LAGr: Label Aligned Graphs for Better Systematic Generalization in Semantic Parsing

Anonymous ACL submission

Abstract

Semantic parsing is the task of producing structured meaning representations for natural language sentences. Recent research has pointed out that the commonly-used sequence-to-sequence (seq2seq) semantic parsers struggle to generalize systematically, i.e. to handle examples that require recombining known knowledge in novel settings. In this work, we show that better systematic generalization can be achieved by producing the meaning representation directly as a graph and not as a sequence. To this end we propose LAGr (**L**abel **A**ligned **Gr**aphs), a general framework to produce semantic parses by independently predicting node and edge labels for a complete multi-layer input-aligned graph. The strongly-supervised LAGr algorithm requires aligned graphs as inputs, whereas weakly-supervised LAGr infers alignments for originally unaligned target graphs using approximate maximum-a-posteriori inference. Experiments demonstrate that LAGr achieves significant improvements in systematic generalization upon the baseline seq2seq parsers in both strongly- and weakly-supervised settings.

1 Introduction

Recent research has shown that neural models struggle to systematically generalize to examples with unseen combinations of seen rules from the training set (Lake and Baroni, 2018; Finegan-Dollak et al., 2018; Hupkes et al., 2019). Systematic generalization is especially important for the task of semantic parsing, which requires models to translate natural language sentences to structured meaning representations (MRs), such as SPARQL database queries or lambda calculus logical forms. To generalize systematically in this task, the model must be capable of producing MRs for examples that feature new combinations of meaning construction rules, such as the rule that maps a noun like "*hedgehog*" in Figure 1 to its respective predicate $hedgehog(.)$, and the rule that defines which semantic role with respect to the verb (e.g. *agent* or *theme*) the resulting predicate takes. Using synthetic (Bahdanau et al., 2019; Kim and Linzen, 2020a; Keysers et al., 2020) and natural benchmarks (Finegan-Dollak et al., 2018; Shaw et al., 2020), researchers have been studying systematic generalization of existing semantic parsing methods as well as proposing new approaches such as using meta-learning (Conklin et al., 2021), pretrained models (Furrer et al., 2020), or intermediate meaning representations (Herzig et al., 2021).

Training: A ***hedgehog*** ate the cake → $*hedgehog(x_1) \wedge cake(x_4) \wedge eat.agent(x_2, x_1) \wedge eat.theme(x_4)$
Generalization: The baby liked the ***hedgehog*** → $*baby(x_1) \wedge hedgehog(x_4) \wedge like.agent(x_2, x_1) \wedge like.theme(x_4))$

Figure 1: Examples from the training and the generalization sets of the COGS dataset (Kim and Linzen, 2020b). While "*hedgehog*" is only observed in the *agent* role during training, the generalization set features this word in the *theme* role.

The dominant framework in these studies is *sequence-to-sequence* (seq2seq, Sutskever et al., 2014; Bahdanau et al., 2015) learning, whereby the model produces a serialized MR in an autoregressive fashion, by predicting one token at a time, while conditioning on all previously generated tokens. We hypothesize that for semantic parsing constructing the MR by combining independent predictions that are not conditioned on each other can generalize more systematically than seq2seq. For example, consider the sentence "*The dog liked that the hippo danced*". Arguably, the predictions that "*dog*" is the agent of "*like*" and that "*hippo*" is the agent of "*danced*" can be made independently of each other. Our intuition is that a model that predicts such aspects of meaning independently of each other can be better at learning context-insensitive rules because the overall context for each individual prediction is reduced.

Following this intuition, we propose LAGr (**L**abel **A**ligned **Gr**aphs), a framework to produce semantic parses by independently labelling the nodes and edges of a fully-connected multi-layer output graph that is aligned with the input utterance. While the general idea of predicting semantic parses as graphs is not new (Lyu and Titov, 2018), the systematic generalization benefits of doing so have not been investigated prior to this work[1]. Importantly, LAGr retains most of the flexibility that seq2seq models have, without the complexity and rigidity that comes with other alternatives to seq2seq, such as grammar-based methods (Herzig and Berant, 2020).

We first introduce LAGr in the strongly-supervised setting where output graphs are aligned to the input sequences, thus allowing for standard supervised training. For the weakly-supervised case when the alignment is not available, we treat it as a latent variable. We infer the latent alignment with a simple and novel approximate maximum-a-posteriori (MAP) inference approach which involves solving several minimum cost bipartite matching problems with the Hungarian algorithm (Kuhn, 1955a). We then use the resulting aligned graphs to train the model. Our experiments demonstrate that in both strongly- and weakly-supervised settings LAGr significantly improves upon comparable seq2seq semantic parsers on the COGS and CFQ datasets (Kim and Linzen, 2020a; Keysers et al., 2020).

2 Semantic Parsing by Labeling Aligned Graphs

We present LAGr (**L**abel **A**ligned **G**raphs), a framework for constructing meaning representations (MR) directly as graphs (i.e., *MR graphs*). When LAGr is used to output logical forms, the graph nodes can be variables, entities, categories and predicates, and graph edges can be the Neo-Davidsonian style semantic role relations that the nodes appear in, e.g. *"is-agent-of"* or *"is-theme-of"* (Parsons, 1990). While this work focuses on predicting logical forms, LAGr can, in principle, also be used to output other kinds of graphs, such as abstract syntax tree parses of SQL queries. As illustrated in Figure 2, LAGr predicts the output by labeling the nodes and edges of a fully-connected multi-layer output graph that is aligned with the input utterance. We label a multi-layer as opposed to a single-layer graph because some MR graphs have more nodes than the number of input tokens (see Section 4.2 for an example).

[1]A concurrent systematic generalization study by Ontañón et al. (2021) that was put on ArXiv on August 5 features a "sequence tagging" approach that is similar to strongly-supervised LAGr.

Notation and Terminology Formally, let $x = x_1, x_2, ..., x_N$ denote a natural language utterance of N tokens. LAGr produces an MR graph G by labeling the nodes and edges of a complete graph Γ_a with $M = L \cdot N$ nodes that are arranged in L layers. The layers are aligned with the input sequence x in a way that for each input position i there is a unique corresponding output node in each layer. We say that nodes from different layers that are aligned with the position i form a column (an example column in Figure 2b contains the nodes labeled as `actor` and `?x0` for the word *star* at the position $i = 3$).

We write $\Gamma_\text{a} = (z, \xi)$ to indicate that a complete labeled graph Γ_a is characterized by its node labels $z \in V_n^M$ and edge labels $\xi \in V_e^{M \times M}$, where V_n and V_e are node and edge label vocabularies, respectively. Both vocabularies also include additional `null` labels that we use as padding (e.g. grey nodes in Figure 2 are labeled as `null`). To produce the output MR graph G from Γ_a, we remove all `null` nodes and `null` edges. Lastly, we use z_j and ξ_{jk} notations to refer to the labels of node j and of the edge (j, k) where $j = (l-1)N + i$ is a one-dimensional index that corresponds to the i-th node in the l-th layer.

2.1 Labeling Aligned Graphs

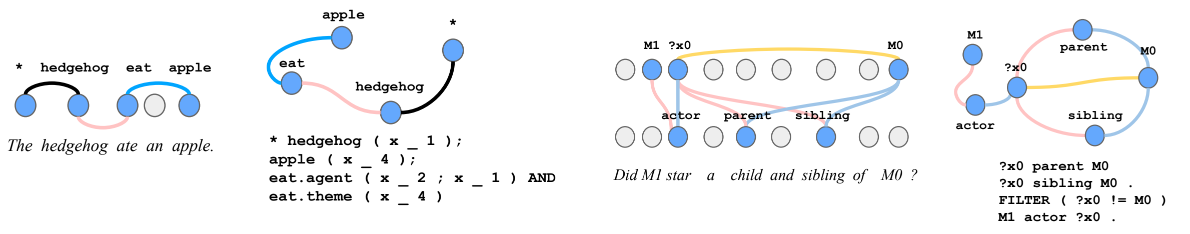
To label the nodes of Γ_a we encode the input utterance x as a matrix of N d-dimensional vectors $H = f_{enc}(x) \in \mathbb{R}^{N \times d}$, where f_{enc} can be an arbitrary encoder model such as LSTM (Hochreiter and Schmidhuber, 1997) or a Transformer (Vaswani et al., 2017). LAGr then defines a factorized distribution $p(z|x)$ over the node labels z as follows:

$$O = \big\Vert_{l=1}^{L} HW^l, \quad (1)$$

$$\pi = \texttt{softmax}(O), \quad (2)$$

$$p(z|x) = \prod_{j=1}^{M} p(z_j|x) = \pi_{j,z_j}, \quad (3)$$

where $O \in \mathbb{R}^{M \times |V_n|}$ contains logits for $M = N \times L$ nodes from all the L graph layers, $||$ denotes the concatenation operation along the node axis, W_l denotes the weight matrix for layer l. Here and



```
* hedgehog ( x _ 1 ) ;
apple ( x _ 4 ) ;
eat.agent ( x _ 2 ; x _ 1 ) AND
eat.theme ( x _ 4 )
```

```
?x0 parent M0
?x0 sibling M0 .
FILTER ( ?x0 != M0 )
M1 actor ?x0 .
```

(a) COGS

(b) CFQ

Figure 2: Aligned and unaligned graphs for COGS (a) and CFQ (b). For COGS, pink, blue and black denote `agent`, `theme` and **`article`** edges, respectively. For CFQ, yellow, pink and blue mark `FILTER`, `agent`, `theme` edges. Grey nodes mark null nodes, and * denotes the definite article. The aligned graph for CFQ is provided for illustration purposes, and was not used for training. For the learned CFQ aligned graphs see Section 4.

in following equations $\mathtt{softmax}(.)$ is applied to the last dimension of the input tensor and every multiplication by a weight matrix is followed by the addition of a bias vector which we omit to enhance clarity. Our edge labelling computation is reminiscent of the multi-head self-attention by Vaswani et al. (2017), with the key difference that softmax is applied across the edge labels and not across positions:

$$H_q^\alpha = \big\Vert_{l=1}^{L} HU^{\alpha,l}, \quad H_k^\alpha = \big\Vert_{l=1}^{L} HV^{\alpha,l},$$

$$\rho = \mathtt{softmax}\left[\underset{\alpha\in V_e}{\mathtt{stack}}\left[H_q^\alpha H_k^{\alpha T}\right]\right],$$

where H_q^α and H_k^α contain concatenated key and query vectors for the label $\alpha \in V_e$ across all L graph layers, $U^{\alpha,l}, V^{\alpha,l} \in \mathbb{R}^{\frac{d}{|V_e|},\frac{d}{|V_e|}}$ are the weights for the edge label α, and the $\mathtt{stack}$ operator stacks the matrices into a 3D tensor to which softmax is subsequently applied. Similarly to $p(z|x)$, we obtain $p(\xi|x)$ as follows:

$$p(\xi|x) = \prod_{j=1}^{M}\prod_{k=1}^{M} p(\xi_{jk}|x) = \prod_{j=1}^{M}\prod_{k=1}^{M} \rho_{jk\xi_{jk}}. \quad (4)$$

The factorized nature of Equations 3 and 4 makes the argmax inference $\hat{z}, \hat{\xi} = \arg\max p(z,\xi|x)$ trivial to perform. When the groundtruth aligned graph $\Gamma_a^* = (z^*, \xi^*)$ for the MR graph G is available, LAGr can be trained by directly optimizing $\log p(z = z^*, \xi = \xi^*|x)$. We refer to this training setting as *strongly-supervised LAGr*.

2.2 Weakly-supervised LAGr

In many practical settings, the alignment between the MR graph G and the question x is unavailable, making the aligned graph Γ_a unknown. To address this common scenario, we propose a *weakly-supervised LAGr* algorithm based on a latent alignment model. Similarly to the strongly-supervised case, we assume that the MR graph can be represented as a labeled complete, multi-layer graph $\Gamma_{na} = (s \in V_n^M, e \in V_e^{M\times M})$, with the difference that in this case the alignment between x and Γ_{na} is not known. We assume a generative process whereby Γ_{na} is obtained by permuting the columns of the latent aligned graph Γ_a with a random permutation a, where a_j is the number of the column in Γ_a that becomes the j-th column in Γ_{na}. For the rest of this section we focus on the single layer ($L = 1$) case to simplify the formulas. For this case our probabilistic model defines the following distribution over $\Gamma_{na} = (s, e)$:

$$\begin{aligned} p(e,s|x) &= \sum_a \sum_z \sum_\xi p(e,s,a,z,\xi|x) \\ &= \sum_a p(a) \prod_j p(z_{a_j} = s_j|x) \qquad (5) \\ &\quad \prod_j \prod_k p(\xi_{a_j a_k} = e_{jk}|x), \end{aligned}$$

where $p(a) = 1/N!$. Computing $p(e,s|x)$ exactly is intractable. For this reason, we train LAGr by using an approximation of $p(e,s|x)$ in which instead of summing over all possible aligments a, we only consider the maximum-a-posteriori (MAP) alignment $\hat{a} = \arg\max_a p(a|e,s,x)$. This approach is sometimes called the hard Expectation-Maximization algorithm in the literature on probabilistic models (Svensén and Bishop, 2007). The training objective thus becomes

$$\begin{aligned} &p(e,s|\hat{a},x) = \\ &\prod_j p(z_{\hat{a}_j} = s_j|x) \prod_j \prod_k p(\xi_{\hat{a}_j,\hat{a}_k} = e_{jk}|x). \end{aligned}$$

To infer the MAP alignment $\hat{a}$, we need to solve the following inference problem:

$$\begin{aligned}\hat{a} &= \arg\max_a p(a|e,s,x) \\ &= \arg\max_a \log p(s|a,x) + \log p(e|a,x) \\ &= \arg\max_a \Big[\sum_j \log p(z_{a_j} = s_j|x) \\ &\quad + \sum_j \sum_k \log p(\xi_{a_j,a_k} = e_{j,k}|x) \Big]\end{aligned} \tag{6}$$

We are not aware of an exact algorithm for solving the above optimization problem, however if the edge log-likelihood term $\log p(e|a,x)$ is dropped in the equations above, maximizing the node label probability $p(s|a,x)$ is equivalent to a standard minimum cost bipartite matching problem. This optimization problem can be solved by a polynomial-time Hungarian algorithm (Kuhn, 1955b). We can thus use an approximate MAP alignment $\hat{a}^1 = \arg\max_a \sum_j \log p(z_{a_j} = s_j|x)$. While dropping $p(e|a,x)$ from Equation 6 is a drastic simplification, in situations where node labels s are unique and the model is sufficiently trained to output sharp probabilities $p(z_j|x)$ we expect $\hat{a}_1$ to often match $\hat{a}$. To further improve the MAP alignment approximation and alleviate the reliance on the node label uniqueness, we generate a shortlist of K candidate alignments by solving K noisy matching problems of the form $\arg\max_a \sum_j \log p(z_{a_j} = s_j|x) + \epsilon_{ja_j}$, where $\epsilon_{ja_j} \sim N(0,\sigma)$. We then select the alignment candidate a that yields the highest full log-likelihood $\log p(s|a,x) + \log p(e|a,x)$.

We refer the reader to Algorithm 1 for a detailed presentation of weakly-supervised LAGr.

Algorithm 1: Training LAGr with weak supervision

```
Init: Let $K$ be the number of alignment candidates, $T$ be the number of
      training steps, and $\theta_t$ be the model parameters after $t$ steps.
1  for $t$=1, ..., $T$ do
2      sample example $(x, e, s)$
3      for $\kappa$=1, ..., $K$ do
4          $\epsilon_{ji} \sim N(0,\sigma)$
5          $cost_{ji} = -\log p(z_i = s_j|x) + \epsilon_{ji}$
6          $a^\kappa$ = MinCostBipartiteMatching($cost$)
7          $J^\kappa = \sum_j \log p(z_{a^\kappa_j} = s_j|x)$
8                 $+ \sum_j \sum_k \log p(\xi_{a^\kappa_j a^\kappa_k} = e_{jk}|x)$
9      $\hat{\kappa} = \arg\max_\kappa J^\kappa$
10     $\theta_{t+1} \leftarrow$ Optimizer($\theta_t, \nabla_\theta - J^{\hat{\kappa}}$)
11 return $\theta_{T+1}$
```

3 Related Work

The LAGr approach is heavily inspired by graph-based dependency parsing algorithms (Mcdonald, 2006). In neural graph-based dependency parsers (Kiperwasser and Goldberg, 2016; Dozat and Manning, 2017) the model is trained to predict the existence and the label of each of the possible edges between the input words. The Abstract Meaning Representation (AMR) parser by Lyu and Titov (2018) brings similar methodology to the realm of semantic parsing, although they do not consider the systematic generalization implications of using a graph-based parser instead of a seq2seq one. Lyu and Titov (2018) only output single layer graphs which requires aggresive graph compression; in LAGr we allow the model to output a multiple layer graph instead. Lastly, the amortized Gumbel-Sinkhorn alignment inference used by Lyu and Titov (2018) is much more complex than the Hungarian-algorithm-based approximate MAP inference that we employ here. Another important inspiration for LAGr is the UDepLambda method (Reddy et al., 2016) that converts dependency parses into graph-like logical forms. LAGr can be seen as an algorithm that produces UDepLambda graphs directly with the neural model, side-stepping the intermediate dependency parsing step.

Another alternative to seq2seq semantic parsers are span-based parsers that predict span-level actions for building MR expressions from sub-expressions. (Herzig and Berant, 2020; Pasupat et al., 2019). A prerequisite for using a span-based parser is an MR that can be viewed as a recursive composition of MRs for subspans. While this strong compositionality assumption holds for the logical forms used in earlier semantic parsing research (e.g. Zettlemoyer and Collins (2005)), an intermediate MR would be required to produce other meaning representations, such as e.g. SPARQL or SQL queries, with a span-based parser. The designer for an intermediate MR for a span-based parser must think about MRs for spans and how they should be composed. This can sometimes lead to non-trivial corner cases, such as e.g. ternary grammar rules in Herzig and Berant (2020). On the contrary, a graph-based parser can in principle produce any graph, although in practice in our experiments we compress the raw graphs slightly to make the learning problem easier.

Other related semantic parsing approaches include the semantic labeling method by Zheng and Lapata (2020) and the structured reordering approach by Wang et al. (2021). Zheng and Lapata (2020) show that labelling the input sequence prior to feeding it to the seq2seq semantic parser improves systematic generalization. Compared to that study, our work goes one step further by adding edge labeling, which allows us to let go of the seq2seq model entirely. Wang et al. (2021) model semantic parsing as structured permutation of the input sequence followed by monotonic segment-level transduction. This approach achieves impressive results, but is considerably more complex than LAGr. Finally, Guo et al. (2020) achieve a very high performance on CFQ by combining the sketch prediction approach (Dong and Lapata, 2018) with an algorithm that outputs the MR as a directed acyclic graph (DAG). Unlike LAGr, their algorithm produces the DAG in a sequential left-to-right fashion. Notably, the non-hierachical version of this algorithm without sketch prediction performs poorly.

Concurrently with this work, Ontañón et al. (2021) show that semantic parsing by sequence tagging improves systematic generalization. Their sequence tags are similar to 1-layer aligned graphs that we predict here. Ontañón et al. (2021) do not discuss how to infer sequence tags from logical forms when the former are not available.

4 Experiments

We demonstrate the effectiveness of LAGr on two systematic generalization benchmarks for semantic parsing: COGS (Kim and Linzen, 2020a) and Compositional Freebase Questions (CFQ, Keysers et al. (2020)).

4.1 COGS

Dataset COGS (Kim and Linzen, 2020a) is a semantic parsing benchmark that requires models to translate English sentences to Neo-Davidsonian lambda calculus logical forms. As shown in Figure 1, the out-of-distribution generalization set of COGS features novel combinations of words and syntactic structures from the training dataset (more examples available in Appendix A.4).

Graph Construction In order to study LAGr on COGS, we first convert the logical forms to UDepLambda-style (Reddy et al., 2016) MR graphs. Specifically, we construct the graph nodes using the one- and two-place predicates and definite articles (e.g. `hedgehog`, `apple`, `eat` and the * nodes in Figure 2a). We do not create dedicated nodes for variables, as every variable in COGS is either an argument to a unique one-place predicate (e.g. x_1 is for `hedgehog`(x_1)), or the first argument to a unique two-place predicate (e.g. x_2 for `eat` in `eat.agent`(x_2, x_1)). Instead, we let the respective predicate node represent the variable.

The labeled edges for our graphs are defined by the Neo-Davidsonian role predicates of the logical forms (such as `agent`, `theme`, `recipient`, `ccomp`, `nmod.on`, `nmod.in`, `xcomp`, `nmod.beside`). For example, the conjunct `eat.agent`(x_2, x_1) results in an `agent` edge between the `eat` and `hedgehog` nodes. We also add special `article` edges to connect definite article nodes (denoted by the * label) to their respective nouns (`hedgehog` in Figure 2a). We take advantage of the correspondence between variable names and input positions (x_i corresponds to the i-th token) to construct single-layer ($L = 1$) aligned graphs Γ_a for COGS that are suitable for strongly-supervised LAGr, as described in Section 2.1. The node and edge vocabularies for the aligned graphs contain 645 and 10 labels respectively, each including a `null` label.

Training Details Hyperparameter tuning on COGS is challenging since the the performance on the in-distribution development set always saturates to near 100%. We adopt the hyperparameter tuning procedure discussed in Conklin et al. (2021) to find the best configuration for our baselines and strongly-supervised LAGr models. Specifically, we create a "Gen Dev" dataset by sampling 1000 random examples from the generalization set and use them to find the best hyperparameter configuration. We find that our Transformer-based seq2seq and LAGr models perform better when embeddings are initialized following He et al. (2015) and when positional embeddings are scaled down by $\frac{1}{\sqrt{dim}}$. The latter technique has been recently proposed by Csordás et al. (2021) under the PED (Positional Embedding Downscaling) name. We report the exact match accuracy, i.e., the percentage of examples for which the predicted graphs after serialization yielded the same logical form, as well as the standard deviation over 10 random seeds. We tune the hyperparameters for strongly-supervised LAGr first;

	Exact match accuracy		
	train	test	gen
LSTM+Attn ◇	-	99.	16. (±8.)
Transformer ◇	-	96.	35. (±6.)
LSTM+Attn ♡	-	-	51. (±5.)
Transformer ♠	-	-	**81.** (±1.)
LSTM + Lex: Simple ♡	-	-	**82.** (±1.)
LSTM + Lex: PMI ♡	-	-	**82.** (±0.)
LSTM + Lex: IBMM2 ♡	-	-	**82.** (±0.)
LSTM+Attn (ours)	100 (±0.0)	99.6 (±0.2)	26.1 (±6.8)
LSTM$_{sh}$ strongly-supervised LAGr	100 (±0.0)	99.9 (±0.1)	39.0 (±9.1)
LSTM$_{sep}$ strongly-supervised LAGr	100 (±0.0)	100 (±0.0)	71.4 (±2.9)
Transformer (ours)	100 (±0.0)	99.2 (±0.1)	70.3 (±5.6)
Transformer$_{sh}$ strongly-supervised LAGr	100 (±0.0)	99.9 (±0.1)	78.1 (±2.4)
Transformer$_{sep}$ strongly-supervised LAGr	100 (±0.0)	99.9 (±0.2)	**82.2** (±2.5)
Transformer$_{sh}$ weakly-supervised LAGr	99.4 (± 0.3)	99.3 (± 0.5)	78.5 (± 3.4)
Transformer$_{sep}$ weakly-supervised LAGr	99.3 (± 0.4)	99.0 (±0.7)	**80.8** (± 2.3)

Table 1: Average exact match accuracy and standard deviation on COGS. **Bottom**: reproduced seq2seq baselines and LAGr over 10 runs. **Middle:** Seq2seq baselines including the original results by Kim and Linzen (2020a) ◇, best Transformer baseline by Csordás et al. (2021) ♣, and the best LSTM baseline by Akyürek and Andreas (2021) ♡. We also show a lexicon-based approach by Akyürek and Andreas (2021).

for weakly-supervised LAGr we reuse the found configuration and only tune the inference hyperparameters, i.e. the number of candidates K and the noise level σ. Weakly-supervised LAGr often does not converge on the training. To remedy this, we tune K and σ to make convergence more frequent. Setting $K = 5$ and $\sigma = 15$ allows us to achieve a convergence rate of above 45%. We restart the experiments that do not achieve at least 98% performance on the training set. For more details on our hyperparameter search, and best configurations, we refer the reader to Appendix A.1.

Baselines We compare LAGr to LSTM- and Transformer- based seq2seq semantic parsers that produce logical forms as sequences of tokens. In addition to training our own seq2seq baselines, we also include baseline results from the original COGS paper by Kim and Linzen (2020a) and from follow-up works by Akyürek and Andreas (2021), and Csordás et al. (2021). We also compare LAGr to a lexicon-based seq2seq model "LSTM+Lex" by Akyürek and Andreas (2021) that leverages the copy mechanism in the seq2seq decoder to perform a lexical lookup to generate the output token.

Results Table 1 shows that our best Transformers trained with LAGr outperform the original (35%) and our reproduced (70.3%) seq2seq Transformer baselines, obtaining 82.2% (±2.5) and 80.8% (±2.3) exact match accuracy in the strongly- and weakly-supervised settings, respectively. Only the very recent work by Csordás et al. (2021) reports seq2seq results that are comparable to LAGr performance, however when applied to our codebase their proposed PED technique only brought a modest improvement for both our seq2seq and LAGr models.

We experiments two variations of LAGr: using shared and separate encoders for node and edge predictions — reflected in Table 1 by the subindex "_*sh*" versus "_*sep*" in the model names respectively. For both strongly- and weakly-supervised LAGr, using separate encoder models achieves the best results. While this setting significantly improves the performance of LAGr in all cases, for the strongly-supervised LSTM-based LAGr models, separating encoders seems to be crucial (71.4% vs 39.0%).

Finally, LAGr is able to match the performance of the LSTM+Lex approach by Akyürek and Andreas (2021) without relying on the use of lexicons — a result we further discuss in Section 5.

4.2 CFQ

Dataset CFQ (Keysers et al., 2020) is a benchmark for systematic generalization in semantic parsing that requires models to translate English sentences to SPARQL database queries. We use CFQ's *Maximum Compound Divergence* (MCD) splits, which were generated by making the distribution of compositional structures in the train and test sets as divergent as possible.

SPARQL queries contain two components: a `SELECT` and a `WHERE` clause. The `SELECT` clause is either of the form `SELECT count(*)` for yes/no questions or `SELECT DISTINCT ?x0` for wh- questions (those starting with "which", "what", "who", etc.). The `WHERE` clause can contain constrains of

	Graph Accuracy					
	Random		Mean MCD	MCD1	MCD2	MCD3
	train	test	test	test	test	test
HPD ♠	-	-	67.3 (∓4.1)	72.0 (∓7.5)	66.1 (∓6.4)	63.9 (∓5.7)
HPD w/o Hierarchical Mechanism ♠	-	-	-	21.3	6.4	10.1
T5-small + IR ◇	-	-	47.9	-	-	-
LSTM + Attn ♡	-	97.4 (∓0.3)	14.9 (∓1.1)	28.9 (∓1.8)	5.0 (∓0.8)	10.8 (∓0.6)
Transformer ♡	-	98.5 (∓0.2)	17.9 (∓0.9)	34.9 (∓1.1)	8.2 (∓0.3)	10.6 (∓1.1)
Universal Transformer ♡	-	98.0 (∓0.3)	18.9 (∓1.4)	37.4 (∓2.2)	8.1 (∓1.6)	11.3 (∓0.3)
Evol. Transformer ♣	-	-	20.8 (∓0.7)	42.4 (∓1.0)	9.3 (∓0.8)	10.8 (∓0.2)
LSTM + Simplified SPARQL ♠	-	-	26.1	42.2	14.5	**21.5**
Transformer + Simplified SPARQL ♠	-	-	31.4	53.0	19.5	**21.6**
T5-small from scratch ◇	-	-	20.8	-	-	-
T5-small from scratch + IR ◇	-	-	22.6	-	-	-
Transformer_{sh} weakly sup. LAGr, $K = 1$	99.6 (∓0.5)	98.5 (∓0.6)	29.2 (∓15.9)	50.9 (∓4.9)	18.3 (∓1.6)	18.4 (∓1.2)
Transformer_{sh} weakly sup. LAGr, $K = 5, \sigma = 10$	**100** (∓0.1)	**99.7** (∓0.2)	**34.9** (∓16.9)	**57.9**(∓3.21)	**26.0** (∓3.0)	**20.9** (∓1.2)

Table 2: Average graph accuracy and standard deviation over 10 runs of weakly-supervised LAGr on CFQ (**bottom**). **Middle:** results by several seq2seq baselines from prior work (Keysers et al. (2020)♡, Furrer et al. (2020) ♣). **Top:** results not directly comparable to LAGr: Hierarchical Poset Decoding (Guo et al., 2020) ♠, and pretrained T5-small seq2seq model with intermediate representations (IR) (Herzig et al., 2021) ◇. Approaches other than LAGr report the average exact match accuracy with 95% confidence intervals.

three kinds: filter constraints ensuring two variables or entities are distinct (e.g. `FILTER ?x0 != M0`), two-place predicates expressing a relation between two entities (e.g. `?x0 parent ?x1`), and one-place predicates expressing if an entity belongs to a category (e.g. `?x0 a ns:film.actor`)

Graph Construction Before constructing the graphs, similarly to prior work (Furrer et al., 2020; Guo et al., 2020), we compress the SPARQL queries by merging some triples in the `WHERE` clauses. As an example, consider the question "*Were M2 and M3 directed by a screenwriter that executive produced M1?*", where the original MR contains both `[M2 directed_by ?x0, M3 directed_by ?x0]` conjuncts. To make it easier to align SPARQL queries to the input question, we merge triples by concatenating their subjects and objects, e.g. yielding `[[M2, M3] directed_by ?x0]` for the above example. With this compression, the SPARQL queries can now contain an arbitrary number of entities in the triples.

To convert the compressed SPARQL queries to graphs we first remove the `SELECT` clauses. To preserve the question type information, for wh-questions we replace the `?x0` variable in the `WHERE` clause with a special `select_?x0` variable. As the example in Figure 2b shows, we define the graph nodes by taking the entities (including variables, e.g. `?x0, M1`) and all predicates (`parent, sibling, actor`) from the triples. For one-place predicates, we connect the entity nodes to the predicate node with an `agent` edge label. For triples with two-place predicates, we connect the predicate to the left-hand side and right-hand side entities with the `agent` and `theme` edge respectively. We add a `FILTER` edge between the variables or entities that participate in a filter constraint. The resulting node and the edge vocabularies contain 84 and 4 labels respectively, each also including a `null` label.

Training Details Unlike COGS, for CFQ we need to accommodate the larger MR graphs by using L=2 graph layers. This is because CFQ contains examples such as "*Who married M1's female German executive producer?*" that contains 8 tokens, but induces the following 10 nodes: `?x1, executive_produced, M1, gender, ns:m.02zsn, nationality, ns:m.0345h, select_?x0, spouses, person`.

In all our CFQ experiments we use a shared Transformer encoder for both node and edge prediction. To assess performance, we use exact graph accuracy, which we define as the percentage of examples where the predicted and true graphs are isomorphic. The predicted graphs contain enough information to exactly reconstruct the SPARQL query, hence our exact graph accuracy can be compared to the exact match accuracy from the prior work. For hyper-parameter tuning, we follow Keysers et al. (2020) and use CFQ's in-distribution *random* split to find the best model configuration. We do this by first fixing the number of candidate alignments at $K = 1$ to search for the best hyperparameters, then fixing the best configuration and varying K and σ. For the best found configuration of $K = 5$ and $\sigma = 10$

		Graph Accuracy	
K	σ	train	test
1	0.0	99.79 (∓0.4)	98.75 (∓0.5)
5	0.01	99.92 (∓0.1)	99.01 (∓0.2)
	0.1	99.88 (∓0.1)	99.10 (∓0.3)
	1.0	99.85 (∓0.2)	99.10 (∓0.3)
	10.0	**99.97** (∓0.1)	**99.69** (∓0.1)
	15.0	83.78 (∓1.6)	83.73 (∓1.7)
	20.0	2.18 (∓0.17)	2.28 (∓0.19)
10	0.01	99.77 (∓0.3)	98.85 (∓0.6)
	0.1	99.92 (∓0.1)	99.10 (∓0.2)
	1.0	99.70 (∓0.3)	98.68 (∓0.7)
	10.0	99.96 (∓0.1)	99.58 (∓0.2)
	15.0	99.77 (∓0.4)	99.42 (∓0.5)
	20.0	69.69 (∓3.9)	68.91 (∓4.0)

Table 3: The effect of the number of alignment candidates K and noise level σ on the performance of weakly-supervised LAGr using CFQ's random split. We report the average graph accuracy and the standard deviation over 5 runs. We show the best configuration in bold.

we report the average graph accuracy and standard deviation for 10 runs of weakly-supervised LAGr on the out-of-distribution splits MCD1, MCD2, and MCD3 as well as on the random split. In contrast to COGS, the PED technique from Csordás et al. (2021) for training Transformers leads to worse results on the random split. For this reason, we use the standard OpenNMT-py Transformer implementation by Klein et al. (2017). Lastly, similarly to COGS, we discard runs where weakly-supervised LAGr does not reach at least 98% graph accuracy on the training set, which for CFQ is rare (less than 5% of all runs). For further details on our CFQ experiments we refer the reader to Appendix A.2.

Results We compare LAGr to seq2seq semantic parsing results reported in prior work (Keysers et al., 2020; Furrer et al., 2020), as well as results obtained with compressed SPARQL queries (Guo et al., 2020; Herzig et al., 2021). As shown in Table 2, weakly-supervised LAGr outperforms all these baselines on the MCD1 and MCD2 splits. On MCD3, we match the compressed SPARQL results reported by Guo et al. (2020). For reference, Table 2 also includes the state-of-the-art Hierarchical Poset Decoding (HPD, Guo et al., 2020) method (see Section 3), which arguably is not a fair baseline to LAGr because of its use of sketch prediction and lexicons. Notably, when these techniques are not used, LAGr performs much better than their base poset decoding algorithm.

Table 3 zooms in on the impact of the hyperparameters of weakly-supervised LAGr, namely, the number of alignment candidates K and the noise level σ. One can see that choosing the best alignment out of $K > 1$ candidates is indeed helpful, and that noise of high magnitude ($\sigma = 10$) brings the best improvement on the random split. These improvements also translate into systematic generalization gains, as shown when comparing the MCD results for $K = 1$ versus $K = 5$ in Table 2.

The positive effect of a larger K is in line with our expectation since 3.7 - 5.7% of examples in each CFQ split have at least two predicates with identical node labels, which can make it hard to align the MR graph to the input by looking at node labels only. Interestingly, in contrast to our intuition, when using ten candidate alignments, the random split test performance is slightly worse than when using five. We show examples of the node labels that weakly-supervised LAGr predicts in the learned aligned CFQ graphs as well as the corresponding SPARQL queries in Figure 3 (Appendix A.3).

5 Discussion & Future Work

In this work we have shown that performing semantic parsing by labeling aligned graphs brings significant gains in systematic generalization. In our COGS and CFQ experiments, LAGr significantly improves upon sequence-to-sequence baselines in both strongly and weakly-supervised settings. Specifically, on COGS, LAGr outperforms our carefully-tuned seq2seq baselines and performs similarly to LSTMs that leverage lexicons. The use of lexicons can be integrated into LAGr although we do not expect this to improve LAGr performance on COGS, as our best performing LAGr model already predicts node labels almost perfectly. Lexicons also bring their own challenges of dealing with context-dependency and ambiguity, hence it is notable that LAGr matches the performance of a lexicon-equipped model while making less assumptions about the nature of the input-to-output mapping. On CFQ, LAGr outperforms all seq2seq baselines on 2 out of 3 MCD splits. Based on our error analysis (see Appendix A.3), we believe that a modification of LAGr that conditions edge predictions on node labels could bring further improvements. Importantly, this modification would be compatible with our current alignment inference algorithm. Another obvious direction to improve LAGr performance is by using a pretrained encoder. Lastly, while the current alignment inference algorithm is effective, applying more advanced discrete optimization or amortized inference methods could be an interesting direction for future work.

References

Ekin Akyürek and Jacob Andreas. 2021. Lexicon Learning for Few-Shot Neural Sequence Modeling. *arXiv:2106.03993 [cs]*. ArXiv: 2106.03993.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations, ICLR 2015*.

Dzmitry Bahdanau, Harm de Vries, Timothy J. O'Donnell, Shikhar Murty, Philippe Beaudoin, Yoshua Bengio, and Aaron Courville. 2019. CLOSURE: Assessing Systematic Generalization of CLEVR Models. *arXiv:1912.05783 [cs]*. ArXiv: 1912.05783.

Henry Conklin, Bailin Wang, Kenny Smith, and Ivan Titov. 2021. Meta-Learning to Compositionally Generalize. *arXiv:2106.04252 [cs]*. ArXiv: 2106.04252.

Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. 2021. The devil is in the detail: Simple tricks improve systematic generalization of transformers. *arXiv preprint arXiv:2108.12284*.

Li Dong and Mirella Lapata. 2018. Coarse-to-Fine Decoding for Neural Semantic Parsing. *arXiv:1805.04793 [cs]*. ArXiv: 1805.04793.

Timothy Dozat and Christopher D. Manning. 2017. Deep Biaffine Attention for Neural Dependency Parsing. *arXiv:1611.01734 [cs]*. ArXiv: 1611.01734.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving Text-to-SQL Evaluation Methodology. *arXiv:1806.09029 [cs]*. ArXiv: 1806.09029.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. 2020. Compositional Generalization in Semantic Parsing: Pre-training vs. Specialized Architectures. *arXiv:2007.08970 [cs]*. ArXiv: 2007.08970.

Yinuo Guo, Zeqi Lin, Jian-Guang Lou, and Dongmei Zhang. 2020. Hierarchical Poset Decoding for Compositional Generalization in Language. *arXiv:2010.07792 [cs]*. ArXiv: 2010.07792.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Jonathan Herzig and Jonathan Berant. 2020. Span-based Semantic Parsing for Compositional Generalization. *arXiv:2009.06040 [cs]*. ArXiv: 2009.06040.

Jonathan Herzig, Peter Shaw, Ming-Wei Chang, Kelvin Guu, Panupong Pasupat, and Yuan Zhang. 2021. Unlocking Compositional Generalization in Pre-trained Models Using Intermediate Representations. *arXiv:2104.07478 [cs]*. ArXiv: 2104.07478.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2019. The compositionality of neural networks: integrating symbolism and connectionism. *arXiv:1908.08351 [cs, stat]*. ArXiv: 1908.08351.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring Compositional Generalization: A Comprehensive Method on Realistic Data. In *International Conference on Learning Representations*. ArXiv: 1912.09713.

Najoung Kim and Tal Linzen. 2020a. COGS: A Compositional Generalization Challenge Based on Semantic Interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105, Online. Association for Computational Linguistics.

Najoung Kim and Tal Linzen. 2020b. Cogs: A compositional generalization challenge based on semantic interpretation. pages 9087–9105.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada. Association for Computational Linguistics.

Harold W. Kuhn. 1955a. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97. Publisher: Wiley Online Library.

Harold W Kuhn. 1955b. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Brenden M. Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 36th International Conference on Machine Learning*. ArXiv: 1711.00350.

Chunchuan Lyu and Ivan Titov. 2018. AMR Parsing as Graph Prediction with Latent Alignment. *arXiv:1805.05286 [cs]*. ArXiv: 1805.05286.

Ryan Mcdonald. 2006. *Discriminative learning and spanning tree algorithms for dependency parsing*. phd, University of Pennsylvania, USA. AAI3225503 ISBN-13: 9780542799785.

Santiago Ontañón, Joshua Ainslie, Vaclav Cvicek, and Zachary Fisher. 2021. Making Transformers Solve Compositional Tasks. *arXiv:2108.04378 [cs]*. ArXiv: 2108.04378.

Terence Parsons. 1990. Events in the semantics of english: A study in subatomic semantics.

Panupong Pasupat, Sonal Gupta, Karishma Mandyam, Rushin Shah, Mike Lewis, and Luke Zettlemoyer. 2019. Span-based Hierarchical Semantic Parsing for Task-Oriented Dialog. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1520–1526, Hong Kong, China. Association for Computational Linguistics.

Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. 2016. Transforming Dependency Structures to Logical Forms for Semantic Parsing. *Transactions of the Association for Computational Linguistics*, 4:127–140.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2020. Compositional Generalization and Natural Language Variation: Can a Semantic Parsing Approach Handle Both? *arXiv:2010.12725 [cs]*. ArXiv: 2010.12725.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112.

Markus Svensén and Christopher M Bishop. 2007. Pattern recognition and machine learning.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*. ArXiv: 1706.03762.

Bailin Wang, Mirella Lapata, and Ivan Titov. 2021. Structured Reordering for Modeling Latent Alignments in Sequence Transduction. *arXiv:2106.03257 [cs]*. ArXiv: 2106.03257.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, pages 658–666.

Hao Zheng and Mirella Lapata. 2020. Compositional generalization via semantic tagging. *arXiv preprint arXiv:2010.11818*.

A Appendix

A.1 COGS Hyperparameter Tuning

COGS does not include an out-of-distribution development set, which makes it challenging to find the best model configuration. To overcome this problem, we followed the same hyperparameter tuning procedure for our baselines and our strongly-supervised LAGr models as proposed by Conklin et al. (2021). We sampled 1000 examples from the generalization set as a "Gen Dev" set which was used to pick the best hyperparameter configuration. We tested 0.001, 0.004, 0.0001 and 0.0004 for learning rates, 64, 128 and 256 for batch sizes, and 0.1 versus 0.4 for dropout. We tested an embedding size of 256 versus 512. Furthermore, for the Transformer baselines and for LAGr with a Transformer encoder, we also tested 2 versus 4 layers, and 4 versus 8 attention heads.

Each configuration was evaluated on 5 seeds. Once the best configuration was found, we retrained all models on 10 new seeds. We trained all models for 70,000 steps validating at every 5000 steps, with no early stopping. We used the same procedure for tuning the original sequence-to-sequence baselines, except we only trained models for 50,000 steps. The best configurations for COGS are shown in Table 6.

For weakly-supervised LAGr, we used the best configuration we found for strongly-supervised LAGr. We then investigated different values for K, the number of candidate alignments, with 1, 5 versus 10, and for the noise levels σ of 0, 0.01, 0.1, 1, 10, 15 and 20. In addition, we also implemented a random restart procedure to restart runs with a new random seed if they were not able to reach at least 98% of training accuracy. We found that only when we used $K = 5$ with a sufficiently high noise level such as $\sigma = 15$, we were able to get 46-47% of the runs to converge. This was different from our CFQ experiments, where 97% of runs converged when appropriate noise levels were chosen (i.e., $\sigma < 15$).

A.2 CFQ Hyperparameter Tuning

We performed hyperparameter tuning on CFQ's random split, and chose the best configuration based on the development exact graph accuracy. For LAGr with both shared and separate Transformer encoders, we tested learning rates of 0.0001, 0.0004, 0.0006, 0.0008 and 0.001, with a linear warmup of 0, 1000 versus 5000 steps, with dropout of 0.1 and 0.4, batch sizes of 64, 128 and 256, and 2 versus

4 Transformer layers. For LAGr with a separate LSTM encoder, we tested learning rates of 64, 128, and 256, with a linear warmup of 1000 versus 5000 steps, a dropout of 0.1 and 0.4, and embedding size of 256 versus 512. In addition, we also tested the PED modifications proposed by Csordás et al. (2021) to improve the performance of Transformer-based models. However, we found that this did not improve our models, so we used the standard Transformer implementation from OpenNMT-py (Klein et al., 2017). Lastly, similarly to COGS, we filtered out runs that diverged in terms of their training graph accuracy. While for COGS weakly-supervised LAGr is more sensitive to varying K and σ, in CFQ, we obtained 97% convergence from all our runs in Table 3. We report the best configuration used for CFQ in Table 7.

A.3 Error analysis

Table 4 shows some commonly encountered errors on COGS with strongly-supervised LAGr. In all examples, the model predicted the correct set of nodes. However, even when all nodes are correctly predicted, some may not show up in the final logical form, if it has no connecting edges to other nodes (see the "dog" node in example 4.).

Figure 3 shows the predicted nodes of aligned graphs and resulting queries produced by the best weakly-supervised LAGr model on CFQ. The top two rows show common errors where some edge labels do not get predicted, and where some nodes are missing due to the model not having predicted any connecting edges for the nodes, thus omitting the nodes from the final output graph. The bottom two rows show the inferred aligned graphs for examples that result in the correct output graph.

A.4 Further COGS examples

Table 5 shows further examples from COGS's generalization set with various cases for challenging models' ability to test systematic generalization.

Example 1: wrong edge label, between right nodes

In	A cockroach sent Sophia the sandwich beside the yacht .
Out	* sandwich (x _ 5) ; * yacht (x _ 8) ; cockroach (x _ 1) AND send . **theme** (x _ 2 , x _ 1) AND send . recipient (x _ 2 , Sophia) AND send . theme (x _ 2 , x _ 5) AND sandwich . nmod . beside (x _ 5 , x _ 8)
Pred	* sandwich (x _ 5) ; * yacht (x _ 8) ; cockroach (x _ 1) AND send . **agent** (x _ 2 , x _ 1) AND send . recipient (x _ 2 , Sophia) AND send . theme (x _ 2 , x _ 5) AND sandwich . nmod . beside (x _ 5 , x _ 8)

Example 2: Right edge label, but between wrong nodes

In	The girl beside the bed lended the manager the leaf .
Out	* girl (x _ 1) ; * bed (x _ 4) ; * manager (x _ 7) ; * leaf (x _ 9) ; **girl . nmod . beside (x _ 1 , x _ 4)** AND lend . agent (x _ 5 , x _ 1) AND lend . recipient (x _ 5 , x _ 7) AND lend . theme (x _ 5 , x _ 9)
Pred	* girl (x _ 1) ; * bed (x _ 4) ; * manager (x _ 7) ; * leaf (x _ 9) ; lend . agent (x _ 5 , x _ 1) AND lend . recipient (x _ 5 , x _ 7) AND lend . theme (x _ 5 , x _ 9) AND **leaf . nmod . beside (x _ 9 , x _ 4)**

Example 3: Mistaking edge labels

In	The dog noticed that a hippo juggled .
Out	* dog (x _ 1) ; notice . agent (x _ 2 , x _ 1) AND notice . ccomp (x _ 2 , x _ 6) AND hippo (x _ 5) AND juggle . **agent** (x _ 6 , x _ 5)
Pred	* dog (x _ 1) ; notice . agent (x _ 2 , x _ 1) AND notice . ccomp (x _ 2 , x _ 6) AND hippo (x _ 5) AND juggle . **theme** (x _ 6 , x _ 5)

Example 4: Correct nodes, but incorrect edges predicted

In	A dog beside a chair said that a melon on the bed was liked .
Out	* bed (x _ 11) ; dog (x _ 1) AND **dog . nmod . beside (x _ 1 , x _ 4)** AND chair (x _ 4) AND say . agent (x _ 5 , x _ 1) AND say . ccomp (x _ 5 , x _ 13) AND melon (x _ 8) AND **melon** . nmod . on (x _ 8 , x _ 11) AND like . theme (x _ 13 , x _ 8)
Pred	* bed (x _ 11) ; chair (x _ 4) AND say . agent (x _ 5 , x _ 4) AND melon (x _ 8) AND **bed** . nmod . in (x _ 11 , x _ 13) AND like . theme (x _ 13 , x _ 8)

Table 4: Incorrectly predicted logical forms for COGS with strongly-supervised LAGr. Errors are highlighted in bold.

Example 1: Wrong edge predictions

Layer 2	?x0	M3	influenced			director		spouse	M2	?x2	cinematographer			M4				?x1	actor
Layer 1																			
Input	Did	M3	influence	a	film	director	,	marry	M2	's	cinematographer	,	influence	M4	,	and	influence	a	actor
Target	?x1 actor . ?x0 director . ?x2 cinematographer M2 . FILTER M3 != ?x2 . M3 influenced [?x0 ?x1 M4] . M3 spouse ?x2																		
Predicted	?x0 actor . ?x0 director . ?x1 director . ?x2 cinematographer M2 . FILTER M3 != ?x2 . M3 influenced [?x0 ?x1 M4] . M3 spouse ?x2																		

Example 2: Missing node

Layer 2	select_?x0	ns:m.0f8l9c		editor		M1	influenced_ by		?x1	employer	?x2	organizations_founded					M2
Layer 1		nationality															
Input	What	French	film	editor	that	M1	influenced	influenced	a	company	s	founder	and	was	influenced	by	M2
Target	?x1 actor . ?x0 director . ?x2 cinematographer M2 . FILTER M3 != ?x2 . M3 influenced [?x0 ?x1 M4] . M3 spouse ?x2																
Predicted	?x0 actor . ?x0 director . ?x1 director . ?x2 cinematographer M2 . FILTER M3 != ?x2 . M3 influenced [?x0 ?x1 M4] . M3 spouse ?x2																

Example 3: Correct prediction

Layer 2	select_?x0	ns:m.05zppz	ns:m.059j2		editor	director	M3
Layer 1			gender	nationality			
Input	Which	male	Dutch	film	editor	directed	M3
Predicted	select_?x0 director M3 . select_?x0 editor . select_?x0 gender ns:m.05zppz . select_?x0 nationality ns:m.059j2						

Example 4: Correct prediction

Layer 2	select_?x0		ns:m.06mkj	actor		influenced	M2		?x1	actor		
Layer 1	nationality	person										
Input	Who	was	a	Spanish	actor	that	influenced	M2	and	influenced	a	actor
Predicted	?x1 actor . select_?x0 actor . select_?x0 influenced ?x1 . select_?x0 influenced M2 . select_?x0 person . select_?x0 nationality ns:m.06mkj											

Figure 3: Predicted nodes of aligned graphs and resulting queries produced by the best weakly-supervised LAGr with k=5, $\sigma = 10$ on the development set of CFQ. Top two rows show common errors with missing edge labels and missing nodes, and bottom rows show the inferred alignments for correct examples.

Case	Training	Generalization
Subject → Object	A **hedgehog** ate the cake.	The baby liked the **hedgehog**.
Object → Subject	Henry liked a **cockroach**.	The **cockroach** ate the bat.
Primitive → Object	**Paula**	The child helped **Paula**.
Depth generalization	Ava saw the ball **in the bottle on the table.**	Ava saw the ball **in the bottle on the table on the floor.**
Active → Passive	Emma **blessed** William.	A child was **blessed**.

Table 5: Example from Kim and Linzen (2020a) that show various linguistic phenomena from the COGS generalization set.

	Reproduced baselines	.	**Strongly-supervised LAGr with different encoders**			
.	LSTM	Transformer	LSTM$_{sh}$	LSTM$_{sep}$	Transformer$_{sh}$	Transformer$_{sep}$
batch_size	256	128	128	64	128	128
learning_rate	0.004	0.0001	0.0001	0.0004	0.0001	0.0001
scheduler	linear with warmup of 1000 steps	linear with no warmup	linear with warmup of 1000 steps	linear with warmup of 1000 steps	linear with no warmup	linear with no warmup
layers	2	4	2	2	4	4
enc_dim	256	256	256	256	512	512
train_steps	50000	50000	70000	70000	70000	70000
validate_every *(step)*	5000	5000	5000	5000	10000	10000
dropout	0.4	0.1	0.1	0.4	0.4	0.4
attention heads	-	8	-	-	4	4

Table 6: Best hyperparameters for our COGS baseline and strongly-supervised LAGr experiments

	CFQ	
	Weakly-supervised LAGr	
	LSTM$_{sep}$	Transformer$_{sh}$
batch_size	64	256
learning_rate	0.001	0.0004
scheduler	linear with warmup of 1000 steps	linear with warmup of 1000 steps
layers	2	4
enc_dim	512	256
train_steps	200000	200000
validate_every (step)	10000	10000
early_stopping (valid steps)	5	5
dropout	0.4	0.1
attention heads	-	8

Table 7: Best configuration for CFQ weakly-supervised LAGr.