^t_{demo})$, we employ an energy-based approach and model the distributions as:

$$p_\theta(\mathcal{A}_{test}|\mathcal{A}_{demo}) = \frac{E_\theta(\mathcal{A}_{test}, \mathcal{A}_{demo}))}{\mathcal{Z}(\mathcal{A}_{test}, \theta)} \tag{1}$$

Here $\mathcal{Z}$ is a normalising constant. In practice, we approximate this, otherwise intractable constant using counter-examples and minimise the negative log-likelihood of

$$\hat{p}_\theta\left(\mathcal{A}_{test}|\mathcal{A}_{demo}, \{\hat{\mathcal{A}}^j_{test}\}^{N_{neg}}_j\right) = \frac{\exp(-E_\theta(\mathcal{A}_{test}, \mathcal{A}_{demo})))}{\exp(-E_\theta(\mathcal{A}_{test}, \mathcal{A}_{demo})) + \sum_j^{N_{neg}} \exp(-E_\theta(\hat{\mathcal{A}}^j_{test}, \mathcal{A}_{demo}))} \tag{2}$$

2.1 Architecture

We are using heterogeneous graphs constructed using features described in Section 1 to represent the alignment between two objects $\mathcal{G}(\{\mathcal{F}^i_A, p^i_A\}^K_i, \{\mathcal{F}^i_B, p^i_B\}^K_i)$. Edges in the graph in the alignment are represented as relative positions between nodes expressed using positional encoding as:

$$e_{ij} = (sin(2^0\pi(p_j - p_i)), cos(2^0\pi(p_j - p_i)), ..., sin(2^{L-1}\pi(p_j - p_i)), cos(2^{L-1}\pi(p_j - p_i))) \tag{3}$$

In our base model, we use $L = 6$. Nodes in the demonstration and test alignment graphs are connected with direction edges equipped with learnable embeddings, effectively propagating information about the demonstration alignments to the test alignment graph. Note, that we are using heterogeneous graphs, meaning different edges (connecting nodes from the same object, target and grasped objects, and connecting demonstration and test graphs) have different types and will be processed with separate learnable parameters. Finally, to make predictions based on the connected graphs, we add an additional type of node to the graph, which aggregates the information from the test alignment graph. This Node can be seen as a $Class$ token, and each is considered alignment in a batch (number of counter-examples + 1) is connected to a separate $Class$ token.

Having the designed graph structure, we use graph transformer convolutions, which can be viewed as a collection of cross-attention modules. These modules facilitate message passing between nodes in the graph, taking into account the specific types of nodes and edges in our heterogeneous graph representation. For a specific type of nodes and edges in the graph, the message passing and attention mechanism can be expressed as:

$$\mathcal{F}_i' = W_1\mathcal{F}_i + \sum_{j\in\mathcal{N}(i)} \alpha_{i,j}\left(W_2\mathcal{F}_j + W_6 e_{ij}\right); \quad \alpha_{i,j} = softmax\left(\frac{(W_3\mathcal{F}_i)^T(W_4\mathcal{F}_j + W_6 e_{ij})}{\sqrt{d}}\right) \tag{4}$$

Embedding from the $Energy$ node (or $Class$ token) is then processed with a small MLP to produce the predicted energy.

Our base model is comprised of 4 graph transformer convolutions with 4 multi-head attention heads, each with a dimension of 64. Final MLP is composed of 2 layers (with dimensions 256) and GeLU activation functions [4]. The full model contains around $5.7M$ trainable parameters.

2.2 Training

To train our proposed energy model, we first need to create alignments of test objects used as counter-examples $\{\hat{\mathcal{A}}^j_{test}\}_j^{N_{neg}}$. We do so by creating copies of $\mathcal{G}_{test}(\{\mathcal{F}_A^i, p_A^i\}_i^K, \{\mathcal{F}_B^i, p_B^i\}_i^K)$, and applying $\mathbb{SE}(3)$ to the nodes in the graph describing the grasped object. Note that demonstration alignment graphs do not need to be copied, as they are connected to the test alignment graph with directional edges, propagating information one way.

To actually transform the nodes in the graph corresponding to the grasped object, both, the position and the feature vectors need to be transformed. Given a transformation T_{noise}, and it's corresponding rotation matrix R_{noise} we update the graph nodes as:

$$[\hat{p}_A, 1] = T_{noise} \times [p_A, 1]^T \quad \hat{\mathcal{F}}_A = R_{noise} \times \mathcal{F}_A \tag{5}$$

Note that this is possible because of the use of SE(3)-equivariant embeddings described in Section 1. During training, we create 256 different $\hat{\mathcal{A}}_{test}$ alignments per batch to approximate $\mathcal{Z}$, each created using a unique T_{noise}. All the counter-examples as alternative graph alignments are created directly on a GPU, facilitating an efficient training phase.

To calculate a set of transformations T_{noise} we use a combination of Langevin Dynamics (described in Section 2.3) and uniform sampling at different scales. We start the training with uniform sampling in ranges of $[-0.8, 0.8]$ metres for translation and $[-\pi, \pi]$ radians for rotation. After N number of optimisation steps ($10K$ in our implementation), we incorporate Langevin Dynamics sampling which we perform every 5 optimisation steps. During this phase, we also reduce the uniform sampling range to $[-0.1, 0.1]$ metres for translation and $[-\pi/4, \pi/4]$ radians for rotation. Although creating negative samples using only Langevin Dynamics is sufficient, in practice, we found that our described sampling strategy leads to faster convergence and more sable training for this specific application of energy-based models.

All models were trained on a single NVIDIA GeForce 3080Ti GPU for approximately 1 day.

During the training of the proposed energy model, several important tricks were employed to ensure stability, efficiency, and smoothness of the energy landscapes for effective gradient-based optimisation. These tricks contribute to the overall training process and facilitate the convergence of the model. The following tricks were identified as particularly significant: $L2$ **Regularisation**: To prevent the logits from diverging towards positive or negative infinity, a small $L2$ regularisation term is added to the loss function. This regularisation term helps to control the magnitude of the logits and maintain stability during training. **Spectral Normalisation**: Spectral normalisation is applied to all layers of the network. In our case, energy landscapes that were learnt without using spectral norms were unusable for gradient-based optimisation. $L2$ **Gradient Penalties**: Gradient penalties are applied to the feature vectors of edges connecting grasped and target objects. This technique imposes an $L2$ regularisation on the gradients, penalising large changes in the input space. By doing so, the energy landscape becomes smoother and more amenable to gradient-based optimisation. **Pre-training on a Subset of the Data**: When dealing with a large and diverse dataset, it is beneficial to initialise the network by pre-training it on a smaller subset of the training data. This pre-training process allows the gradients to flow in regions of the loss-function landscape that would otherwise be relatively flat. As a result, the network can start from a better initialisation point, accelerating the training process. In the specific case mentioned, pre-training on approximately 1,000 samples saved approximately 70% of the total training time.

2.3 Inference Optimisation

Assuming a learnt previously described energy-based model, our goal at inference is to use it to sample from the conditional distribution $p_\theta(\mathcal{A}_{test}|\mathcal{A}_{demo})$. We can not directly sample alignments of objects $\mathcal{A}_{test}$ but we can compute $\mathbb{SE}(3)$ transformation $\mathcal{T}$, that when applied to the grasped object would result in an alignment between the objects that are within the distribution $p_\theta(.)$.

$$\mathcal{T} = \underset{\mathcal{T} \in SE(3)}{\operatorname{argmin}} \; E_\theta(\mathcal{A}_{test}(\mathcal{T} \times O_A, O_B), \mathcal{A}_{demo}) \tag{6}$$

To solve Equation 6, we utilise an iterative gradient-based approach (Langevin Dynamics sampling). Each iteration step k in the optimisation process updates the nodes of the graph alignment representation corresponding to the grasped object as:

$$[p_A^{k+1}, 1] = \frac{\lambda}{2} T_{update}^k \times T_{noise}(\epsilon^k) \times [p_A^k, 1]^T, \quad \hat{\mathcal{F}}_A^{k+1} = R_{update}^k \times R_{noise}(\epsilon^k) \times \mathcal{F}_A^k \tag{7}$$

Here, $\epsilon^k \sim \mathcal{N}(0, \sigma_k^2) \in \mathbb{R}^6$ and T_{noise} is calculated using exponential mapping to project it to $\mathbb{SE}(3)$ as $T_{noise}(\epsilon) = \text{Expmap}(\epsilon)$. In practice, To calculate $T_{update} \in \mathbb{SE}(3)$ (and $R_{update} \in \mathbb{SO}(3)$), we first transform the appropriate nodes in the graph using an identity transformation $T_k \in \mathbb{SE}(3)$ and calculate its gradient using back-propagation as $\nabla_{T_k} E_\theta(\mathcal{A}_{test}(T_I \times O_A, O_B), \mathcal{A}_{demo}) \in \mathbb{R}^6$. Finally, T_{update} is calculated by taking an exponential mapping of $\nabla_{T_k} E_\theta(\cdot)$ as $T_{update}^k = \mathrm{Expmap}(\nabla_{\mathrm{T_k}} \mathrm{E}_\theta(\cdot))$.

3 Experiments

3.1 Task Definitions

We evaluate our approach on six different tasks: 1) *Grasping*. The goal is to grasp different pans by the handle, where success means the pan is lifted by the gripper. 2) *Stacking*. The goal is to stack two bowls, where success means one bowl remains inside the other bowl. 3) *Sweeping*. The goal is to sweep marbles into a dustpan with a brush, where success means that 2 out of the 3 marbles end up in the dustpan. 4) *Hanging*. The goal is to hang a cap onto a deformable stand, where success means the cap rests on the stand. 5) *Inserting*. The goal is to insert a bottle into a shoe, where success means the bottle stays upright in the shoe. 6) *Pouring*. The goal is to pour a marble into a mug, where success means the marble ends up in the mug.

3.2 Exploring Demonstration Diversity

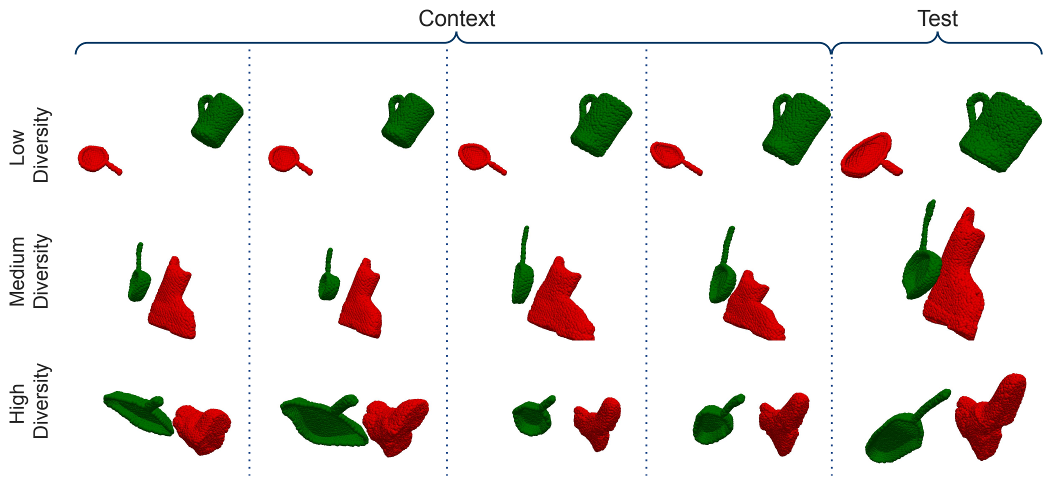

Figure 2: Random samples from the 3 different datasets used for the data diversity experiment. Green point cloud represent object A, while red point cloud represent object B.

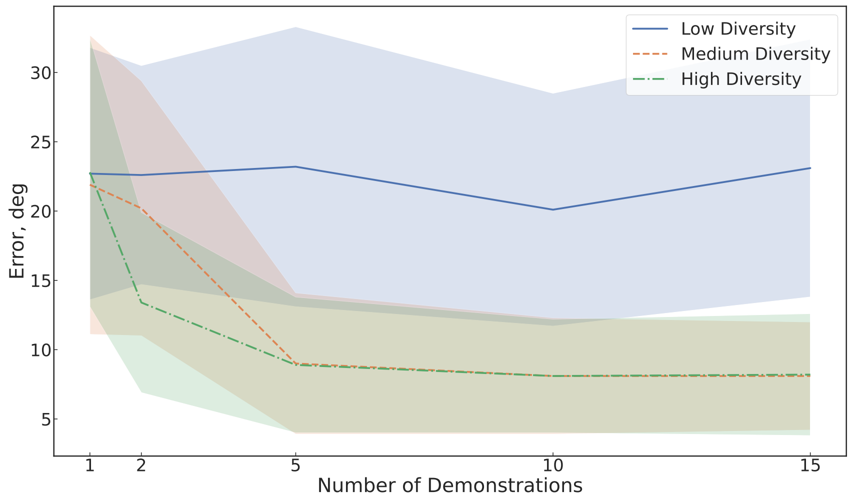

Figure 3: Rotational error based on the number of demonstrations for 3 different sets of diversities.

References

[1] C. Deng, O. Litany, Y. Duan, A. Poulenard, A. Tagliasacchi, and L. J. Guibas. Vector neurons: A general framework for so (3)-equivariant networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12200–12209, 2021.

[2] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019.

[3] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[4] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[5] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[6] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.