A GRAPHLOG

A.1 EXTENDED TERMINOLOGY

In this section, we extend the terminology introduced in Section 3 to introduce the term *descriptor*. A descriptor of a query graph g is computed by concatenating the relations occurring on the shortest path from the source and sink in the query (u, v). For example, if a query graph is described in the form of a list of edges such as $g \rightarrow (0, 1, r_1), (1, 2, r_2), (1, 3, r_3), (2, 4, r_4)$, where r_i are the relations, then the descriptor of this graph given the query $(0, 4)$ would be $D_g = r_1, r_2, r_4$. In all our train, validation and test splits, we split graphs based on the unique set of descriptors, such as to maintain inductive reasoning aspect of the task.

A.2 DATASET GENERATION

This section follows up on the discussion in Section 3.1. We describe all the steps involved in the dataset generation process.

Rule Generation. In Algorithm 1, we describe the complete process of generating rules in GraphLog . We require the set of K relations, which we use to sample the rule set $\mathcal{R}$. Then, we iterate through all possible combinations of relations in DataLog format to sample possible candidate rules. We impose two constraints on the candidate rule: **(i)** No two rules in $\mathcal{R}$ can have the same body. This ensures consistency between the rules. **(ii)** Candidate rules cannot have common relations among the *head* and *body*. This ensures absence of cycles. We also add the inverse rule of our sampled candidate rule and check the same consistencies again. We employ two types of unary Horn clauses to perform the closure of the available rules and to check the consistency of the different rules in $\mathcal{R}$. Using this process, we ensure that all generated rules are sound and consistent with respect to $\mathcal{R}$.

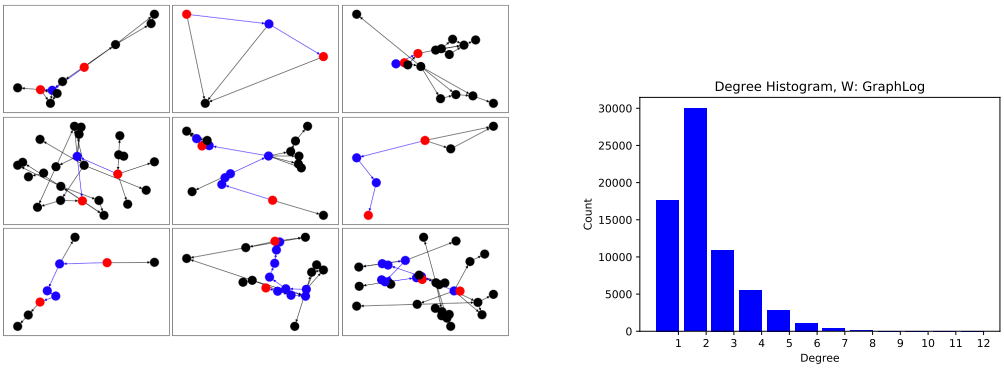
World Sampling. From the set of rules in $\mathcal{R}$, we partition rules into buckets for different worlds (Algorithm 2). We use a simple policy of bucketing via a sliding window of width w with stride s, to classify rules pertaining to each world. For example, two such consecutive worlds can be generated as $\mathcal{R}^t = [\mathcal{R}_i \dots \mathcal{R}_{i+w}]$ and $\mathcal{R}^{t+1} = [\mathcal{R}_{i+s} \dots \mathcal{R}_{i+w+s}]$. (Algorithm 2) We randomly permute $\mathcal{R}$ before bucketing in-order.

Graph Generation. This is a two-step process where first we sample a *world graph* (Algorithm 3) and then we sample individual graphs from the *world graph* (Algorithm 4). Given a set of rules $\mathcal{R}_\mathcal{S}$, in the first step, we recursively sample and apply rules in $\mathcal{R}_\mathcal{S}$ to generate a *relation graph* called *world graph*. This sampling procedure enables us to create a diverse set of *world graphs* by considering only certain subsets (of $\mathcal{R}$) during sampling. By controlling the extent of overlap between the subsets of $\mathcal{R}$ (in terms of the number of rules that are common across the subsets), we can precisely control the *similarity* between the different *world graphs*. By selecting subsets which have higher dissimilarity between each other, we introduce more diversity in terms of logical rules.

In the second step (Algorithm 4), the *world graph* is used to sample a set of query graphs $G_W^S = \{g_1, \cdots g_N\}$. A query graph g_i is sampled from G_W by sampling a pair of nodes (u, v) from G_W and then by sampling a *resolution path* $p_{G_W}^{u,v}$. The edge $r_i(u, v)$ provides the target relation that the learning model has to predict. Since the *relation* for the edge $r_i(u, v)$ can be *resolved* by composing the relations along the *resolution* path, the relation prediction task tests for the compositional generalization abilities of the models. We first sample all possible resolution paths and get their individual descriptors D_{g_i}, which we split in training, validation and test splits. We then construct the training, validation and testing graphs by first adding all edges of an individual D_{g_i} to the corresponding query graph g_i, and then sampling neighbors of p_{g_i}. Concretely, we use Breadth First Search (BFS) to sample the neighboring subgraph of each node $u \in p_{g_i}$ with a decaying selection probability γ. This allows us to create diverse input graphs while having precise control over its resolution by its descriptor D_{g_i}. Splitting the dataset over these descriptor paths ensures inductive generalization.

A.3 COMPUTING SIMILARITY

GraphLog provides precise control for categorizing the similarity between different worlds by computing the overlap of the underlying rules. Concretely, the similarity between two worlds W^i and



(a) (b)

Figure 7: Figure 7(a) represents graphs drawn at random from different worlds available in GraphLog . Nodes in red denote the query nodes, and the nodes in blue denote the shortest path among the query nodes. Figure 7(b) represents the degree distribution of all graphs from all worlds in GraphLog .

Algorithm 1 Rule Generator

Input: Set of K relations $\{r_i\}_K, K > 0$
Define an empty rule set $\mathcal{R}$
for all $r_i \in \{r_i\}_K$ **do**
  for all $r_j \in \{r_i\}_K$ **do**
    for all $r_k \in \{r_i\}_K$ **do**
      Define candidate rule $t : [r_i, r_j] \implies r_k$
      if Cyclical rule, i.e. $r_i == r_k$ OR $r_j == r_k$ **then**
        Reject rule
      end if
      if $t[body] \notin \mathcal{R}$ **then**
        Add t to $\mathcal{R}$
      end if
    end for
  end for
end for
Check and remove any further cyclical rules.

W^j is defined as $\text{Sim}(W^i, W^j) = |\mathcal{R}^i \cap \mathcal{R}^j|$, where W_i and W_j are the graph worlds and $\mathcal{R}^i$ and $\mathcal{R}^j$ are the set of rules associated with them. Thus GraphLog enables various training scenarios - training on highly similar worlds or training on a mix of similar and dissimilar worlds. This fine grained control allows GraphLog to mimic both in-distribution and out-of-distribution scenarios - during training and testing. It also enables us to precisely categorize the effect of multi-task pre-training when the model needs to adapt to novel worlds.

Algorithm 2 Partition rules into overlapping sets

Require: Rule Set $\mathcal{R}_\mathcal{S}$
Require: Number of worlds $n_w > 0$
Require: Number of rules per world $w > 0$
Require: Overlapping increment stride $s > 0$
  for $i = 0; i < |\mathcal{R}_\mathcal{S}| - w;$ **do**
    $\mathcal{R}_i = \mathcal{R}_\mathcal{S}[i; i + w]$
    $i = i + s$
  end for

Algorithm 3 World Graph Generator

Require: Set of relations $\{r_i\}_K, K > 0$
Require: Set of rules derived from $\{r_i\}_K$, $|\mathcal{R}| > 0$
Require: Set rule selection probability gamma $\gamma = 0.8$
  Set rule selection probability $P[\mathcal{R}[i]] = 1, \forall i \in |\mathcal{R}|$
Require: Maximum number of expansions $s \geq 2$
Require: Set of available nodes N, s.t. $|N| \geq 0$
Require: Number of cycles of generation $c \geq 0$
  Set *WorldGraph* set of edges $G_m = \emptyset$
  while $|N| > 0$ or $c > 0$ **do**
    Randomly choose an expansion number for this cycle: steps = $\texttt{rand}(2, s)$
    Set added edges for this cycle $E_c = \emptyset$
    for all step in steps **do**
      if step = 0 **then**
        With uniform probability, either:
        Sample r_t from $\mathcal{R}_{\mathcal{S}}[head]$ and sample $u, v \in N$ without replacement, OR
        Sample an edge (u, r_t, v) from G_m
        Add (u, r_t, v) to E_c and G_m
      else
        Sample an edge (u, r_t, v) from E_c
      end if
      Sample a rule $\mathcal{R}[i]$ from $\mathcal{R}$ following P s.t. $[r_i, r_j] \implies r_t$
      $P[\mathcal{R}[i]] = P[\mathcal{R}[i]] * \gamma$
      Sample a new node $y \in N$ without replacement
      Add edge (u, r_i, y) to E_c and G_m
      Add edge (y, r_j, v) t E_c and G_m
    end for
    if All rules in $\mathcal{R}$ is used atleast once **then**
      Increment c by 1
      Reset rule selection probability $P[\mathcal{R}[i]] = 1, \forall i \in |\mathcal{R}|$
    end if
  end while

A.4 COMPUTING DIFFICULTY

Recent research in multitask learning has shown evidence that models prioritize selection of difficult tasks over easy tasks while learning to boost the overall performance (Guo et al., 2018). Thus, GraphLog also provides a method to examine how pretraining on tasks of different difficulty level affects the adaptation performance. Due to the stochastic effect of partitioning of the rules, GraphLog consists of datasets with varying range of difficulty. We use the supervised learning scores (Table 6) as a proxy to determine the the relative difficulty of different datasets. We cluster the datasets such that tasks with prediction accuracy greater than or above 70% are labeled as *easy* difficulty, 50-70% are labeled as *medium* difficulty and below 50% are labeled as *hard* difficulty dataset. We find that the labels obtained by this criteria are consistent across the different models (Figure 3).

Graph properties affecting difficulty. While we compute difficulty based on the proxy of supervised learning scores, we observe that the relative difficulty of the tasks are highly correlated with the number of *descriptors* (Section A.1) available for each task. We can control the distribution of the dataset by explicitly requiring diversity in the descriptors. Worlds having less number of descriptors are prone to be more difficult for the task. This is due to the fact that will less available descriptors with respect to the budget of data samples, our generation module samples the same set of descriptors while adding variable noise. Thus, datasets with low descriptor count ends up with more relative noise. This shows that for a learner, a dataset with enough variety among the resolution paths of the graphs with less noise is relatively easier to learn compared to the datasets which has less variation and more noise.

Algorithm 4 Graph Sampler

Require: Rule Set $\mathcal{R}_{\mathcal{S}}$
Require: World Graph $G_m = (V_m, E_m)$
Require: Maximum Expansion length $e > 2$
  Set Descriptor set $S = \emptyset$
  for all $u, v \in E_m$ **do**
    Get all walks $Y_{(u,v)} \in G_m$ such that $|Y_{(u,v)}| \leq e$
    Get all descriptors $D_{Y_{(u,v)}}$ for all walks $Y_{(u,v)}$
    Add $D_{Y_{(u,v)}}$ to S
  end for
  Set train graph set $G_{train} = \emptyset$
  Set test graph set $G_{test} = \emptyset$
  Split descriptors in train and test split, S_{train} and S_{test}
  for all $D_i \in S_{train}$ or S_{test} **do**
    Set source node $u_s = D_i[0]$ and sink node $v_s = D_i[-1]$
    Set prediction target $t = E_m[u_s][v_s]$
    Set graph edges $g_i = \emptyset$
    Add all edges from D_i to g_i
    for all $u, v \in D_i$ **do**
      Sample Breadth First Search connected nodes from u and v with decaying probability γ
      Add the sampled edges to g_i
    end for
    Remove edges in g_i which create shorter paths between u_s and v_s
    Add (g_i, u_s, v_s, t) to either G_{train} or G_{test}
  end for

B SUPERVISED LEARNING ON GRAPHLOG

We perform extensive experiments over *all* the datasets available in GraphLog (statistics given in Table 6). We observe that in general, for the entire set of 57 worlds, the `GAT_E-GAT` model performs the best.

C MULTITASK LEARNING

C.1 MULTITASK LEARNING ON DIFFERENT DATA SPLITS BY DIFFICULTY

		Easy	Medium	Difficult
f_r	f_c	Accuracy	Accuracy	Accuracy
GAT	E-GAT	**0.729** $\pm$0.05	**0.586** $\pm$0.05	0.414 $\pm$0.07
Param	E-GAT	0.728 $\pm$0.05	0.574 $\pm$0.06	0.379 $\pm$0.06
GCN	E-GAT	0.713 $\pm$0.05	0.55 $\pm$0.06	0.396 $\pm$0.05
GAT	RGCN	0.695 $\pm$0.04	0.53 $\pm$0.03	**0.421** $\pm$0.06
Param	RGCN	0.551 $\pm$0.08	0.457 $\pm$0.05	0.362 $\pm$0.05
GCN	RGCN	0.673 $\pm$0.05	0.514 $\pm$0.04	0.396 $\pm$0.06

Table 4: Inductive performance on data splits marked by difficulty

In Section A.4 we introduced the notion of *difficulty* among the tasks available in GraphLog . Here, we consider a set of experiments where we perform multitask training and inductive testing on the worlds bucketized by their relative difficulty (Table 4). We sample equal number of worlds from each difficulty bucket, and separately perform multitask training and testing. We evaluate the average prediction accuracy on the datasets within each bucket. We observe that the average multitask performance *also* mimics the relative task difficulty distribution. We find `GAT-E-GAT` model outperforms other baselines in *Easy* and *Medium* setup, but is outperformed by `GAT-RGCN` model in the *Difficult* setup. For each model, we used the same architecture and hyperparameter settings across the buckets. Optimizing individually for each bucket may improve the relative performance.

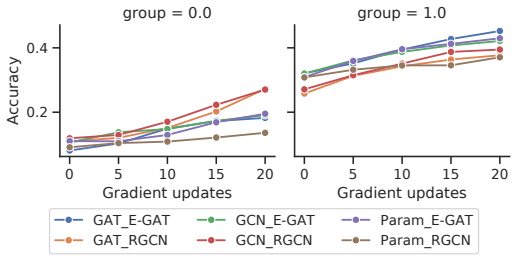


Figure 8: We perform fine-grained analysis of *few shot* adaptation capabilities in Multitask setting. Group 0.0 and 1.0 corresponds to 0% and 100% similarity respectively.

		Easy	Medium	Difficult
f_r	f_c	Accuracy	Accuracy	Accuracy
GAT	E-GAT	0.531 ±0.03	**0.569** ±0.01	**0.555** ±0.04
Param	E-GAT	0.520 ±0.02	0.548 ±0.01	0.540 ±0.01
GCN	E-GAT	**0.555** ±0.01	0.561 ±0.02	0.558 ±0.01
GAT	RGCN	0.502 ±0.02	0.532 ±0.01	0.532 ±0.01
Param	RGCN	0.535 ±0.01	0.506 ±0.04	0.539 ±0.04
GCN	RGCN	0.481 ±0.02	0.516 ±0.02	0.520 ±0.01
Mean		0.521	**0.540**	0.539

Table 5: Convergence performance on 3 held out datasets when pre-trained on easy, medium and hard training datasets

C.2 Multitask Pre-training by task similarity

In the main paper (Section 5.2) we introduce the setup of performing multitask pre-training on GraphLog datasets and adaptation on the datasets based on relative similarity. Here, we perform fine-grained analysis of *few-shot* adapatation capabilities of the models. We analyze the adaptation performance in two settings - when the adaptation dataset has complete overlap of rules with the training datasets (*group*=1.0) and when the adaptation dataset has zero overlap with the training datasets (*group*=0.0). We find RGCN family of models with a graph based representation function has faster adaptation on the dissimilar dataset, with `GCN-RGCN` showing the fastest improvement. However on the similar dataset the models follow the ranking of the supervised learning experiments, with `GAT-EGAT` model adapting comparitively better.

C.3 Multitask Pre-training by task difficulty

Using the notion of *difficulty* introduced in Section A.4, we perform the suite of experiments to evaluate the effect of pre-training on *Easy*, *Medium* and *Difficult* datasets. Interestingly, we find the performance on convergence is better on Medium and Hard datasets on pre-training, compared to the Easy dataset (Table 5). This behaviour is also mirrored in k-shot adaptation performance (Figure **??**), where pre-training on Hard dataset provides faster adaptation performance on 4/6 models.

D Continual Learning

A natural question arises following our continual learning experiments in Section 5.3 : does the *order* of difficulty of the worlds matter? Thus, we perform an experiment following Curriculum Learning (Bengio et al., 2009) setup, where the order of the worlds being trained is determined by their relative difficulty (which is determined by the performance of models in supervised learning setup, Table 6, i.e., we order the worlds from easier worlds to harder worlds). We observe that while the current task accuracy follows the trend of the difficulty of the worlds (Figure 10(a)), the mean of past accuracy is significantly worse. This suggests that a curriculum learning strategy might not be optimal to learn graph representations in a continual learning setting. We also performed the same experiment with sharing only the composition and representation functions (Figure 10(b)), and observe similar trends where sharing the representation function reduces the effect of catastrophic forgetting.

World ID	NC	ND	Split	ARL	AN	AE	D	M1	M2	M3	M4	M5	M6
rule_0	17	286	train	4.49	15.487	19.295	Hard	0.481	0.500	0.494	0.486	0.462	0.462
rule_1	15	239	train	4.10	11.565	13.615	Hard	0.432	0.411	0.428	0.406	0.400	0.408
rule_2	17	157	train	3.21	9.809	11.165	Hard	0.412	0.357	0.373	0.347	0.347	0.319
rule_3	16	189	train	3.63	11.137	13.273	Hard	0.429	0.404	0.473	0.373	0.401	0.451
rule_4	16	189	train	3.94	12.622	15.501	Medium	0.624	0.606	0.619	0.475	0.481	0.595
rule_5	14	275	train	4.41	14.545	18.872	Hard	0.526	0.539	0.548	0.429	0.461	0.455
rule_6	16	249	train	5.06	16.257	20.164	Hard	0.528	0.514	0.536	0.498	0.495	0.476
rule_7	17	288	train	4.47	13.161	16.333	Medium	0.613	0.558	0.598	0.487	0.486	0.537
rule_8	15	404	train	5.43	15.997	19.134	Medium	0.627	0.643	0.629	0.523	0.563	0.569
rule_9	19	1011	train	7.22	24.151	32.668	Easy	0.758	0.744	0.739	0.683	0.651	0.623
rule_10	18	524	train	5.87	18.011	22.202	Medium	0.656	0.654	0.663	0.596	0.563	0.605
rule_11	17	194	train	4.29	11.459	13.037	Medium	0.552	0.525	0.533	0.445	0.456	0.419
rule_12	15	306	train	4.14	11.238	12.919	Easy	0.771	0.726	0.603	0.511	0.561	0.523
rule_13	16	149	train	3.58	11.238	13.549	Hard	0.453	0.402	0.419	0.347	0.298	0.344
rule_14	16	224	train	4.14	11.371	13.403	Hard	0.448	0.457	0.401	0.314	0.318	0.332
rule_15	14	224	train	3.82	12.661	15.105	Hard	0.494	0.423	0.501	0.402	0.397	0.435
rule_16	16	205	train	3.59	11.345	13.293	Hard	0.318	0.332	0.292	0.328	0.306	0.291
rule_17	17	147	train	3.16	8.163	8.894	Hard	0.347	0.308	0.274	0.164	0.161	0.181
rule_18	18	923	train	6.63	25.035	33.080	Easy	0.700	0.680	0.713	0.650	0.641	0.618
rule_19	16	416	train	6.10	17.180	20.818	Easy	0.790	0.774	0.777	0.731	0.729	0.702
rule_20	20	2024	train	8.63	34.059	45.985	Easy	0.830	0.799	0.854	0.756	0.741	0.750
rule_21	13	272	train	4.58	10.559	11.754	Medium	0.621	0.610	0.632	0.531	0.516	0.580
rule_22	17	422	train	5.21	16.540	20.681	Medium	0.586	0.593	0.628	0.530	0.506	0.573
rule_23	15	383	train	4.97	17.067	21.111	Hard	0.508	0.522	0.493	0.455	0.473	0.476
rule_24	18	879	train	6.33	21.402	26.152	Easy	0.706	0.704	0.743	0.656	0.641	0.638
rule_25	15	278	train	3.84	11.093	12.775	Hard	0.424	0.419	0.382	0.358	0.345	0.412
rule_26	15	352	train	4.71	14.157	17.115	Medium	0.565	0.534	0.532	0.466	0.461	0.499
rule_27	16	393	train	4.98	14.296	16.499	Easy	0.713	0.714	0.722	0.632	0.604	0.647
rule_28	16	391	train	4.82	17.551	21.897	Medium	0.575	0.564	0.571	0.503	0.499	0.552
rule_29	16	144	train	3.87	10.193	11.774	Hard	0.468	0.445	0.475	0.325	0.336	0.389
rule_30	17	177	train	3.51	10.270	11.764	Hard	0.381	0.426	0.382	0.357	0.316	0.336
rule_31	19	916	train	5.90	20.147	26.562	Easy	0.788	0.789	0.770	0.669	0.674	0.641
rule_32	16	287	train	4.66	16.270	20.929	Medium	0.674	0.671	0.700	0.621	0.594	0.615
rule_33	18	312	train	4.50	14.738	18.266	Medium	0.695	0.660	0.709	0.710	0.679	0.668
rule_34	18	504	train	5.00	15.345	18.614	Easy	0.908	0.888	0.906	0.768	0.762	0.811
rule_35	19	979	train	6.23	21.867	28.266	Easy	0.831	0.750	0.782	0.680	0.700	0.662
rule_36	19	252	train	4.66	13.900	16.613	Easy	0.742	0.698	0.698	0.659	0.627	0.651
rule_37	17	260	train	4.00	11.956	14.010	Easy	0.843	0.826	0.826	0.673	0.698	0.716
rule_38	17	568	train	5.21	15.305	20.075	Easy	0.748	0.762	0.733	0.644	0.630	0.719
rule_39	15	182	train	3.98	12.552	14.800	Easy	0.737	0.642	0.635	0.592	0.603	0.587
rule_40	17	181	train	3.69	11.556	14.437	Medium	0.552	0.584	0.575	0.525	0.472	0.479
rule_41	15	113	train	3.58	10.162	11.553	Medium	0.619	0.601	0.626	0.490	0.468	0.470
rule_42	14	95	train	2.96	8.939	9.751	Hard	0.511	0.472	0.483	0.386	0.393	0.395
rule_43	16	162	train	3.36	11.077	13.337	Medium	0.622	0.567	0.579	0.473	0.482	0.437
rule_44	18	705	train	4.75	15.310	18.172	Hard	0.538	0.561	0.603	0.498	0.519	0.450
rule_45	15	151	train	3.39	9.127	10.001	Medium	0.569	0.580	0.592	0.535	0.524	0.524
rule_46	19	2704	train	7.94	31.458	43.489	Easy	0.850	0.820	0.828	0.773	0.762	0.749
rule_47	18	647	train	6.66	22.139	27.789	Easy	0.723	0.667	0.708	0.620	0.649	0.611
rule_48	16	978	train	6.15	17.802	21.674	Easy	0.812	0.798	0.812	0.772	0.763	0.753
rule_49	14	169	train	3.41	9.983	11.177	Easy	0.714	0.734	0.700	0.511	0.491	0.615
rule_50	16	286	train	3.99	12.274	16.117	Medium	0.651	0.653	0.656	0.555	0.583	0.570
rule_51	16	332	valid	4.44	16.384	21.817	Easy	0.746	0.742	0.738	0.667	0.657	0.689
rule_52	17	351	valid	4.81	16.231	20.613	Medium	0.697	0.716	0.754	0.653	0.655	0.670
rule_53	15	165	valid	3.65	10.838	12.378	Hard	0.458	0.464	0.525	0.334	0.364	0.373
rule_54	13	303	test	5.25	13.503	15.567	Medium	0.638	0.623	0.603	0.587	0.586	0.555
rule_55	16	293	test	4.83	16.444	20.944	Medium	0.625	0.582	0.578	0.561	0.528	0.571
rule_56	15	241	test	4.40	14.010	16.702	Medium	0.653	0.681	0.692	0.522	0.513	0.550
AGG	16.33	428.94		4.70	14.89	18.37		**0.618 / 26**	0.603 / 10	0.611 / 20	0.530 / 1	0.526 / 0	0.539 / 0

Table 6: Results on Single-task supervised setup for all datasets in GraphLog. Abbreviations: **NC**: Number of Classes, **ND**: Number of Descriptors, **ARL**: Average Resolution Length, **AN**: Average number of nodes, **AE**: Average number of edges
, **D**: Difficulty, **AGG**: Aggregate Statistics. List of models considered : **M1**: GAT-EGAT, **M2**: GCN-E-GAT, **M3**: Param-E-GAT, **M4**: GAT-RGCN, **M5**: GCN-RGCN and **M6**: Param-RGCN. Difficulty is calculated by taking the scores of the model (M1) and partitioning the worlds according to their accuracy (≥ 0.7 = Easy, ≥ 0.54 and < 0.7 = Medium, and < 0.54 = Hard). We provide both the mean of the raw accuracy scores for all models, as well as the number of times the model is ranked first in all the tasks.

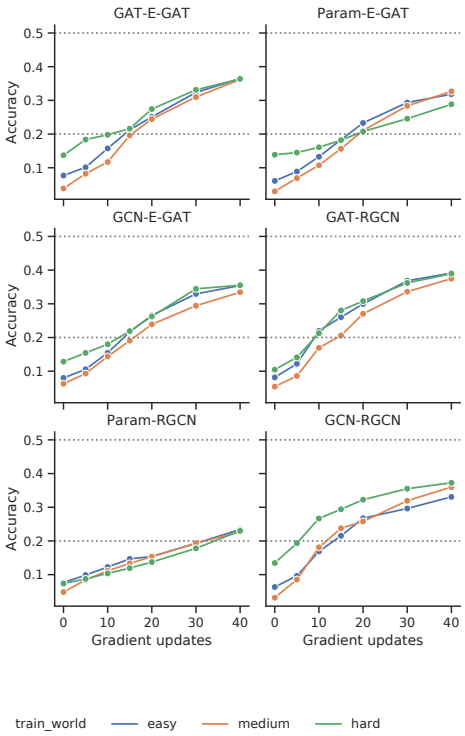


Figure 9: We evaluate the effect of k-shot adaptation on held out datasets when pre-trained on *easy*, *medium* and *hard* training datasets, among the different model architectures. Here, k ranges from 0 to 40.

E Hyperparameters and Experimental Setup

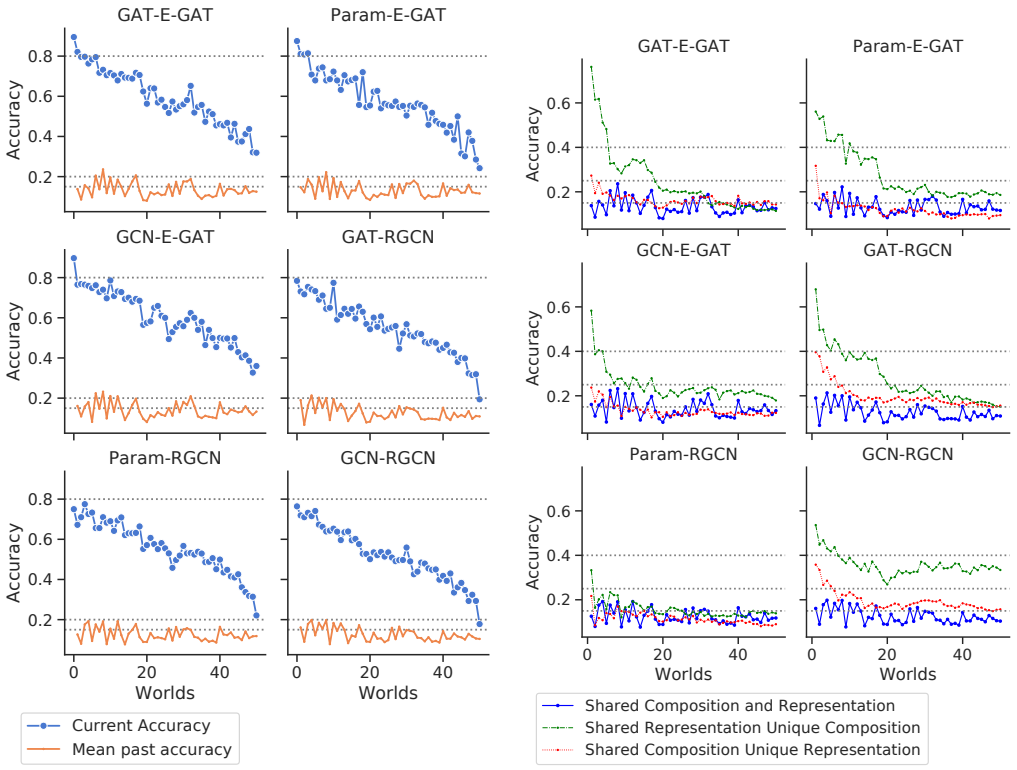
In this section, we provide detailed hyperparameter settings for both models and dataset generation for the purposes of reproducibility. The codebase and dataset used in the experiments are attached with the Supplementary materials, and will be made public on acceptance.

E.1 Dataset Hyperparams

We generate GraphLog with 20 relations or classes (K), which results in 76 rules in $\mathcal{R}_{\mathcal{S}}$ after consistency checks. For unary rules, we specify half of the relations to be symmetric and other half to have their invertible relations. To split the rules for individual worlds, we choose the number of rules for each world $w = 20$ and stride $s = 1$, and end up with 57 worlds $\mathcal{R}_0 \dots \mathcal{R}_{56}$. For each world $\mathcal{R}_i$, we generate 5000 training, 1000 testing and 1000 validation graphs.

E.2 Model Hyperparams

For all models, we perform hyper-parameter sweep (grid search) to find the optimal values based on the validation accuracy. For all models, we use the relation embedding and node embedding to be 200 dimensions. Since the nodes in GraphLog does not have any features or attributes, we randomly initialize the embeddings in the GNN message passing layers for each epoch for all experiments. We train all models with Adam optimizer with learning rate 0.001 and weight decay of 0.0001. For supervised setting, we train all models for 500 epochs, and we add a scheduler for learning rate to decay it by 0.8 whenever the validation loss is stagnant for 10 epochs. In multitask setting, we sample a new task every epoch from the list of available tasks. Here, we run all models for 2000 epochs when we have the number of tasks ≤ 10. For larger number of tasks (Figure 4), we train by proportionally increasing the number of epochs compared to the number of tasks. (2k epochs for 10 tasks, 4k epochs for 20 tasks, 6k epochs for 30 tasks, 8k epochs for 40 tasks and 10k epochs for



(a) (b)

Figure 10: Curriculum Learning strategy in Continual Learning setup of GraphLog. Figure 10(a) presents the current task accuracy (blue) and mean of all previous task accuracy (orange). Figure 10(b) presents the mean of previous task accuracy when either the composition function or the representation function is shared for all worlds.

50 tasks). For continual learning experiment, we train each task for 100 epochs for all models. No learning rate scheduling is used for either multitask or continual learning experiments. Individual model hyper-parameters are as follows:

- Representation functions :
  - `GAT` : Number of layers = 2, Number of attention heads = 2, Dropout = 0.4
  - `GCN` : Number of layers = 2, with symmetric normalization and bias, no dropout
- Composition functions:
  - `E-GAT`: Number of layers = 6, Number of attention heads = 2, Dropout = 0.4
  - `RGCN`: Number of layers = 2, no dropout, with bias.