GraphLog: A Benchmark for Measuring Logical Generalization in Graph Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

Relational inductive biases have a key role in building learning agents that can generalize and reason in a compositional manner. While relational learning algorithms such as graph neural networks (GNNs) show promise, we do not understand their effectiveness to adapt to new tasks. In this work, we study the logical generalization capabilities of GNNs by designing a benchmark suite grounded in first-order logic. Our benchmark suite, GraphLog, requires that learning algorithms perform rule induction in different synthetic logics, represented as knowledge graphs. GraphLog consists of relation prediction tasks on 57 distinct procedurally generated logical worlds. We use GraphLog to evaluate GNNs in three different setups: single-task supervised learning, multi-task (with pre-training), and continual learning. Unlike previous benchmarks, GraphLog enables us to precisely control the logical relationship between the different worlds by controlling the underlying first-order logic rules. We find that models' ability to generalize and adapt strongly correlates to the availability of diverse sets of logical rules during multi-task training. We also find the severe catastrophic forgetting effect in continual learning scenarios, and GraphLog provides a precise mechanism to control the distribution shift. Overall, our results highlight new challenges for the design of GNN models, opening up an exciting area of research in generalization using graph-structured data.

1 Introduction

Relational reasoning, or the ability to reason about the relationship between objects and entities, is considered one of the fundamental aspects of intelligence [Krawczyk et al., 2011] [Halford et al., 2010], and is known to play a critical role in cognitive growth of children [Son et al., 2011] [Farrington-Flint et al., 2007] [Richland et al., 2010]. This ability to infer relations between objects/entities/situations, and to compose relations into higher-order relations, is one of the reasons why humans quickly learn how to solve new tasks [Holyoak and Morrison, 2012] [Alexander, 2016]. The perceived importance of relational reasoning for generalization has fueled the development of several neural network architectures that incorporate relational inductive biases [Battaglia et al., 2016] [Santoro et al., 2017] [Battaglia et al., 2018]. Graph neural networks (GNNs), in particular, have emerged as a dominant computational paradigm within this growing area [Scarselli et al., 2008] [Hamilton et al., 2017a] [Gilmer et al., 2017] [Schlichtkrull et al., 2018] [Du et al., 2019]. However, despite the growing interest in GNNs and their promise of relational generalization, we currently lack an understanding of how effectively these models can adapt and generalize across distinct tasks.

In this work, we study the task of logical generalization in the context of relational reasoning using GNNs. One example of such a reasoning task (from everyday life) can be in the context of a family graph where the nodes are family members, and edges represent the relationships (brother, father, etc). The objective is to learn logical rules, which are compositions of a specific form, such as “the son of my son is my grandson”. As new compositions of existing relations are discovered during the lifetime of a learner, (e.g., the son of my daughter is my grandson), it should remember the old compositions, even as it learns new compositions (just like we humans do). This simplistic example can be extended to the more complex (and yet practical) scenarios like identifying novel chemical compositions, or recommender systems, where agents have to learn and retain compositions of existing relations.
We study the effect of such generalization by analyzing the ability of GNNs in learning new relation compositions, leveraging first-order logic. In particular, we study how GNNs can induce logical rules and generalize by combining such rules in novel ways after training. We propose a benchmark suite, GraphLog, that is grounded in first-order logic. Figure 1 shows the setup of the benchmark. Given a set of logical rules, we create a diverse set of logical worlds with different underlying logics. Our setup allows us to control the distribution shift by controlling the similarity between the different worlds, in terms of the overlap in logical rules between different worlds.

Our analysis provides the following insights about logical generalization capabilities of GNNs:

- Two architecture choices for GNNs strongly (and positively) impact generalization: 1) incorporating multi-relational edge features using attention, and 2) modularising GNN architecture to include a parametric representation function, to learn representations for the relations using a knowledge graph structure.

- In the multi-task setting, training a model on a more diverse set of logical worlds improves generalization and adaptation performance.

- All the evaluated models exhibit catastrophic forgetting. This indicates that the models are not learning transferable representations and compositions and just overfitting to the current task —highlighting the challenge of lifelong learning in context of logical generalization and GNNs.

## 2 BACKGROUND AND RELATED WORK

**GNNs.** Several GNN architectures have been proposed to learn representations of the graph inputs (Scarselli et al., 2008; Duvenaud et al., 2015; Defferrard et al., 2016; Kipf and Welling, 2016; Gilmer et al., 2017; Hamilton et al., 2017b; Schlichtkrull et al., 2018). Previous works have focused on evaluating GNNs in terms of their expressive power (Barceló et al., 2019; Morris et al., 2019; Xu et al., 2018), usefulness of features (Chen et al., 2019), and explaining their predictions (Ying et al., 2019). Complementing these works, we evaluate GNN models on the task of logical generalization.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>IR</th>
<th>D</th>
<th>LG</th>
<th>M</th>
<th>S</th>
<th>Me</th>
<th>Mu</th>
<th>LL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEVR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CoGen</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CLUTR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCAN (De and Farley, 2017)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SQoOP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TextWorld (Coté et al., 2018)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GraphLog (Proposed)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 1: Features of related datasets that: 1) test compositional generalization and reasoning, and 2) are procedurally generated. We compare the datasets along the following axis: Inspectable Rules (IR), Diversity, Compositional Generalization (CG), Modality and if the following training setups are supported: Supervised, Meta-learning, Multitask & Continual learning (CL).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of relations</th>
<th>Total number of WorldGraphs</th>
<th>Total number of unique rules</th>
<th>Training Query graphs per WorldGraph</th>
<th>Validation Query graphs per WorldGraph</th>
<th>Testing Query graphs per WorldGraph</th>
<th>Average number of nodes per Query graph</th>
<th>Average number of edges per Query graph</th>
<th>Average number of triangles per Query graph</th>
<th>Number of rules per WorldGraph</th>
<th>Maximum length of resolution path</th>
<th>Minimum length of resolution path</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td>57</td>
<td>76</td>
<td>5000</td>
<td>1000</td>
<td>1000</td>
<td>14.55</td>
<td>17.94</td>
<td>11.27</td>
<td>20</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 2: Aggregate statistics of the worlds in GraphLog. Statistics for each individual world are in the Appendix.

Figure 1: GraphLog setup: A set of rules (grounded in propositional logic) is partitioned into overlapping subsets. It is used to define unique worlds. Within each world $W_k$, several knowledge graphs $g_i^k$ (governed by rule set of $W_k$) are generated.
Knowledge graph completion. Many knowledge graph datasets are available for relation prediction tasks (also known as knowledge base completion). Prominent examples include Freebase15K (Bordes et al., 2013), WordNet (Miller 1995), NELL (Mitchell and Fredkin 2014), and YAGO (Suchanek et al., 2007; Hoffart et al., 2011; Mahdisoltani et al., 2013). Since these datasets are derived from real-world knowledge graphs, they are generally noisy and incomplete, and many facts are not available in the underlying knowledge bases (West et al., 2014; Paulheim, 2017). Moreover, the underlying logical rules are often opaque and implicit (Guo et al., 2016), thus reducing the utility of these datasets for understanding the logical generalization capability of neural networks.

Procedurally generated datasets for reasoning. Several procedurally generated benchmarks have been proposed to study the relational reasoning and compositional generalization properties of neural networks (Table 1). These datasets provide a controlled testbed for evaluating compositional reasoning capabilities of neural networks in isolation. Based on these insightful works, we enumerate the four key desiderata that, we believe, such a benchmark should provide:

1. Human interpretable rules should be used to generate the dataset.
2. The datasets should be diverse, and the compositional rules used to solve different tasks should be distinct, so that adaptation on a new task is not trivial. The degree of similarity across the tasks should be configurable to evaluate the role of diversity in generalization.
3. The benchmark should test for compositional generalization, which is the ability to solve unseen combinations of existing rules, thus generalizing through the composition of known concepts.
4. The benchmark should support creating a large number of tasks and enable a more fine-grained inspection of the generalization capabilities of the model in different setups, e.g., supervised learning, multi-task learning, and continual learning.

As shown in Table 1, GraphLog is unique in satisfying all of these desiderata. We highlight that unlike previous works which have been largely focused on the image and text modalities, GraphLog is one of the unique attempts to test logical generalization on graph-structured data using Graph Neural Networks. CLUTRR (Sinha et al., 2019) also provides similar underlying graphs in the text corpus, although GraphLog is different based on the following properties:

- CLUTRR consists of a single rule world which in turn contains only 15 rules. GraphLog extends the concept to a more general framework where a user can define how many rules can occur in a single world, as well as define multiple such worlds.
- GraphLog allows building multiple worlds consisting of either the same, overlapping, or distinct set of rules - which allows practitioners to test multi-task and continual learning scenarios in minute detail by controlling the distribution shift, which is a key difference with CLUTRR.
- CLUTRR is built on a static set of relations (22 family relations) while GraphLog can contain any number of such relations since it’s a flexible generator along with a dataset.

Synthetic Graph Generation. Synthetic graph generation is extensively studied using various neural generative approaches for scale-free random graphs (You et al., 2018; Liao et al., 2019). Unlike these approaches, GraphLog is built using a procedural graph generator, which allows us to control the logical validity of the query graphs, which is defined in Section 3. Control over logical validity is easier in synthetically generated datasets, which are used extensively in Inductive Logic Programming (ILP) / Rule Learning context (Cornelio and Thost, 2019). GraphLog on the other hand, is targeted towards Graph Neural Network evaluation, which also supports for richer distribution shift evaluation through Multi-task learning and Continual learning scenarios.

3 GraphLog

Background and Terminology. We leverage terminology and definitions from both knowledge graph and logic programming literature. A graph \( G = (V_G, E_G) \) is a collection of a set of nodes \( V_G \) and a set of edges \( E_G \) between the nodes. We assume that the graphs are relational, meaning that each edge between two nodes (say \( u \) and \( v \)) is assigned a label, and can be denoted as \( r_i(u, v) \in E_G \). A relation set \( \mathcal{R} \) is a set of relations \( \{ r_1, r_2, ..., r_K \} \).

http://ilp16.doc.ic.ac.uk/competition
In logic programming terms, nodes in the graph correspond to \textit{constants}, and edges correspond to \textit{ground atoms}. Thus, as a slight abuse and mixture of notations, existence of an edge in the graph, i.e., \( r_i(u, v) \in E_G \), is equivalent to background assumption that ground atom is true (i.e., that \( r_i(u, v) \leftarrow \)). We define a \textit{rule set} \( \mathcal{R} \) as set of \textit{dyadic definite Datalog clauses} (Abiteboul et al., 1995) of form:

\[
    r_k(U, V) \leftarrow r_i(U, Z) \land r_j(Z, V).
\]

Note that Equation 1 is also termed as \textit{chain-rule} in logic literature. Following standard convention, we use upper-case to denote variables (e.g., the variables in Equation 1 can be substituted for nodes) and lower-case for constants (e.g., to denote nodes \( v \in V_G \)). The relations \( r_i, r_j \) form the \textit{body} while the relation \( r_k \) forms the \textit{head} of the rule. Path-based Horn clauses of this form represent a limited and well-defined subset of first-order logic. They encompass the types of logical rules learned by state-of-the-art rule induction systems for knowledge graph completion (Das et al. (2017); Evans and Grefenstette, 2017; Meilicke et al., 2018; Sadeghian et al., 2019; Teru et al., 2020; Yang et al., 2017; Zhang et al., 2019) and are thus a useful synthetic test-bed.

We use \( p_{G}^{u,v} \) to denote a \textit{path} from node \( u \) to \( v \) in a graph \( G \) (i.e., a sequence unique nodes connected by edges). In logical terms, \( p_{G}^{u,v} \) corresponds to the conjunction of all the edge predicates in the path.

We construct graphs according to rules of the form in Equation 1 so that a path between two nodes, and applying rules in \( \mathcal{R} \) recursively sample and use rules in \( \mathcal{R} \). As discussed in Section 2, we want our proposed benchmark to provide four key desiderata: (i) specific rule generation is given in Appendix (Algorithm 1).

Thus, by following the path between two nodes, and applying rules of the form of Equation 1 according to the edges of the path, we can always \textit{resolve} the relationship between the nodes.

\textbf{Problem Setup.} Given this setup, the task is to predict the relation between a pair of nodes. In particular, we define a query \((g, u, v)\) as follows: (i) the \textit{query subgraph} \( g \subset G \) is a subgraph of the full graph \( G \); (ii) the \textit{query nodes} are present in the query subgraph (i.e., \( u, v \in V_g \)); (iii) the edge between the query nodes is \textit{not} present in the query subgraph, but the edge \( g \) is present in the underlying graph (i.e., \( r \subgraph g \in E_G, r \notin E_g \)). The goal is to infer the missing edge between query nodes \( r \subgraph g \). We assume that the learning agent has access to a set of training queries \( g_1, \ldots, g_n \subset G \) to optimize a prediction model before being tested on a set of held-out queries \( g_{n+1}, \ldots, g_{n+w} \subset G \).

In knowledge graph terms, this is a form of link or relation prediction, as our goal is to infer the missing edge between two nodes. Unlike most work on knowledge graph completion, we emphasize an \textit{inductive} problem setup, where the query graph \( g \) in each training and testing example is unique, requiring generalization to unseen graphs (Teru et al., 2020).

Finally, note that we will sometimes refer to the full graphs \( G \) as \textit{world graphs}, in order to distinguish them from the \textit{query graphs} \( g \subset G \). We make this distinction because GraphLog involves link prediction over several different \textit{logical worlds}, denoted \( W = (G_W, \mathcal{R}_W) \), with each defined with its own underlying world graph \( G_W \) and rule set \( \mathcal{R}_W \). Our setup enables controlling the overlap between the rules in these different logical worlds, allowing for a unique ability to quantify the logical generalization capabilities of a learning agent.

\subsection{Dataset Generation}

As discussed in Section 2, we want our proposed benchmark to provide four key desiderata: (i) interpretable rules, (ii) diversity, (iii) compositional generalization and (iv) large number of tasks. We describe how our dataset generation process ensures all four aspects.

\textbf{Rule generation.} We create a set \( \mathcal{R} \) of possible relations and use it to create a randomly generated rule set \( \mathcal{R} \), such that two key constraints are satisfied: (i) No two rules in \( \mathcal{R} \) can have the same body, ensuring that Equation 2 is satisfied. (ii) Rules cannot have common relations among the \textit{head} and \textit{body}, ensuring the absence of cyclic dependencies in rules. Generating the dataset using a consistent and well-defined rule set ensures interpretability in the resulting dataset. The full algorithm for rule generation is given in Appendix (Algorithm 1).

\textbf{Graph generation.} The graph generation process has two steps: In the first step, we sample overlapping subsets of \( \mathcal{R}_W \subset \mathcal{R} \) to create individual \textit{worlds} (as shown in Figure 1). In each world, we recursively sample and use rules in \( \mathcal{R}_W \) to generate the world graph \( G_W \) for that world. This
sampling procedure creates a diverse set of world graphs by considering only certain subsets of $\mathcal{R}$. By controlling the extent of overlap between the subsets of $\mathcal{R}$ (in terms of the number of common rules), we can precisely control the similarity between the different worlds. The full algorithm for generating the world graphs and controlling the similarity between the worlds is given in Appendix (Algorithm 3 and Section A.2).

In the second step, the world graph $G_W$ in each world is used to sample a set of query graphs $G_{W}^q = \{g_1, \cdots, g_N\}$ for that specific world (shown as Step (a) in Figure 2). 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 path $p_{G_W}^{u,v}$. The edge $r_i(u, v)$ between the source and sink node of the path provides the target relation for the learning model to predict. To increase the complexity of the sampled $g_i$ graphs (beyond being simple paths), we also add nodes to $g_i$ by sampling neighboring nodes of the nodes on $p_{G_W}^{u,v}$, such that no other shorter path exists between $u$ and $v$. Algorithm 4 (in the Appendix) details our graph sampling approach. Note that this sampling process leads to rich and complex graph structures, including cycles and motifs (see Figure 7(a) in Appendix for example of graphs).

**GraphLog Dataset Suite.** We use the above data generation process to instantiate a dataset suite with 57 distinct logical worlds with 5000 query graphs per world (Figure 1). The dataset is split into train, validation, and test worlds. The query graphs within each world are also split into train, validation, and test sets (Table 2). Though we instantiate 57 worlds, the GraphLog generator can instantiate an arbitrary number of worlds and has been included in the supplementary material.

### 3.2 Setups supported in GraphLog

**Supervised learning.** A model is trained (and evaluated) on the train (and test split) of a world. The number of rules grows exponentially with the number of relations $K$, making it impossible to train on all possible combinations of the relations. We expect that a perfectly systematic model inductively generalizes to unseen combinations of relations by training only on a subset of combinations.

**Multi-task learning.** GraphLog provides multiple logical worlds, with their training and evaluation splits. A model is trained on (and evaluated on) the train (and test) splits of several worlds ($W_1, \cdots, W_M$). GraphLog enables us to control the complexity of each world and similarity between the worlds to evaluate how model performance varies across similar vs. dissimilar worlds. GraphLog is designed to study the effect of pre-training on adaptation. In this setup, the model is pre-trained on the train split of multiple worlds ($W_1, \cdots, W_M$) and fine-tuned on the train split of the unseen heldout worlds ($W_{M+1}, \cdots, W_N$). The model is evaluated on the test split of the heldout worlds. GraphLog enables mimicking in-distribution and out-of-distribution train (and test) scenarios and quantify the effect of multi-task pre-training for adaptation performance.

**Continual learning.** The model is trained on a sequence of worlds. Before training on a new world, it is evaluated on all the worlds that it has trained on so far. Given the challenges involved in continual learning, we do not expect the models to perform well on the previous tasks. Nonetheless, given that we are evaluating the models for relational reasoning and our datasets share relations, we expect the models to retain some knowledge of previous tasks. We use the performance on the previous tasks to infer if the models learn to solve the relational reasoning tasks or just fit to the current dataset distribution.

**Controlling similarity.** An unique feature of GraphLog is it allows practitioners to control the distribution shift when evaluating multi-task and continual learning baselines. In GraphLog the diversity of worlds can be controlled via similarity. Concretely, the similarity between two worlds $W^i$ and $W^j$ 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.

---

2Due to the current formulation (Equation 2), GraphLog is restricted to have at most a single edge between a pair of nodes. This formulation is chosen to make the task simpler in order to analyze the effects of generalization in detail. In future work, we will be exploring the effect of multiple edges using this benchmark, which is a characteristic of many real-world knowledge graphs.
4 \ Models

In this section, we describe the message passing graph neural networks that we evaluate on GraphLog benchmark. These models take as input the query \((g_i, u, v)\) and a learnable relation embedding \(r_i \in \mathbb{R}^d\) (Step (d) and (e) in Figure 2), and output the relation as a k-way softmax (Step (f) in Figure 2). We use both attention-based and non-attention based models.

**Relational Graph Convolutional Network (RGCN).** For a relational graph, the RGCN model (Schlichtkrull et al., 2018) is a natural choice for a baseline architecture. In this approach, we iterate a series of message passing operations:

\[
    h_u^{(t)} = \text{ReLU} \left( \sum_{r_i \in R} \sum_{v \in N_{r_i}(u)} r_i \times 1 \mathcal{T} \times 3 h_{v}^{(t-1)} \right),
\]

where \(h_u^{(t)} \in \mathbb{R}^d\) denotes representation of a node \(u\) at the \(t^{th}\) layer of the model, \(T \in \mathbb{R}^{d_r \times d \times d}\) is a learnable tensor, \(r \in \mathbb{R}^d\) is relation representation, and \(N_{r_i}(u)\) denotes the neighbors of \(u\) (related by \(r_i\)), \(\times j = 1, 2, 3\) denotes multiplication across the modes of the tensor. RGCN model learns a relation-specific propagation matrix, specified by the interaction between relation embedding \(r_i\) and shared tensor \(T\).

**Edge-based Graph Attention Network (Edge-GAT).** Many recent works have highlighted the importance of the attention mechanism, especially in the context of relational reasoning (Vaswani et al., 2017; Santoro et al., 2018; Schlag et al., 2019). Motivated by this observation, we investigate an extended version of the GAT (Velickovic et al., 2017), in order to handle the edge types. We complement GAT by incorporating gating via an LSTM (Hochreiter and Schmidhuber, 1997) and an extended version of the GAT (Veličković et al., 2017), in order to handle the edge types. We complement GAT by incorporating gating via an LSTM (Hochreiter and Schmidhuber, 1997) and where the attention is conditioned on both the incoming message (from the other nodes) and the relation embedding (of the other nodes): \(m^{(u)} = \sum_{r_i \in R} \sum_{v \in N_{r_i}(u)} \alpha \left( h_{u}^{(t-1)}, h_{v}^{(t-1)}, r \right), h_{u}^{(t)} = \text{LSTM}(m^{(u)}, h_{u}^{(t-1)})\). Following the original GAT model, the attention function \(\alpha\) uses concatenated input vectors. We refer to this model as the Edge GAT (E-GAT) model.

**Query and node representations.** We predict the relation for query \((g_i, u, v)\) by concatenating \(h_u^{(K)}, h_v^{(K)}\) (final-layer query node embeddings) and applying a feedforward network (Step (f) in Figure 2). In absence of node features, we randomly initialize all the node embeddings in the GNNs.

**Representation functions** GNN models described above expect to learn \(d\)-dimensional relation representations. We can further re-structure them notation-wise into two modules: a) **Representation module** which is represented as a function \(f_r : G_W \times R \rightarrow \mathbb{R}^d\), that maps logical relations within a particular world \(W\) to \(d\)-dimensional vector representations, and b) **Composition module**, which is a function \(f_c : G_W \times V_{G_W} \times V_{G_W} \times \mathbb{R}^{d \times |R|} \rightarrow R\), that learns to compose the relation representations learned by \(f_r\) and answer queries over a knowledge graph. In this formulation, the relation embedding matrix \(r_i \in \mathbb{R}^d\) used in RGCN and Edge-GAT can be thought of the output of a direct parameterization module \(\text{Param}\), which is devoid of any input and simply returns the embeddings from a lookup table. Thus these two baselines can be redefined as \text{Param}-RGCN and \text{Param}-E-GAT models, where RGCN and E-GAT are the composition modules. A major limitation of this approach is that the relation representations are optimized specifically for each logical world, and there is no inductive bias towards learning representations that can generalize.

**Learning representations from the graph structure.** To define a more powerful representation function, we consider learning relation representations as a function of the graph \(G_W\) underlying a logical world \(W\). We consider an “extended” form of \(G_W\), which is a graph with additional edges that connect nodes of the same type.

This extended graph is used to learn the relation representations \(r\) via a GNN model. The composition function takes as input the query \((g_i, u, v)\) and the relational representation \(r\), and outputs the relation as a k-way softmax. The composition function predicts the relation between the nodes \(u\) and \(v\).
denoted $\hat{G}_W$, which introduces new nodes (called edge-nodes) corresponding to each edge in the original world graph $G_W$. For an edge $r(u,v) \in E_G$, the corresponding edge-node $(u - r - v)$ is connected to only those nodes that were incident to it in the original graph (i.e. nodes $u$ and $v$; see Figure 2, Step (b)). This new graph $\hat{G}_W$ only has one type of edge and comprises of nodes from both the original graph and from the set of edge-nodes. We learn the relation representations by training a GNN model on $\hat{G}_W$ and by averaging the edge-node embeddings corresponding to each relation type $r_i \in \mathcal{R}$, (Step (c) in Figure 2). For the GNN model, we consider the Graph Convolutional Network (GCN) [Kipf and Welling, 2016] and the Graph Attention Network (GAT) architectures. The intuition behind creating the extended-graph is that the representation GNN function can learn the relation embeddings based on the structure of the complete relational graph $G_W$. Both the representation module and the composition modules are trained end-to-end to predict the relations. Thus, we end up with four more models to evaluate GraphLog, namely GCN-RGCN, GCN-E-GAT, GAT-RGCN and GAT-E-GAT.

5.1 Single Task Supervised Learning

We evaluate the models on all the 57 worlds, one model-world pair at a time (342 combinations). With GraphLog, it is difficult for one model to outperform others on all the 57 datasets just by exploiting dataset-specific bias, thereby making the conclusions more robust. We categorize the worlds in three categories of difficulty—easy, moderate and difficult—based on the relative test performance of the models on each world and present the results in Figure 3. (A complimentary formulation of difficulty is explained in Appendix A.4). We observe that the models using E-GAT composition function always outperform the models using RGCN. This confirms our hypothesis about leveraging attention to improve the performance on relational reasoning tasks. Interestingly, the relative ordering among the worlds, in terms of the test accuracy of the different models, is consistent irrespective of the model we use, highlighting the intrinsic ordering of difficulty of the different worlds in GraphLog.

5.2 Multi-Task Training

Standard multi-task training. First, we evaluate how changing similarity among training worlds affects test performance in the multi-task setup. We train a model on eight worlds and test on
three distinct worlds. In Table 3, we observe that mix of similar and dissimilar worlds improves generalization capabilities of all the models. Similar to the trends observed in the single task setup, the GAT-EGAT model consistently performs either as good as or better than other models. The models using EGAT for the composition function perform better than the ones using the RGCN model. Figure 4 shows how the models’ performance changes when we perform multi-task training on an increasingly large set of worlds. Interestingly, we see that model performance improves as the number of worlds is increased from 10 to 20 but then begins to decline, indicating model capacity saturation in the presence of too many diverse worlds.

Multi-task pre-training. In this setup, we pre-train the model on multiple worlds and adapt on a heldout world. We study how the models’ adaptation capabilities vary as the similarity between the training and the evaluation distributions changes. Figure 5 considers the case of zero-shot adaptation and adaptation till convergence. As we move along the x-axis, the zero-shot performance (shown with solid colors) decreases in all the setups. This is expected as the similarity between the training and the evaluation distributions also decreases. An interesting trend is that the model’s performance, after adaptation, increases as the similarity between the two distributions decreases. This suggests that training over a diverse set of distributions improves adaptation capability.

5.3 Continual Learning Setup

We evaluate knowledge retention capability by training GNNs on a sequence of worlds (arranged by relative similarity). After convergence on any world, we report average of model’s performance on all previous worlds as it trains on new worlds. Figure 6(a) shows the rapid degradation in model’s performance on the previous worlds as it trains on new worlds, highlighting the limitation of current models for continual learning.

The role of the representation function. We also investigate the model’s performance in a continual learning setup where the model learns only a world-specific representation function or a world-specific composition function, and where the other module is shared across the worlds. In Figure 6(b), we observe that sharing the representation function reduces the effect of catastrophic forgetting, but sharing the composition function does not have the same effect. This suggests that the representation function learns representations that are useful across the worlds.

6 Discussion & Conclusion

In this work, we propose GraphLog, a benchmark suite for evaluating the logical generalization capabilities of GNNs. GraphLog is grounded in first-order logic and provides access to a large number of diverse tasks that require compositional generalization to solve, including single task supervised learning, multi-task learning, and continual learning. Our results highlight the importance of attention
Figure 6: Evaluation of models in continual learning setup. In Figure 6(a), the blue curve shows the accuracy on the current world and the orange curve shows the mean of accuracy on all previously seen worlds. Training on new worlds, degrades model’s performance on the past worlds (catastrophic forgetting). In Figure 6(b) sharing representation function reduces catastrophic forgetting.

mechanisms and modularity to achieve logical generalization, while also highlighting open challenges related to multi-task and continual learning in the context of GNNs. A natural direction for future work is leveraging GraphLog for studying fast adaptation and meta-learning in the context of logical reasoning (e.g., via gradient-based meta-learning), as well as integrating state-of-the-art methods (e.g., regularization techniques) to combat catastrophic forgetting in the context of GNNs.

REFERENCES


A GraphLog

A.1 Extended Terminology

In this section, we extend the terminology introduced in Section [3.1] 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 \( \mathcal{R}^i = [\mathcal{R}_i \ldots \mathcal{R}_{i+w}] \) and \( \mathcal{R}^{i+1} = [\mathcal{R}_{i+s} \ldots \mathcal{R}_{i+w+s}] \). (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 \( \mathcal{R}_S \), in the first step, we recursively sample and apply rules in \( \mathcal{R}_S \) 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 \( G^S_W = \{g_1, \ldots, g_N \} \). 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 \( p^u \) from \( G_W \). 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 \( \gamma \). 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
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 $R$
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 $t: [r_i, r_j] \Rightarrow r_k$
      if Cyclical rule, i.e. $r_i == r_k$ OR $r_j == r_k$ then
        Reject rule
      end if
      if $t[body] \not\in R$ then
        Add $t$ to $R$
      end if
    end for
  end for
end for
Check and remove any further cyclical rules.

$W_i^j$ is defined as $\text{Sim}(W_i, W_j) = |R_i \cap R_j|$, where $W_i$ and $W_j$ are the graph worlds and $R_i$ and $R_j$ 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 $R_S$
Require: Number of worlds $n_w > 0$
Require: Number of rules per world $w > 0$
Require: Overlapping increment stride $s > 0$
for $i = 0; i < |R_S| - w; \text{do}$
  $R_i = R_S[i; i + w]$
  $i = i + s$
end for
Algorithm 3 World Graph Generator

Require: Set of relations \( \{ r_i \}_K, K > 0 \)
Require: Set of rules derived from \( \{ r_i \}_K, |R| > 0 \)
Require: Set rule selection probability gamma \( \gamma = 0.8 \)
Set rule selection probability \( P[|R[i]|] = 1, \forall i \in |R| \)
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 \( \text{WorldGraph} \) set of edges \( G_m = \emptyset \)

while \( |N| > 0 \) or \( c > 0 \) do

Randomly choose an expansion number for this cycle: \( \text{steps} = \text{rand}(2, s) \)
Set added edges for this cycle \( E_c = \emptyset \)

for all step in \( \text{steps} \) do

if \( \text{step} = 0 \) then

With uniform probability, either:
Sample \( r_t \) from \( R_S[\text{head}] \) 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 \( R[i] \) from \( R \) following \( P \) s.t. \( [r_t, r_j] \implies r_t \)
Set \( P[|R[i]|] = P[|R[i]|] * \gamma \)
Sample a new node \( y \in N \) without replacement
Add edge \( (u, r_t, y) \) to \( E_c \) and \( G_m \)
Add edge \( (y, r_j, v) \) to \( E_c \) and \( G_m \)
end for

if All rules in \( R \) is used atleast once then
Increment \( c \) by 1
Reset rule selection probability \( P[|R[i]|] = 1, \forall i \in |R| \)
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 $\mathcal{R}_S$
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 $\gamma$
    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

<table>
<thead>
<tr>
<th></th>
<th>Easy</th>
<th>Medium</th>
<th>Difficult</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>$J_2$</td>
<td>Accuracy</td>
<td>Accuracy</td>
</tr>
<tr>
<td>GAT</td>
<td>E-GAT</td>
<td>0.729 ±0.05</td>
<td>0.586 ±0.05</td>
</tr>
<tr>
<td>Param</td>
<td>E-GAT</td>
<td>0.728 ±0.05</td>
<td>0.574 ±0.06</td>
</tr>
<tr>
<td>GCN</td>
<td>E-GAT</td>
<td>0.713 ±0.05</td>
<td>0.55 ±0.06</td>
</tr>
<tr>
<td>GAT</td>
<td>RGCN</td>
<td>0.695 ±0.04</td>
<td>0.53 ±0.03</td>
</tr>
<tr>
<td>Param</td>
<td>RGCN</td>
<td>0.551 ±0.08</td>
<td>0.457 ±0.05</td>
</tr>
<tr>
<td>GCN</td>
<td>RGCN</td>
<td>0.673 ±0.05</td>
<td>0.514 ±0.04</td>
</tr>
</tbody>
</table>

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.
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 adaptation 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-E-GAT model adapting comparatively 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 9), 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.
### Table 6: Results on Single-task supervised setup for all datasets in GraphLog.

<table>
<thead>
<tr>
<th>World ID</th>
<th>NC</th>
<th>ND</th>
<th>Split</th>
<th>ARL</th>
<th>AN</th>
<th>AE</th>
<th>D</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>M6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>17</td>
<td>Medium</td>
<td>4.58</td>
<td>15.59</td>
<td>17.15</td>
<td>Hard</td>
<td>0.54</td>
<td>0.52</td>
<td>0.51</td>
<td>0.48</td>
<td>0.52</td>
<td>0.56</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>18</td>
<td>Medium</td>
<td>0.64</td>
<td>0.65</td>
<td>0.65</td>
<td>Easy</td>
<td>0.54</td>
<td>0.54</td>
<td>0.52</td>
<td>0.50</td>
<td>0.52</td>
<td>0.54</td>
</tr>
<tr>
<td>3</td>
<td>19</td>
<td>18</td>
<td>Medium</td>
<td>0.56</td>
<td>0.57</td>
<td>0.57</td>
<td>Easy</td>
<td>0.52</td>
<td>0.54</td>
<td>0.53</td>
<td>0.52</td>
<td>0.54</td>
<td>0.56</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>20</td>
<td>Medium</td>
<td>0.55</td>
<td>0.56</td>
<td>0.56</td>
<td>Easy</td>
<td>0.53</td>
<td>0.55</td>
<td>0.53</td>
<td>0.52</td>
<td>0.55</td>
<td>0.57</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>21</td>
<td>Medium</td>
<td>0.58</td>
<td>0.59</td>
<td>0.59</td>
<td>Easy</td>
<td>0.54</td>
<td>0.56</td>
<td>0.55</td>
<td>0.53</td>
<td>0.55</td>
<td>0.57</td>
</tr>
<tr>
<td>6</td>
<td>22</td>
<td>22</td>
<td>Medium</td>
<td>0.62</td>
<td>0.63</td>
<td>0.63</td>
<td>Easy</td>
<td>0.58</td>
<td>0.60</td>
<td>0.59</td>
<td>0.57</td>
<td>0.58</td>
<td>0.61</td>
</tr>
<tr>
<td>7</td>
<td>23</td>
<td>23</td>
<td>Medium</td>
<td>0.64</td>
<td>0.65</td>
<td>0.65</td>
<td>Easy</td>
<td>0.60</td>
<td>0.62</td>
<td>0.61</td>
<td>0.59</td>
<td>0.62</td>
<td>0.64</td>
</tr>
<tr>
<td>8</td>
<td>24</td>
<td>24</td>
<td>Medium</td>
<td>0.66</td>
<td>0.67</td>
<td>0.67</td>
<td>Easy</td>
<td>0.62</td>
<td>0.64</td>
<td>0.63</td>
<td>0.61</td>
<td>0.64</td>
<td>0.66</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>25</td>
<td>Medium</td>
<td>0.70</td>
<td>0.71</td>
<td>0.71</td>
<td>Easy</td>
<td>0.66</td>
<td>0.68</td>
<td>0.67</td>
<td>0.66</td>
<td>0.68</td>
<td>0.70</td>
</tr>
<tr>
<td>10</td>
<td>26</td>
<td>26</td>
<td>Medium</td>
<td>0.72</td>
<td>0.73</td>
<td>0.73</td>
<td>Easy</td>
<td>0.68</td>
<td>0.70</td>
<td>0.70</td>
<td>0.69</td>
<td>0.71</td>
<td>0.73</td>
</tr>
<tr>
<td>11</td>
<td>27</td>
<td>27</td>
<td>Medium</td>
<td>0.74</td>
<td>0.75</td>
<td>0.75</td>
<td>Easy</td>
<td>0.70</td>
<td>0.72</td>
<td>0.72</td>
<td>0.70</td>
<td>0.73</td>
<td>0.75</td>
</tr>
<tr>
<td>12</td>
<td>28</td>
<td>28</td>
<td>Medium</td>
<td>0.76</td>
<td>0.77</td>
<td>0.77</td>
<td>Easy</td>
<td>0.72</td>
<td>0.74</td>
<td>0.74</td>
<td>0.72</td>
<td>0.75</td>
<td>0.77</td>
</tr>
<tr>
<td>13</td>
<td>29</td>
<td>29</td>
<td>Medium</td>
<td>0.78</td>
<td>0.79</td>
<td>0.79</td>
<td>Easy</td>
<td>0.74</td>
<td>0.76</td>
<td>0.76</td>
<td>0.74</td>
<td>0.77</td>
<td>0.79</td>
</tr>
<tr>
<td>14</td>
<td>30</td>
<td>30</td>
<td>Medium</td>
<td>0.80</td>
<td>0.81</td>
<td>0.81</td>
<td>Easy</td>
<td>0.76</td>
<td>0.78</td>
<td>0.78</td>
<td>0.76</td>
<td>0.79</td>
<td>0.81</td>
</tr>
<tr>
<td>15</td>
<td>31</td>
<td>31</td>
<td>Medium</td>
<td>0.82</td>
<td>0.83</td>
<td>0.83</td>
<td>Easy</td>
<td>0.78</td>
<td>0.80</td>
<td>0.80</td>
<td>0.78</td>
<td>0.80</td>
<td>0.82</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>32</td>
<td>Medium</td>
<td>0.84</td>
<td>0.85</td>
<td>0.85</td>
<td>Easy</td>
<td>0.80</td>
<td>0.82</td>
<td>0.82</td>
<td>0.80</td>
<td>0.82</td>
<td>0.84</td>
</tr>
<tr>
<td>17</td>
<td>33</td>
<td>33</td>
<td>Medium</td>
<td>0.86</td>
<td>0.87</td>
<td>0.87</td>
<td>Easy</td>
<td>0.82</td>
<td>0.84</td>
<td>0.84</td>
<td>0.82</td>
<td>0.84</td>
<td>0.86</td>
</tr>
</tbody>
</table>

**AGG**: Aggregate Statistics.

### Difficulties
- **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 ($\geq 70\% = Easy$, $\geq 50\%$ and $< 70\% = Medium$, and $< 50\% = Hard$). 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 $\mathcal{R}_S$ 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 $\mathcal{R}_0 \ldots \mathcal{R}_{56}$. 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 $\leq 10$. 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 50 tasks.)
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.