Examining Algebraic Recombination for Compositional Generalisation

Joaquin Cardona Ruiz, Antske Fokkens, Lucia Donatelli
Vrije Universiteit Amsterdam
j.cardonaruiz@student.vu.nl, {antske.fokkens, l.e.donatelli}@vu.nl

Abstract

We introduce DEAR, a compositional semantic parser that extends the LEAR model (Liu et al., 2021) to handle non-projective syntactic derivations. We evaluate our parser on SLOG (Li et al., 2023), a compositional generalisation benchmark focused on structural generalisation. We find that DEAR achieves stronger performance on syntactically complex examples than projective baselines, even as it incurs a modest decrease in overall accuracy due to a larger search space. Analysis of model behaviour confirms a fundamental trade-off: while projective architectures yield efficient, benchmark-aligned learning, non-projective mechanisms provide the expressive capacity needed to capture richer linguistic dependencies. These findings contribute to current discussions on how more linguistically faithful semantic parsers can be developed beyond benchmark-specific biases.

Keywords: Compositional generalisation, Semantic parsing, Non-projectivity

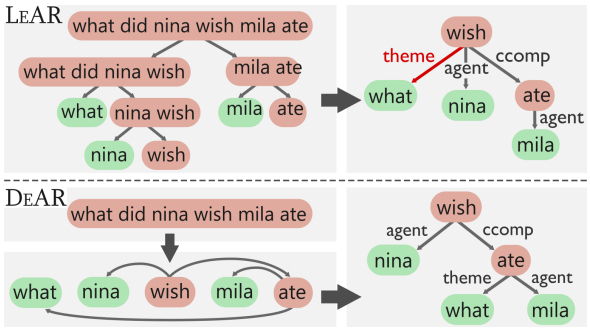

Figure 1: DEAR extends LEAR's compositional parsing architecture by replacing its constituency structure with a dependency structure. This allows DEAR to handle non-projective arguments that LEAR fails to capture, as seen in red.

1. Introduction

Compositional generalisation, the ability to recombine known linguistic components to derive the meanings of novel constructions (Partee, 2004), is an essential component of human language acquisition and understanding. Despite their broad linguistic coverage, sequence-to-sequence (seq2seq) models often fail to generalise compositionally in semantic parsing tasks (Lake and Baroni, 2018; Dehghani et al., 2019a; Herzig and Berant, 2021). When presented with structures requiring the systematic recombination of known elements, such models tend to overfit to surface-level correlations rather than capturing underlying syntactic–semantic regularities that make human generalisation effective.

Models designed to be explicitly compositional, such as LEAR (Liu et al., 2021) and the AM-Parser (Groschwitz et al., 2018), address this limitation by deriving an intermediate syntactic representation from the input that can be algebraically recombined into a target semantic graph. These architectures can achieve strong compositional generalisation while being very data efficient. Nevertheless, syntactic assumptions built into these models often impose projectivity constraints that limit their overall expressivity. As a result, they struggle to capture the flexible, non-local dependencies that characterise natural language *in the wild* beyond more predictable tendencies of structures found in benchmark datasets (Shaw et al., 2021) (Sec. 2).

To examine the tension between *expressivity* and *efficiency* in compositional language models, we introduce DEAR (**De**pendency **A**lgebraic **R**ecombination), a compositional semantic parser that extends LEAR's architecture to model a wider range of structures (Sec. 3). DEAR's most notable introduction is the use of an unconstrained dependency intermediate representation, rather than a constituency one, which allows for the modelling of non-projective structures (Fig. 1). This design draws inspiration from empirical evidence of non-projective structures across languages (Havelka, 2007) and theoretical work demonstrating that non-projective mechanisms possess greater semantic expressive capacity than projective ones (Venant and Koller, 2019). All code developed for this study is publicly available.[1]

We explore the performance and architectural implications of both models through the SLOG benchmark (Li et al., 2023) (Sec. 4, 5). SLOG maps English sentences to logical forms representing their meaning, and features a generalisation split that requires models to recombine seen structures

[1] https://github.com/JoaquinCardonaRuiz/thesis-compositionality

Proceedings of the 15th Workshop on Cognitive Modeling and Computational Linguistics (CMCL) @ LREC 2026, pages 230–249
16 May 2026.

to generalise to new ones. This focus on *structural generalisation* is crucial, given that recent work has demonstrated that lexical generalisation, the generalisation of individual lexical units to unseen positions, is easier than structural generalisation (Weißenhorn, 2022). Notably, SLOG includes instances of non-projectivity, the presence of crossing edges in an ordered dependency tree (analogous to the non-contiguity of constituents in a constituency tree, see Hays, 1964 and Nivre, 2008). The dataset is thus an ideal setting to probe whether more expressive parsing mechanisms confer empirical benefits.

This study makes the following contributions:

1. A novel compositional parser, DEAR, which extends LEAR to support non-projective syntactic structures. DEAR outperforms projective parsers on specific structurally complex phenomena.
2. An empirical study of expressivity–efficiency trade-offs in compositional parsing, showing how controlled non-projectivity impacts both search complexity and generalisation.

Our results affirm the potential of principled linguistic design in semantic parsers to generalise beyond benchmark-specific biases to more versatile natural language structures, and highlight the value of interpretability in the development and evaluation of NLP tools (Sec. 6).

2. Compositional Generalisation

The Principle of Compositionality states that the meaning of a complex expression is determined by the meanings of its parts and the rules combining them.[2] This principle underlies humans' ability to generate infinite meanings from finite elements (Partee, 2004; Chomsky, 1965; Fodor and Pylyshyn, 1998).

Recent work has examined whether language models can replicate this ability in systematic semantic parsing tasks (Lake and Baroni, 2018; Dehghani et al., 2019a), raising the question of why compositional generalisation matters for NLU systems. From a pragmatic perspective, compositionality in a language model serves as a domain-specific form of parameter-sharing, reducing the number of parameters needed by sharing components to process similar structures in arbitrary contexts. Learning to identify common patterns independently of position, rather than learning each structure anew, removes a hard barrier to a model's ability to generalise, and is a major factor for sample efficiency (Lake et al., 2015): a model which can leverage compositionality to correctly assess infinite recombinations of seen components needs significantly fewer training examples to generalise.

More conceptually, the in-distribution approach often employed in supervised learning is not well suited to evaluate the linguistic capabilities of modern NLU systems, as generalisation becomes hard to distinguish from memorisation of their vast training datasets (Kim et al., 2022). To examine a model's ability to generalise out-of-distribution, and therefore to learn, evaluation can be grounded on the principles of human understanding of language. The compositionality of language means humans innately display a capacity for *compositional generalisation*, recombining known components to derive the meaning of unseen constructions (Partee, 2004). Semantic parsing benchmarks can target this capability by presenting primitives and basic structures during training, but withholding examples for evaluation which require recombining these elements into unseen positions and structures.

Importantly, compositional behaviour depends on how models represent and constrain syntax. Most relevant for this study is the constraint of projectivity, which in dependency syntax requires that for every arc from a head to a dependent, all words linearly intervening between them are dominated by that head (Jurafsky and Martin, 2026). Most sentences in a language are projective (Melčuk, 1988), yet non-projective dependencies arise naturally in valid constructions, most notably in languages with flexible word order (Havelka, 2007), but also in English, for instance in cases of long-distance *wh*-movement. In projective systems, composition therefore occurs only between adjacent constituents (Nivre, 2008). While this constraint simplifies inference, it limits expressivity: natural language frequently exhibits non-projective dependencies that require non-local composition (Howell and Zamaraeva, 2018; Venant and Koller, 2019). This section surveys datasets and modelling approaches that probe this tension between structural simplicity and expressive capacity in compositional generalisation.

2.1. Datasets

Semantic parsing benchmarks designed to specifically target compositional generalisation have overwhelmingly been synthetic in nature, and thus not representative of language *in the wild* (Shaw et al., 2021). The most notable early efforts are SCAN (Lake and Baroni, 2018) and CFQ (Dehghani et al., 2019a). Both map simple sentences to computer commands, and feature **generalisation sets**, which present sentences with known atoms in previously unseen positions. The dataset examined in this work, SLOG (Li et al., 2023), and its predecessor COGS (Kim and Linzen, 2020), introduce sentences closer to natural language in diversity

[2] Usually attributed to Gottlob Frege; see Janssen (2012).

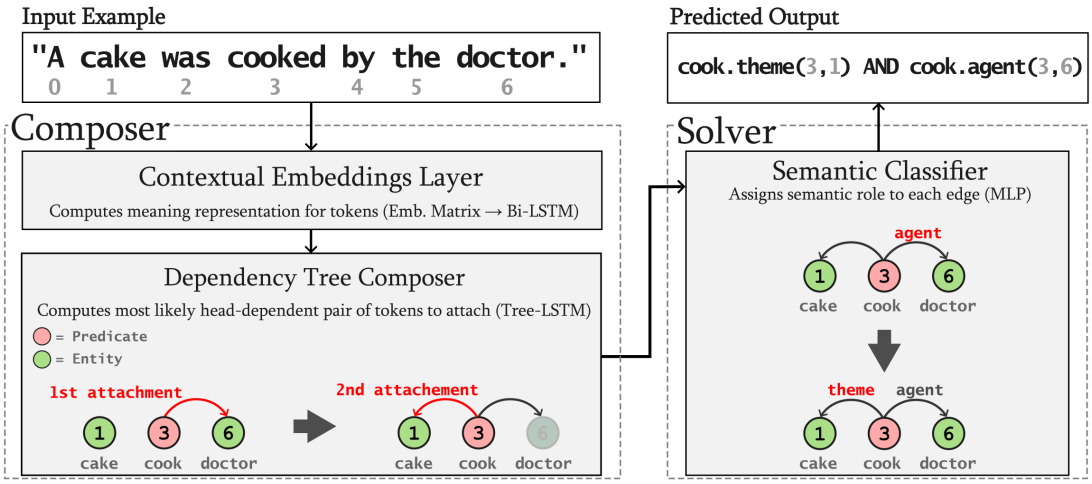

Figure 2: Overview of DEAR's architecture, showcasing an example from COGS. The input sentence (top-left) gets fed into the *Composer* module (left), which creates a dependency tree. Then the *Solver* (right) labels it, and produces an output (top-right).

and complexity, mapped to meaning representations based on the Lambda Calculus. This frames semantic parsing as a sequence-to-sequence problem, as shown in Example (1).

(1) **Sentence**: *Liam liked to run.*
Target Form: `like.agent(x_1, x_0) AND like.xcomp(x_1, x_3) AND run.agent(x_3, x_0)`

SLOG extends COGS to more structurally complex generalisation cases, most notably non-projective dependencies. Test cases are grouped into categories of 1000 cases each, in which a specific systematic gap is evaluated. Where most categories in COGS are focused on *lexical generalisation*, SLOG targets all its categories towards structural recombination, which requires recombining entire syntactic structures to arrive at previously unseen ones.

2.2. Approaches

Despite recent seq2seq models displaying seemingly impressive linguistic capabilities in both comprehension and production, they perform inconsistently on compositional generalisation benchmarks when presented with linguistic structures withheld during training (Lake and Baroni, 2018; Loula et al., 2018; Andreas, 2020; Oren et al., 2020; Dehghani et al., 2019a; Herzig and Berant, 2021). This suggests a lack of the same systematic generalisation capabilities humans possess, especially when it comes to structural—rather than lexical—recombination (Zhu et al., 2021; Yao and Koller, 2022; Zheng and Lapata, 2022; Li et al., 2023).

Compositional parsers architecturally model a latent structure of the input expression and recombine its components to derive the correct meaning representation. Many efforts have focused on the SCAN dataset, making use of either quasi-synchronous context-free grammars (Kim, 2021; Qiu et al., 2022) or constituency trees (Liu et al., 2020, 2021; Herzig and Berant, 2021) as latent representations. LEAR extends constituency-based approaches on simpler benchmarks to COGS, achieving near-perfect performance (97.7% accuracy). It learns a latent constituency tree through a Tree-LSTM (Tai et al., 2015) and assigns semantic operations to each node through several MLPs. Another compositional parser, the AM parser (Groschwitz et al., 2018), instead implements a dependency parser based on stacked Bidirectional LSTMs and an MLP supertagging component. Its application to the SLOG benchmark in Li et al. (2023) yields outstanding results (70.8% accuracy).

3. DEAR

We introduce DEAR, a compositional semantic parsing model that builds on LEAR (Liu et al., 2021). As in LEAR, the model consists of two components: a *Composer*, which induces a latent syntactic structure, and a *Solver*, which assigns semantic relations to structural edges. The two stages are optimised through reinforcement learning using separate reward signals. An overview of DEAR's components, alongside an example input and output is shown in Figure 2.

The primary departure from LEAR lies in the choice of structural representation. Whereas LEAR constructs binary constituency trees, DEAR induces a *dependency* tree over abstracted lexical items. Dependency-based composition allows the model to directly represent predicate–argument relations and, crucially, does not impose projectivity constraints during tree construction. As a result, DEAR can produce crossing dependencies when

they are semantically licensed, making the architecture more suitable for naturalistic text and languages exhibiting freer word order. An overview of LEAR's architecture, similar to the one presented for DEAR in Figure 2, is shown in Appendix B.

3.1. Input Token Classification and Representation

Following LEAR, DEAR assumes a predefined lexicon of *content words* in the training corpus, each typed as either an *entity* or a *predicate* according to SLOG's context-free grammar. Input tokens that appear in this lexicon are labelled with their respective type, and those that do not are labelled as *non-content words*. The model therefore does not identify predicate–argument structure from raw text; instead, it focuses on determining the correct relations between a known closed set of primitives. This design isolates the model's ability to generalise to new structures, without confounding effects from lexical ambiguity. Details on the construction of this lexicon are provided in Appendix A.2.

All tokens are mapped to 256-dimensional embeddings and fed into a bidirectional LSTM encoder. The encoder consists of independent forward and backward LSTMs with hidden size 256. Their outputs are concatenated (yielding 512 dimensions) and passed through a learned linear projection to 512 dimensions, followed by a $\tanh$ activation. The resulting vector is then split evenly into a hidden state and a cell state (h_i, c_i) for each token, which serve as its initial representation for downstream composition. Tokens also retain their label as either *entity*, *predicate*, or *non-content word*.

3.2. *Composer*: Structure Induction Via Dependency Composition

The *Composer* iteratively constructs a dependency tree over *content words*. Its architecture, described below, is additionally illustrated in Figure 3, alongside that of the Solver. As in LEAR, *non-content words* are masked out from the set of candidate nodes, although information about them is preserved in the contextualised BiLSTM representations of *content words*.

At each iteration, the Composer evaluates every ordered pair (i, j) of remaining content-word nodes by first computing a candidate composed parent state via the binary Tree-LSTM:

$$(\tilde{h}_{i\leftarrow j}, \tilde{c}_{i\leftarrow j}) \;=\; \mathrm{TLSTM}\big((h_i, c_i), (h_j, c_j)\big), \qquad (1)$$

where the Tree-LSTM computes gate activations from a linear projection of the concatenated child hidden states and combines child cell states (see Appendix A.1 for notes on our implementation of Tai et al. (2015)'s Tree-LSTM). Each candidate composed hidden state is then scored by taking its dot product with a learned parameter vector $\mathbf{q}$, producing a single real number used as the unnormalised logit score for the arc:

$$s(i, j) \;=\; \tilde{h}_{i\leftarrow j}^{\top}\mathbf{q}. \qquad (2)$$

A categorical distribution over valid arcs is formed from these scores after masking out invalid attachments (self-loops and previously attached dependents). During training, an arc is sampled from this distribution; at test time, the highest-scoring arc is selected deterministically. The chosen dependent is composed into the head (the head's state is replaced by the composed state) and removed from further consideration. This iterative process continues until a single root remains. Because attachment decisions depend only on pairwise token representations and no span-based constraints are imposed, the resulting trees may be non-projective.

This design contrasts with LEAR, which iteratively scores all valid adjacent spans of content words and merges the best-scoring span into a single constituent. LEAR 's design enforces binary-branching constituency structure and therefore cannot produce crossing edges.

Furthermore, because the representations of chosen dependent tokens are composed into that of their corresponding heads, the head's state becomes a summary of the entire subtree rooted at that node, enabling the model to perform genuine recursive composition.

3.3. *Solver*: Semantic Labelling

Given the induced tree, the *Solver* assigns a semantic relation to each directed edge. We train separate feed-forward classifiers (one-hidden-layer MLPs) for different head–dependent type configurations: *predicate–entity*, *predicate–predicate*, and *entity–entity*. Edges of type *entity–predicate* are deterministically assigned a *relative clause* label.

Each MLP takes as input the concatenation of the hidden states of the head and dependent:

$$\lambda_{h,d} = \mathrm{MLP}_{\tau(h),\tau(d)}([h_h; h_d]). \qquad (3)$$

Concretely, each classifier consists of a linear projection from 2×256 to 256, followed by a ReLU activation and a final linear layer mapping to the label set associated with that type pair (e.g., *agent*, *theme*, *recipient* for *predicate–entity* edges).

Post-processing expands relative clauses and introduces the reentrancies required by the target logical form, converting the dependency arborescence into a directed graph that no longer satisfies strict tree constraints. Further details on post-processing are provided in Appendix A.4.

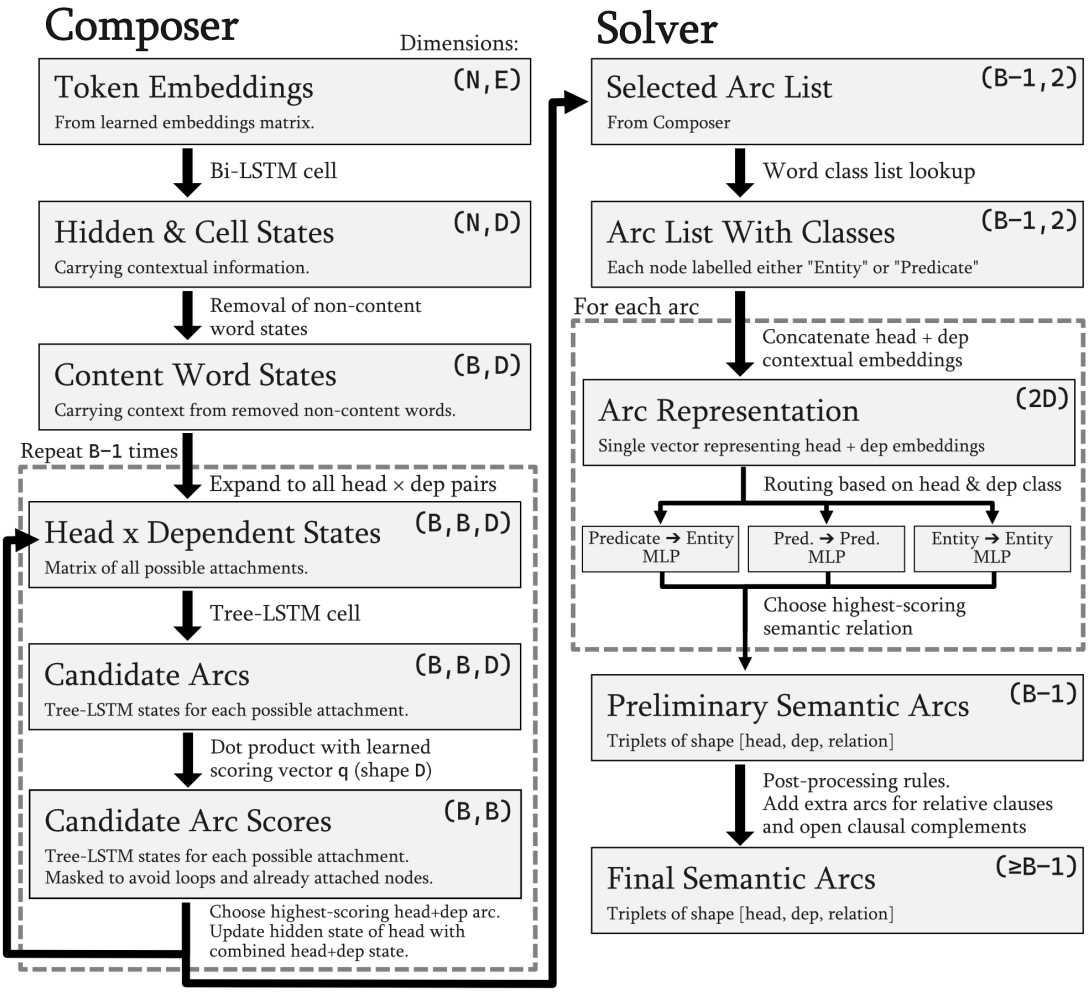

Figure 3: Detailed diagram of DEAR's architecture. The Composer (left column) takes the input sentence and predicts dependency edges, and the Solver (right column) labels those edges with semantic relations, producing a final labelled graph. At each step, the dimensions of the vector representations are shown to the right of the box, where N is the number of input tokens, B is the number of content words ($B \leq N$), E is the word embedding dimension, and D is the hidden state dimension, used for the Bi-LSTM, Tree-LSTM, and MLPs.

3.4. Reinforcement Learning Objective

Training follows the hierarchical reinforcement-learning setup of Liu et al. (2021). For each input, the model samples K complete derivations (we use K=15), each consisting of a compositional structure proposed by the *Composer* and a semantic labelling of all induced edges produced by the *Solver*. For a predicted derivation we denote by $P = \{(h, d, \lambda)\}$ its labelled edges, and by $\bar{P}$ the same edges without labels; G and $\bar{G}$ denote the corresponding gold sets.

The structural and semantic components receive separate rewards. The *Composer* receives a reward for correctly recovering unlabelled dependency arcs, while the *Solver* receives a reward for correctly labelling those edges that were structurally correct:

$$R_{\text{comp}} = H\Big(\tfrac{|\bar{P}\cap\bar{G}|}{|\bar{P}|},\ \tfrac{|\bar{P}\cap\bar{G}|}{|\bar{G}|}\Big)$$

$$R_{\text{solv}} = \begin{cases} \dfrac{|P \cap G|}{|\bar{P} \cap \bar{G}|}, & |\bar{P} \cap \bar{G}| > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

For each sampled derivation we record the summed log-probability of the Composer's discrete attachment decisions ($\log p_{\text{comp}}$) and the summed log-probability of the Solver's label decisions ($\log p_{\text{solv}}$). Each module is then updated *independently* using REINFORCE with a simple mean-baseline. Here $E[R_{\text{comp}}]$ and $E[R_{\text{solv}}]$ denote the empirical mean reward over the K sampled derivations for the corresponding module, and p_{comp} and p_{solv} are the model's own probabilities assigned to the specific sequence of sampled decisions:

$$\begin{aligned} \mathcal{L}_{\text{comp}} &= -\big(R_{\text{comp}} - E[R_{\text{comp}}]\big)\ \log p_{\text{comp}} \\ \mathcal{L}_{\text{solv}} &= -\big(R_{\text{solv}} - E[R_{\text{solv}}]\big)\ \log p_{\text{solv}} \end{aligned} \tag{5}$$

We optimise the model using two separate Adadelta optimisers. The *high-level* optimiser updates all parameters associated with structural decisions—namely the *Composer*, the Tree-LSTM, and the input encoder—while the *low-level* optimiser updates only the Solver's MLP classifiers. Gradients for the two modules are kept disjoint: after computing the relevant loss, we apply gradient clipping to the corresponding parameter subset and perform an optimiser step only for that subset. This ensures that the *Composer* receives no gradient signal from semantic mislabelling, and that the *Solver* is trained solely from label rewards on structurally correct arcs. In this respect our model differs from LEAR, which trains both modules using a single shared reward.

4. Experimental Setup

We evaluate the DEAR model introduced in this work on the SLOG benchmark, and report per-category accuracies. Additionally, we evaluate the LEAR model (Liu et al., 2021) and two pre-trained large language models (Llama 3 8b, Mistral 7b) to provide context for DEAR's performance and examine the impact of the different model architectures. We re-report Li et al. (2023)'s results for the AM-Parser and a vanilla Transformer. Details on computational resources and hardware setup can be found in Appendix C.2, and the hyperparameters chosen for each model can be found in Appendix C.1. We evaluate the models based on accuracy: the percentage of samples in which the model produces a semantically equivalent output to the ground truth. Reported results are measured on SLOG's generalisation set.

4.1. Adapting LEAR to SLOG

LEAR's functionality is heavily reliant on heuristics and pre-computed alignment tables specific to the COGS dataset it was developed for. Adapting the model to process the SLOG dataset was not trivial. Implemented changes are outlined below.

Content questions: The SLOG dataset incorporates syntactic structures not present in COGS, amongst them content questions. In their target logical form, *wh*-words are abstracted with a generic *interrogated entity* symbol, rather than being treated as indexed tokens, as shown in Example (2). In order to keep LEAR's architecture unchanged, sentences were pre-processed to treat *wh*-words as *entity* words.

(2) **Sentence**: *Who did Henry adore?*
Target Form: `adore.agent(x_3,x_2) AND adore.theme(x_3,?)`
Adjusted Form: `adore.agent(x_3,x_2) AND adore.theme(x_3,x_0)`

Standalone NP: SLOG introduces sentences without a root predicate, absent from COGS, as shown in Example (3). To account for LEAR's architectural constraints, which demand a root predicate to assign arguments to, a *null predicate* is introduced without an index value, and assigned the root NP as a theme.

(3) **Sentence**: *The cake on a tree.*
Target Form: `cake.nmod.on(x_1, x_4)`
Adjusted Form: `null.theme(x, x_1) AND cake.nmod.on(x_1, x_4)`

Relative Clauses: Unlike COGS, SLOG features relative clauses, as shown in Example (4). LEAR's output representation was updated to allow relative clause tags to be present, though the model remains unable to predict such relations due to being architecturally restricted from creating *entity→predicate* edges.

(4) **Sentence**: *Liam was lent the rose that was liked.*
Target Form: `lend.recipient(x_2, x_0) AND lend. theme(x_2, x_4) AND rose.relcl(x_4, x_7) AND like.theme(x_7, x_4)`

These changes enable the evaluation of LEAR on the SLOG dataset, which served as a crucial step towards the development of the DEAR model, informing the architectural changes to be made, and providing a baseline for comparison.

5. Results & Error Analysis

Per-category results are presented in Table 1 and examined in detail in the following sections. While the most relevant categories are briefly described below, Appendix D contains a complete description of all categories, alongside example sentences, for ease of reference. Overall, compositional models consistently outperform seq2seq ones. DEAR is the only such model to achieve non-zero performance in SLOG's non-projective category, *Wh Q. long movement*. Pre-training aids performance of seq2seq models, but obfuscates the systematic gap between training and evaluation data, leading to potential overestimation of compositional capabilities (Kim et al., 2022).

5.1. LEAR's Performance on SLOG

LEAR achieves an accuracy of 83.2% on SLOG's in-distribution validation set and 54.2% on its generalisation set. Several of SLOG's generalisation categories present a significant challenge to LEAR's architecture. Below is an overview of those that prove architecturally impossible for the model to accurately predict, and therefore have an in-category accuracy of 0%.

RC in subj. NPs, RC in ind. object NPs & Ind. object-extracted RC: These categories introduce systematic gaps by presenting models with relative

Category	LEAR$^{\kappa}$	DEAR$^{\kappa}$	Transf.†	AM-Parser$^{\dagger\kappa}$	LLaMA 3$^{\pi}$	Mistral$^{\pi}$
Shallower PP rec.	100.0	99.4	98.7	100.0	96.0	94.9
Deeper PP rec.	100.0	38.0	13.1	100.0	21.5	29.3
Shallower tail CP rec.	100.0	99.0	32.6	100.0	80.1	66.8
Deeper tail CP rec.	100.0	31.9	0.2	100.0	8.5	7.9
Shallower center emb.	**0.0**	**95.9**	0.0	100.0	17.3	0.0
Deeper center emb.	**0.0**	**23.4**	0.0	99.5	0.1	0.3
PP in subj. NPs	66.9	**16.9**	0.0	57.6	46.0	16.2
PP in ind. object NPs	67.4	**0.6**	42.5	90.4	74.9	61.3
RC in subj. NPs	**0.0**	**3.6**	0.0	55.8	22.6	28.0
RC in ind. object NPs	**0.0**	**1.8**	34.4	74.4	62.4	47.5
Ind. obj-extracted RC	**0.0**	**99.9**	4.7	0.0	6.1	0.0
Dir. object wh Q.	48.7	70.1	2.8	29.4	5.5	40.6
Ind. object wh Q.	100.0	94.1	35.9	41.4	11.1	17.7
Active subj. wh Q.	100.0	99.0	96.7	99.8	96.5	97.7
Passive subj. wh Q.	100.0	100.0	27.4	100.0	18.7	40.1
Wh Q. modified NPs	38.0	14.9	17.6	55.6	27.3	31.5
Wh Q. long movement	**0.0**	**7.8**	4.0	0.0	10.9	33.6
Overall	54.2	53.1	24.2	70.8	35.6	36.1

Table 1: Main Results. Mean Accuracy (%) for models evaluated on each of SLOG's categories. Label κ indicates a compositional model, and π indicates a pre-trained model. Results labelled † are re-reported from Li et al. (2023) for comparison.

clauses in previously unseen grammatical positions. This does not present a challenge to LEAR's capability for generalisation, but to its inductive bias. Relative clauses introduce *semantic reentrancy*, wherein a single entity fills a role in more than one predicate. LEAR's semantic dependency tree does not allow for reentrancy, as all entities get assigned a role in a single predicate. Importantly, this is a failure of LEAR's *Solver* component, not of its *Composer*. The *Composer* module is capable of producing the correct syntactic structure. However, because of the *Solver*'s limitations outlined above, it is never rewarded for doing so.

This pattern holds for all sentences featuring relative clauses in both the training and generalisation sets. 34.5% of predictions for the *RC in subject NP* category, 53.6% for *RC in indirect object NP*, and 25.3% for *Ind. object-extracted RC* contain only errors caused by this architectural limitation. The remaining predictions contain at least one *unforced* error, which consists of entities being assigned to erroneous roles by the *Solver*. **Shallower center embedding & Deeper center embedding:** The accuracy of 0% on these categories is due to the same reasons as the categories above: LEAR is incapable of modelling relative clauses.

Wh Q. long movement. This category presents sentences that exhibit *long wh*-movement, wherein the *wh*-word is separated from its corresponding predicate by a subordinate clause. This makes the structures in this category non-projective. LEAR's *Composer* creates phrases by selecting one pair of adjacent tokens (or previously composed phrases) in each iteration to merge. This adjacency restriction means the model is incapable of producing such non-projective trees, and thus the accuracy is 0%.

(5) **Sentence**: *What did Liam wish that the father ate?*
Latent syntax:
`[[what [liam wish]] [father ate]]`
Resulting edges:
`agent(wish, liam), theme(wish, what), ccomp(wish, eat), agent(eat, father)`

In Example (5), the *wh*-word *what*, which should be the *theme* of predicate *ate*, is separated from its predicate by several tokens. LEAR's prediction erroneously creates an edge attaching the *wh*-word to its nearest predicate, highlighted in red. This limitation is inherent to LEAR's Composer, which can only attach tokens to adjacent phrases, and is a major motivation for the development of DEAR in the present study.

5.2. DEAR's Performance on SLOG

DEAR achieves an accuracy of 97.8% on SLOG's in-distribution validation set and 53.1% on its generalisation set. It presents non-zero accuracies on all categories in SLOG, unlike comparable models, as can be seen in Table 1. Nevertheless, in-category performance is generally lower than other compositional models, which may stem from the relation between model complexity and the limited dataset size. This is further discussed in Section 6. Details on time and computational expense can be found in Appendix C.2. Below is an overview of the categories in which the results achieved by DEAR differ substantially from those achieved by LEAR.

Deeper PP recursion & Deeper tail CP recursion: DEAR achieves near perfect accuracy on the categories presenting *shallower* recursions than those seen in training. In the categories presenting

Category	5	6	7	8	9	10	11	12
PP recursion	93.6	91.2	68.8	40.0	20.8	1.6	1.6	0.8
Comp. Reward	99.0	97.7	90.9	82.4	75.5	66.0	62.7	58.1
Solv. Reward	100.0	100.0	99.9	99.8	99.7	99.9	99.7	99.9
Tail CP recursion	70.4	59.2	30.4	11.2	6.4	2.4	0.8	0.8
Comp. Reward	92.5	88.0	80.8	71.6	66.6	69.6	64.7	60.9
Solv. Reward	98.6	99.3	99.1	99.1	99.5	99.1	99.6	99.6
Center embedding	93.6	61.6	20.8	3.2	0.8	0.8	0.0	0.0
Comp. Reward	98.3	89.9	76.4	63.4	58.2	55.4	48.0	45.5
Solv. Reward	99.4	95.4	93.1	94.0	95.2	93.5	93.0	90.7

Table 2: Mean Accuracy (%), Mean Composer Reward (%) and Mean Solver Reward (%) achieved by DEAR at each level of Recursion Depth (columns) for the categories Deeper PP recursion, Deeper tail CP recursion, & Deeper center embedding (rows). Deeper recursion levels trigger a fall in *Composer* reward and overall accuracy, but not on *Solver* reward.

deeper recursion, however, performance is mixed. A closer inspection of the results shows that the model struggles with the deeper levels of recursion, with performance dropping as the number of nested clauses in the sentence grows, as can be seen in Table 2.

This indicates DEAR is able to generalise up to recursion depths unseen during training, but is limited in its capacity to accurately model long or deeply nested sentences. The *Solver* rewards being mostly unaffected by sentence depth show that this is mostly a limitation of DEAR's *Composer*.

Shallower center embedding & Deeper center embedding: Thanks to the inclusion of *entity* to *predicate* edges in DEAR, and the resulting ability for the *Solver* to tag dependencies as relative clauses, these categories are able to be captured with non-zero accuracy. However, the same pattern of decreasing accuracy with increasing depth can be seen in Table 2.

RC in subj. NPs, RC in ind. object NPs & Ind. object-extracted RC: Similarly to the categories involving center embedding, the integration of relative clauses to the model allows for these categories to be predicted for with non-zero accuracy. Nevertheless, accuracies on *RC in subj. NPs* and *RC in ind. object NPs* remain low. Alongside the similarly low scores in *PP in subj. NPs* and *PP in ind. object NPs*, this reflects the model's difficulties transferring multi-token sequences to positions unseen during training, which these four categories require. This is further confirmed by the near-perfect performance on *Ind. obj-extracted RC*.

Notably, *RC in subj. NPs*, which requires NPs seen in direct object positions to be generalised to subject positions, sees mostly failures in the *Composer*. These involve either correctly parsing the transferred NP, but misplacing it as the root of the entire sentence, or fully failing to capture it. These syntactic failures occur in 51% of sentences in this category, with only 14% displaying any *Solver* errors. *RC in ind. object NPs*, which requires direct object NPs to be generalised to indirect object positions, presents mostly *Solver* errors, which appear in 56% of sentences, as opposed to 20% for the *Composer*. *Solver* errors in both categories almost entirely consist of mislabelled roles within relative clauses.

Wh Q. long movement: 92.2% of sentences in this category are classified incorrectly, exhibiting the same fault present in LEAR's predictions. The *wh*-words introducing the questions are attached to their nearest predicate, rather than to the correct distant predicate. DEAR is architecturally capable of modelling such sentences correctly, as indicated by the non-zero accuracy, but in practice falls short of consistently identifying them. This limitation of the *Composer* seems to apply uniformly to most sentences in this category, without an identifiable pattern in the few sentences that are predicted correctly. This points to the relationship between the increased model complexity of DEAR and the limited training data as the culprit, which will be discussed in Section 6.

6. Discussion

Our results show that the choice of latent syntactic representation substantially affects both a model's expressivity and its computational efficiency. LEAR 's *Composer* encodes a strong inductive bias aligned with simple English syntax. This projective design limits expressivity for non-projective constructions but sharply reduces the syntactic search space compared to DEAR 's dependency-based approach. Concretely, LEAR 's space grows exponentially with sentence length, following the Catalan sequence, whereas DEAR 's grows super-exponentially according to Cayley's formula (See Appendix A.5). As a result, DEAR 's search space becomes quickly intractable for long or recursive constructions, such as PP and CP embeddings.

Sentence length also mediates generalisation. On SLOG's validation set—dominated by short sentences—DEAR outperforms LEAR (97.8% to 83.2%), but performance equalises on the gen-

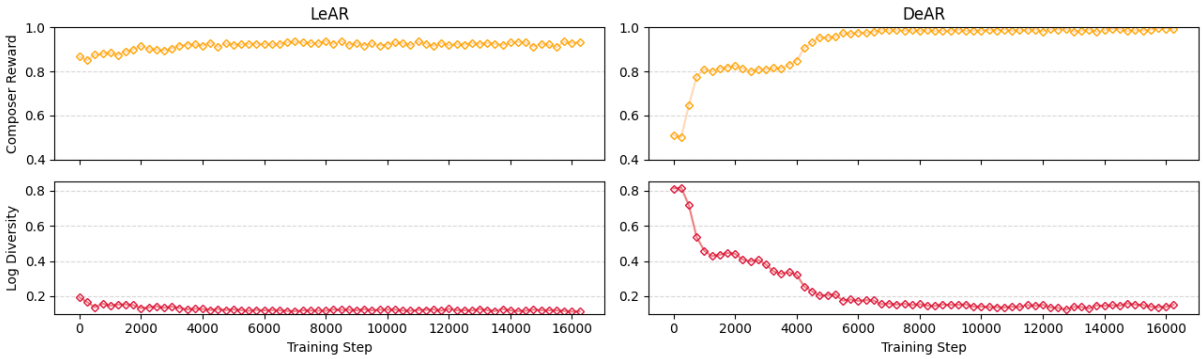

Figure 4: Mean *Composer* Reward (Top) and Diversity (Bottom) during training for LEAR (Left) and DEAR (Right). Each data-point aggregates 500 sentences.

eralisation set (53.1% to 54.2%), which includes longer and more complex sentences. This inversion reflects the combinatorial growth of DEAR 's hypothesis space with increasing length.

Training dynamics further reveal this trade-off (Figure 4). Visualisation of model predictions shows that DEAR initially explores a much wider range of syntactic hypotheses (diversity ≈ 0.8 to LEAR 's ≈ 0.2), leading to slower convergence and less consistent structural reuse. Although DEAR ultimately achieves higher *Composer* rewards, these do not translate to improved generalisation—likely because LEAR 's constrained search promotes more transferable compositional patterns.

These findings are in line with the well-established inverse relationship between the size of the search-space in a reinforcement learning algorithm and its sample-efficiency (Yu, 2018), as well as computational efficiency in dependency parsers in general (Gómez-Rodríguez, 2016). A larger training set would likely be the most impactful factor in allowing DEAR to converge to a stable and transferable solution. Still, there are other considerations beyond the quantity of data available. The models' Lambda-style compositional architectures, which enforce a strict separation between syntactic and semantic representations, mean that semantic placeholders cannot be used to limit the search space (e.g. a predicate having filled all its expected semantic roles, thus not allowing more to be attached, see Venant and Koller (2019)). Still, other linguistically motivated constraints on DEAR's dependency-based *Composer* are feasible. Much work has been dedicated to explore potential constraints of non-projective dependency parsers that allow for the expressivity necessary to model natural language without the combinatorial explosion of the search space that leads to poor efficiency (Kahane et al., 1998; Nivre, 2006; Tratz and Hovy, 2011; Kuhlmann, 2013). The interpretability of DEAR may prove valuable in exploring the impacts of such constraints in the future. We hope this study serves to highlight the necessity of such theoretically-informed methods to incorporate non-projectivity while constraining expressivity in ways that are minimally detrimental to performance on natural, multilingual samples.

7. Conclusion

We introduce DEAR, a dependency-based compositional semantic parser that extends the expressivity of LEAR by handling non-projective syntax. Our experiments on the SLOG benchmark demonstrate that relaxing projectivity constraints enables DEAR to capture a broader range of syntactic–semantic dependencies. However, the additional expressivity comes at the cost of overall performance, particularly in deeply nested sentences, confirming a known trade-off between achieving the expressivity necessary to capture natural language and maintaining efficiency by reducing a model's search space. While systems that rely heavily on dataset-specific assumptions like LEAR are beneficial in demonstrating the value of compositional architectures, designs with increased expressivity are inevitably necessary as benchmarks evolve to better approximate the structural diversity of natural language.

Future work can explore integrating linguistically motivated constraints to temper DEAR 's combinatorial explosion while retaining its expressive advantages. Additionally, larger and more structurally diverse datasets could reveal whether the observed trade-offs persist in more naturalistic settings.

8. Limitations

Limitations of this study include the synthetic nature of the datasets utilised, which limits the applicability of any results to real-world natural language data (see Shaw et al., 2021). A focus on English as an input language provides another, more general limitation, as different languages present unique challenges to the assumptions made by syntactic models, particularly regarding the frequency of non-projective structures. Lastly, the framing of the task assumed by the chosen datasets is an inherent simplification of all that could potentially be captured within semantic parsing. Semantic factors like quantifiers, events and their place in time, conditionals and truth conditions, and co-reference are left out of scope (see Flickinger et al., 2014).

From a methodological standpoint, the results reported here are based on a single experimental run per model due to computational constraints, and no systematic hyper-parameter search was conducted. While this limits the robustness of the quantitative comparisons, we verified that the observed trends were stable across preliminary exploratory runs. Future work should aim to replicate these results under multiple random seeds and systematically tuned hyper-parameters to ensure statistical reliability.

A natural progression of this work and of related studies may be to expand existing compositional generalisation benchmarks to more closely reflect the diversity and complexity of natural language, to cover languages beyond English, and to adopt more comprehensive meaning representations. Furthermore, the potential of recent architectural innovations such as Universal Transformers (Dehghani et al., 2019b; Tan et al., 2023) in explicitly compositional models would be a fruitful area for further work.

9. Ethical considerations

This work presents a methodological contribution to compositional semantic parsing in a benchmark-specific setting, with no immediate real-world deployment. As such, it is not expected to pose direct risks of harm or misuse. We note, however, that the study involves fine-tuning two large language models, which entails a non-negligible environmental cost due to energy consumption associated with GPU usage. All other experiments, namely the training of LEAR and DEAR, were performed on CPU hardware, with comparatively minimal resource demands. Finally, all datasets used in this study are synthetic and publicly available, containing no personally identifiable information or sensitive content.

10. References

Jacob Andreas. 2020. Good-Enough Compositional Data Augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7556–7566, Online. Association for Computational Linguistics (ACL).

Noam Chomsky. 1965. *Aspects of the Theory of Syntax*, 50th edition. The MIT Press, Cambridge, MA.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019a. Measuring Compositional Generalization: A Comprehensive Method on Realistic Data. In *International Conference on Learning Representations*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019b. Universal transformers. In *International Conference on Learning Representations*.

Dan Flickinger, Emily M. Bender, and Stephan Oepen. 2014. Towards an Encyclopedia of Compositional Semantics: Documenting the Interface of the English Resource Grammar. In *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC'14)*, pages 875–881, Reykjavik, Iceland. European Language Resources Association (ELRA).

Jerry A. Fodor and Zenon W. Pylyshyn. 1998. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1):3–71.

Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2018. AMR dependency parsing with a typed semantic algebra. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1831–1841, Melbourne, Australia. Association for Computational Linguistics (ACL).

Carlos Gómez-Rodríguez. 2016. Restricted non-projectivity: Coverage vs. efficiency. *Computational Linguistics*, 42(4):809–817.

Jiří Havelka. 2007. Beyond projectivity: Multilingual evaluation of constraints and measures on non-projective structures. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 608–615, Prague, Czech Republic. Association for Computational Linguistics (ACL).

David G. Hays. 1964. Dependency theory: A formalism and some observations. *Language*, 40(4):511–525.

Jonathan Herzig and Jonathan Berant. 2021. Span-based semantic parsing for compositional generalization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 908–921, Online. Association for Computational Linguistics (ACL).

Kristen Howell and Olga Zamaraeva. 2018. Clausal modifiers in the grammar matrix. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2939–2952, Santa Fe, NM, USA. Association for Computational Linguistics (ACL).

Theo Janssen. 2012. Compositionality: Its Historic Context. In *The Oxford Handbook of Compositionality*. Oxford University Press, Oxford.

Daniel Jurafsky and James H. Martin. 2026. *Speech and Language Processing*, 3rd edition. Draft of January 6, 2026. https://web.stanford.edu/~jurafsky/slp3/.

Sylvain Kahane, Alexis Nasr, and Owen Rambow. 1998. Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics*.

Najoung Kim and Tal Linzen. 2020. 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 (ACL).

Najoung Kim, Tal Linzen, and Paul Smolensky. 2022. Uncontrolled Lexical Exposure Leads to Overestimation of Compositional Generalization in Pretrained Models. *arXiv:2212.10769*.

Yoon Kim. 2021. Sequence-to-sequence learning with latent neural grammars. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, pages 26302–26317, Red Hook, NY, USA. Curran Associates Inc.

Marco Kuhlmann. 2013. Mildly non-projective dependency grammar. *Computational Linguistics*, 39(2):355–387.

Brenden M. Lake and Marco Baroni. 2018. Generalization without systematicity: On the Compositional Skills of Sequence-to-Sequence Recurrent Networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, pages 4487–4499, Stockholm, Sweden. Proceedings of Machine Learning Research (PMLR).

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

Bingzhi Li, Lucia Donatelli, Alexander Koller, Tal Linzen, Yuekun Yao, and Najoung Kim. 2023. SLOG: A Structural Generalization Benchmark for Semantic Parsing. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3213–3232, Singapore. Association for Computational Linguistics (ACL).

Chenyao Liu, Shengnan An, Zeqi Lin, Qian Liu, Bei Chen, Jian-Guang Lou, Lijie Wen, Nanning Zheng, and Dongmei Zhang. 2021. Learning Algebraic Recombination for Compositional Generalization. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1129–1144, Online. Association for Computational Linguistics (ACL).

Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. 2020. Compositional generalization by learning analytical expressions. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pages 11416–11427, Red Hook. Curran Associates Inc.

João Loula, Marco Baroni, and Brenden Lake. 2018. Rearranging the familiar: Testing compositional generalization in recurrent networks. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 108–114, Brussels, Belgium. Association for Computational Linguistics (ACL).

Igor A. Melčuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany, NY.

Joakim Nivre. 2006. Constraints on non-projective dependency parsing. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 73–80, Trento, Italy. Association for Computational Linguistics (ACL).

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Inbar Oren, Jonathan Herzig, Nitish Gupta, Matt Gardner, and Jonathan Berant. 2020. Improving

Compositional Generalization in Semantic Parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2482–2495, Online. Association for Computational Linguistics (ACL).

Barbara Partee. 2004. Compositionality. In *Compositionality in Formal Semantics: Selected Papers by Barbara H. Partee*. Blackwell Publishing, Malden.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Pawel Nowak, Tal Linzen, Fei Sha, and Kristina Toutanova. 2022. Improving Compositional Generalization with Latent Structure and Data Augmentation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4341–4362, Seattle, WA, USA. Association for Computational Linguistics (ACL).

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional Generalization and Natural Language Variation: Can a Semantic Parsing Approach Handle Both? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 922–938, Online. Association for Computational Linguistics (ACL).

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China. Association for Computational Linguistics (ACL).

Shawn Tan, Yikang Shen, Zhenfang Chen, Aaron Courville, and Chuang Gan. 2023. Sparse universal transformer. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 169–179, Singapore. Association for Computational Linguistics (ACL).

Stephen Tratz and Eduard Hovy. 2011. A fast, accurate, non-projective, semantically-enriched parser. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1257–1268, Edinburgh, Scotland, UK. Association for Computational Linguistics (ACL).

Antoine Venant and Alexander Koller. 2019. Semantic expressive capacity with bounded memory. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 65–79, Florence, Italy. Association for Computational Linguistics (ACL).

Pia Weißenhorn. 2022. Investigations on what makes compositional generalization challenging. Master's thesis, Saarland University.

Yuekun Yao and Alexander Koller. 2022. Structural generalization is hard for sequence-to-sequence models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5048–5062, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics (ACL).

Yang Yu. 2018. Towards Sample Efficient Reinforcement Learning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence Early Career*, pages 5739–5743. International Joint Conferences on Artificial Intelligence Organization (IJCAI).

Hao Zheng and Mirella Lapata. 2022. Disentangled Sequence to Sequence Learning for Compositional Generalization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4256–4268, Dublin, Ireland. Association for Computational Linguistics (ACL).

Wang Zhu, Peter Shaw, Tal Linzen, and Fei Sha. 2021. Learning to generalize compositionally by transferring across semantic parsing tasks. *arXiv:2111.05013*.

A. Appendix: Implementation Notes

This appendix contains notes on the implementation of the LEAR model as presented by Liu et al. (2021) and as modified in this study for the SLOG dataset, as well as on the implementation of the DEAR model.

A.1. Tai et al. (2015)'s Tree-LSTM in LEAR and DEAR

LEAR's *Composer* merges the hidden and cell states of selected nodes into those of a single parent node using Tai et al. (2015)'s N-ary Tree-LSTM. Nevertheless, their implementation of the Tree-LSTM cell differs from that of the original authors in a significant way.

Tai et al. define their cell transition equations for the kth child of node j as follows:

$$i_j = \sigma\Big(\sum_{\ell=1}^{N} U_{\ell}^{(i)}\, h_{j\ell} + b^{(i)}\Big), \tag{6}$$

$$f_{jk} = \sigma\Big(\sum_{\ell=1}^{N} U_{k\ell}^{(f)}\, h_{j\ell} + b^{(f)}\Big), \tag{7}$$

$$o_j = \sigma\Big(\sum_{\ell=1}^{N} U_{\ell}^{(o)}\, h_{j\ell} + b^{(o)}\Big), \tag{8}$$

$$u_j = \tanh\Big(\sum_{\ell=1}^{N} U_{\ell}^{(u)}\, h_{j\ell} + b^{(u)}\Big), \tag{9}$$

$$c_j = i_j \odot u_j \; + \sum_{\ell=1}^{N} f_{j\ell} \odot c_{j\ell}, \tag{10}$$

$$h_j = o_j \odot \tanh(c_j). \tag{11}$$

Liu et al. mostly maintain this architecture for LEAR's *Composer*, but crucially replace Equation (10) with the following:

$$c_j = \tanh\Big(i_j \odot u_j \; + \sum_{\ell=1}^{N} f_{j\ell} \odot c_{j\ell}\Big) \tag{12}$$

In developing the DEAR model, it was chosen to revert to the original formulation of c_j expressed in Equation (10). The reason for this is twofold.

Firstly, it was assumed to be an error on the part of LEAR's authors, given that the hyperbolic tangent is applied to c_j by invoking PyTorch's `tanh_` method rather than `tanh` when computing Equation (11). Because `tanh_` is an in-place operation, this results in the hyperbolic tangent being both used in the calculation of h_j *and* applied to the final value of c_j before being returned.

```
h = o * c.tanh_()
```
(13)

Secondly, this additional hyperbolic tangent was found to cause numerical instability in certain hyper-parameter configurations which particularly affected DEAR. Thus, it was chosen to modify this part of the cell to restore the behaviour described by Tai et al., by making use of the out-of-place `tanh`. This results in an identical value for h_j, so Equation (11) remains unaffected, but the value of c_j is no longer being computed using a hyperbolic tangent.

```
h = o * c.tanh()
```
(14)

A.2. Pre-Computed Tables

LEAR and DEAR make use of several pre-computed alignment tables that provide the models with dataset-specific information about tokens. Below is a categorised description of the tables in use, with statistics as of Liu et al. (2021)'s original implementation for the COGS dataset:

A.2.1. Lemmatisation:

The file `enct2dect` maps all *content words* found in the dataset to their corresponding lemmas. Given the simple nature of the dataset's vocabulary, the only content words whose lemma differs from their original form are predicates conjugated in the past tense, which are mapped to their unconjugated counterparts. There are a total of 731 tokens in this table, 591 of which have lemmas identical to their original form. There are 643 resulting unique lemmas, due to predicates in both their present and past forms being present.

A.2.2. Token Classification:

There are three tables that serve to classify tokens. These are `entity`, `unac_predicate`, and `caus_predicate`. Tokens can be classified as either an *entity* if they are on the `entity` table, or a *predicate*, if they are in either `unac_predicate` or `caus_predicate`. If a token is in none of the tables, it is deemed not to be a content word, and thus does not become a node during parsing. The `entity` table has 509 entries, `unac_predicate` has 21, and `caus_predicate` 113, for a total of 643 content words.

A.2.3. Indexing:

The file `encode_tokens` maps all tokens in the dataset to a unique, incremental ID based on their index in the table. It contains 741 entries, which account for all 731 content words plus 10 non-content words, which are: "*on*", "*by*", "*was*", "*beside*", "*the*", "*in*", "*that*", "*to*", "*a*", and ".".

In summary, there are 741 distinct tokens present in the dataset. 731 of them are considered content

words. These content words are lemmatised to 643 distinct values, 509 of which are classified as *entities* and 134 as *predicates*.

A.3. Pre–Computed Tables for SLOG

The alignment tables utilised by LEAR and DEAR, detailed in Appendix A.2, were expanded to encompass the additional vocabulary present in SLOG. An overview of the changes is shown below:

A.3.1. Entities:

4 new tokens were added to the entities list due to being present in SLOG but not in COGS. These consist of 2 regular entities *"gardener"* and *"monastery"*, alongside the 2 words introducing *wh*-questions, *who* and *what*. Note that the word *"gardener"* is present in the publicly available version of SLOG (Li et al., 2023) in the form *"gardner"*. This was manually changed to *"gardener"* in the data used for this study to avoid confusion.

A.3.2. Predicates:

All predicates present in SLOG are also present in COGS, so there are no additions to the `caus_predicate` and `unac_predicate` tables. However, 33 predicates that are only present in their past conjugation in COGS appear in their present conjugation in SLOG. These forms were correspondingly added to the lemmatisation table `enct2dect`. The list of such predicates can be seen below:

`dream, declare, pass, loan, say, award, mail, promise, prove, post, think, hope, sell, feed, confess, bring, lend, expect, believe, forward, give, send, slip, serve, wish, wire, rent, return, offer, hand, mean, support, imagine`

A.4. DEAR Post-processing

Once DEAR's *Composer* has produced a valid dependency tree, and its *Solver* has labelled all edges, the resulting labelled dependency tree lacks the edges corresponding to re-entrant entities, which can be deterministically derived from the existing labels. This is performed in two steps, *Relative Clause Reentrancy* and *Open Complement Reentrancy*, both of which are illustrated in Figure 5.

A.4.1. Relative Clause Reentrancy

Relative clauses are represented as an edge from the root *entity* to the *predicate* that modifies it. These relationships trigger an additional edge in the opposite direction, expressing the semantic role the *entity* plays in its *predicate*. While this edge is created manually, based on the existence of the original edge labelled as a relative clause (`relcl`), it is labelled by applying the same MLP mechanism as any other *predicate→entity* edge.

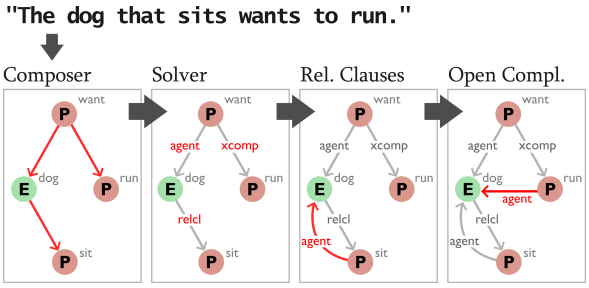

Figure 5: Diagram illustrating DEAR's two post-processing steps. From left to right, the *Composer* outputs a dependency tree, the *Solver* labels edges with semantic relations, and the post-processing steps (*Rel. Clauses* and *Open Compl.*) deterministically add the extra edges required to match the target representation. In this example sentence, the entity *"dog"* is the agent of all three predicates.

A.4.2. Open Complement Reentrancy

Predicates positioned as arguments of other predicates can be labelled as *open clausal complements* (`xcomp`), in which case both share an agent. Given the existence of such a relation in the *Composer*'s output, an additional edge is inserted to express this agent reentrancy.

A.5. Search Space Growth by Sentence Length

As discussed in Section 6, LEAR and DEAR's syntactic search spaces, the number of possible trees they could construct, grow with the sentence length (the number of content words), in different ways. LEAR's *Composer* follows the Catalan sequence, detailed below

$$P_{\text{LeAR}}(n) = \frac{(2n)!}{(n+1)!n!} \sim \frac{4^n}{n^{\frac{3}{2}}\sqrt{\pi}} \tag{15}$$

Whereas DEAR's follows Cayley's formula, as below:

$$P_{\text{DeAR}}(n) = n^{n-2} \tag{16}$$

Table 3 compares the corresponding amount of possible trees for sentences of lengths commonly found in the SLOG dataset.

B. Appendix: LEAR's Architecture

An overview of LEAR's architecture, and an example input and output, is shown in Figure 6.

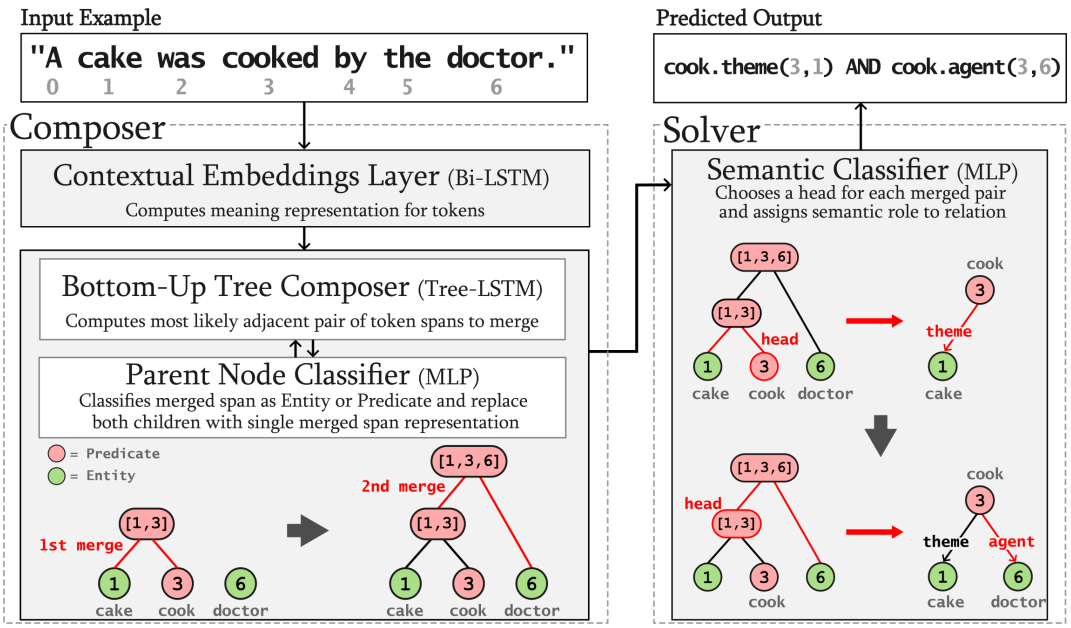

Figure 6: Overview of LEAR's architecture, showcasing an example from COGS. The input sentence (top-left) gets fed into the *Composer* module (left), which creates a constituency tree. Then the *Solver* (right) predicts a dependency structure from each constituent, and produces an output (top-right).

Length	LEAR	DEAR
2	1	1
3	2	3
4	5	16
5	14	125
6	42	1,296
7	132	16,807
8	429	262,144
9	1,430	4,782,969
10	4,862	100,000,000
11	16,796	2,357,947,691
12	58,786	61,917,364,224
13	208,012	1,792,160,394,037
14	742,900	56,693,912,375,296

Table 3: Comparison of LEAR and DEAR's search spaces across sentence lengths.

C. Appendix: Training Notes

C.1. Hyper-parameters

Due to constrained resources, a systematic hyper-parameter search was not performed in this study. Additionally, reported results were produced by a single training run, so a seed-sensitivity analysis remains for future work. The architectural differences between the LEAR and DEAR models, and DEAR's increased complexity in particular, make it unsuitable to compare them using identical hyper-parameters. Specifically, LEAR's learning rate results in numerical instability when applied to DEAR's larger architecture, so a lower one was chosen. Additionally, DEAR's increased search-space benefits from a higher number of samples in its REINFORCE loop. Table 4 below shows a list of the models' principal hyper-parameters, alongside a description for each.

The hyper-parameters used to fine-tune the LLMs in this study, Llama3 8b and Mistral 7b, are shown below.

- **LoRA configuration:** rank $r = 16$, $\alpha = 32$, dropout $= 0.05$, bias = none
- **Target modules:** `[q_proj, k_proj, v_proj, o_proj, up_proj, down_proj, gate_proj]`
- **Quantization:** 4-bit NF4 with double quantization; compute dtype = `bfloat16`/`float16` depending on GPU support
- **Optimiser:** `paged_adamw_8bit`
- **Learning rate:** 5×10^{-5} with cosine scheduler and warmup ratio $= 0.03$
- **Batch size:** 1 (train), 2 (evaluation)
- **Gradient accumulation steps:** 16
- **Epochs:** 1
- **Precision:** `fp16` enabled, `bf16` disabled
- **Gradient checkpointing:** enabled
- **Sequence length:** 512 tokens
- **Evaluation interval:** every 500 steps
- **Logging interval:** every 10 steps
- **Scheduler warmup ratio:** 0.03
- **Inference parameters:** batch size = 128, maximum generation length = 1024 tokens

Hyper-parameter	Description	LEAR	DEAR
Hidden Layer Size	Vector size of token embeddings, LSTM states, and MLP inputs.	128	256
High Learning Rate	Learning rate for the *Composer* parameters, including embeddings.	1.0	0.1
Low Learning Rate	Learning rate for the *Solver* parameters.	0.5	0.05
Batch Size	Batch Size for gradient accumulation.	1	8
Number of Samples	Number of Monte-Carlo samples explored for each sentence.	10	15
Epochs	Number of training epochs during training.	1	3
Optimisation Method	Stochastic Gradient Descent algorithm employed during training.	AdaDelta	AdaDelta

Table 4: Hyper-parameters chosen for the LEAR and DEAR models.

C.2. Resource and Time Requirements

All training for both LEAR and DEAR was performed on an `AMD Ryzen 7 5825U` CPU, at 2GHz, running as a single-threaded process. Training time was found to scale mostly linearly with the number of parameters, determined by the *Hidden Layer Size* hyper-parameter, the number of Monte–Carlo samples performed, and the number of epochs needed, which was affected by learning rates. Whereas LEAR achieved its maximum performance on epoch 1, over-fitting afterwards, DEAR required at least 3 epochs to adequately train. Table 5 below shows a summary of training times for LEAR and DEAR.

While the increased training time exhibited by DEAR as compared to LEAR is inherent to its architecture and the chosen hyper-parameters (see Appendix C.1), it is likely not reflective of parallelised GPU performance. This is because the increased Monte-Carlo samples, as well as the larger hidden vector size, would not scale the computational expense to the same degree.

Fine-tuning of Llama3 8b and Mistral 7b was done on an A100 GPU, with a training duration of approximately 5 hours each.

D. Appendix: Dataset Categories

Below are all the categories in the generalisation set for both COGS and SLOG, alongside its identifier within the dataset (for reference when working with the code published alongside this paper), a description of the category, and an example taken from the original authors' publications (Kim and Linzen, 2020; Li et al., 2023).

D.1. COGS: Lexical Recombination Categories

D.1.1. Subject → Object (common noun):

(ID: `subj_to_obj_common`) Entities consisting of common nouns only seen in subject positions during training, presented in object positions in the generalisation set.
Train.: *A **hedgehog** ate the cake.*
Gen.: *The baby liked the **hedgehog**.*

D.1.2. Subject → Object (proper noun):

(ID: `subj_to_obj_proper`) Entities consisting of proper nouns only seen in subject positions during training, presented in object positions in the generalisation set.
Train.: ***Lina** gave the cake to Olivia.*
Gen.: *A hero shortened **Lina**.*

D.1.3. Object → Subject (common noun):

(ID: `obj_to_subj_common`) Entities consisting of common nouns only seen in object positions during training, presented in subject positions in the generalisation set.
Train.: *Henry liked a **cockroach**.*
Gen.: *The **cockroach** ate the bat.*

Model/Dataset	Mean Epoch Time (minutes)	# Epochs	Total Time (minutes)
LEAR (SLOG)	73.4	1	73.4
DEAR (SLOG)	394.7	3	1184.2

Table 5: Training times, in minutes, for the LEAR and DEAR models on the SLOG dataset.

D.1.4. Object → Subject (proper noun):

(ID: `obj_to_subj_proper`) Entities consisting of proper nouns only seen in object positions during training, presented in subject positions in the generalisation set.
Train.: *The creature grew **Charlie**.*
Gen.: ***Charlie** worshipped the cake.*

D.1.5. Primitive noun → Subject (common noun):

(ID: `prim_to_subj_common`) Common noun entity that only appears as a stand-alone primitive in training, and is presented as the subject in the generalisation set.
Train.: ***shark***
Gen.: *A **shark** examined the child.*

D.1.6. Primitive noun → Subject (proper noun):

(ID: `prim_to_subj_proper`) Proper noun entity that only appears as a stand-alone primitive in training, and is presented as the subject in the generalisation set.
Train.: ***Paula***
Gen.: ***Paula** sketched William.*

D.1.7. Primitive noun → Object (common noun):

(ID: `primn_to_obj_common`) Common noun entity that only appears as a standalone primitive in training, and is presented as the object in the generalisation set.
Train.: ***shark***
Gen.: *A chief heard the **shark**.*

D.1.8. Primitive noun → Object (proper noun):

(ID: `primn_to_obj_proper`) Proper noun entity that only appears as a standalone primitive in training, and is presented as the object in the generalisation set.
Train.: ***Paula***
Gen.: *The child helped **Paula**.*

D.1.9. Primitive verb → Infinitival argument:

(ID: `prim_to_inf_arg`) Verb that only appears as a stand-alone primitive in training, and is presented in an infinitival open clausal complement in the generalisation set.
Train.: ***crawl***
Gen.: *A baby planned to **crawl**.*

D.1.10. Active → Passive:

(ID: `active_to_passive`) Verbs used in active constructions during training appear in passive constructions in the generalisation set.
Train.: *The crocodile **blessed** William.*
Gen.: *A muffin **was blessed**.*

D.1.11. Passive → Active:

(ID: `passive_to_active`) Verbs used in passive constructions during training appear in active constructions in the generalisation set.
Train.: *The book **was squeezed**.*
Gen.: *The girl **squeezed** the strawberry.*

D.1.12. Object-omitted transitive → Transitive:

(ID: `obj_omitted_transitive _to_transitive`) Verbs seen with omitted objects during training appear with explicit objects in the generalisation set.
Train.: *Emily **baked**.*
Gen.: *The giraffe **baked a cake**.*

D.1.13. Unaccusative → Transitive:

(ID: `unacc_to_transitive`) Verbs displaying causative alternation are used in their intransitive form during training and in their transitive form in the generalisation set.
Train.: *The glass **shattered**.*
Gen.: *Liam **shattered** the jigsaw.*

D.1.14. Double object dative → PP dative:

(ID: `do_dative_to_pp_dative`) Verbs displaying dative alternation are shown in their double object dative form during training and in prepositional dative form in the generalisation set.
Train.: *The girl **teleported Liam the cookie**.*
Gen.: *Benjamin **teleported the cake to Isabella**.*

D.1.15. PP dative → Double Object Dative:

(ID: pp_dative_to_do_dative) Verbs displaying dative alternation are shown in their prepositional dative form during training and in double object dative form in the generalisation set.
Train.: *Jane **shipped the cake to John**.*
Gen.: *Jane **shipped John the cake**.*

D.1.16. Agent NP → Unaccusative subject:

(ID: only_seen_as_transitive_subj_as_unacc_subj) Entities seen as agentive subjects in training are presented as theme subjects in non-causative predicates in the generalisation set.
Train.: *The **cobra** helped a dog.*
Gen.: *The **cobra** froze.*

D.1.17. Theme NP → Object-omitted transitive subject:

(ID: only_seen_as_unacc_subj_as_obj_omitted_transitive_subj) Entities seen as theme subjects in non-causative predicates in training are presented as agentive subjects of transitive predicates without an explicit theme in the generalisation set.
Train.: *The **hippo** decomposed.*
Gen.: *The **hippo** painted.*

D.1.18. Theme NP → Unergative subject:

(ID: only_seen_as_unacc_subj_as_ unerg_subj) Entities seen as theme subjects in non-causative predicates in training are presented as agentive subjects of intransitive predicates in the generalisation set.
Train.: *The **hippo** decomposed.*
Gen.: *The **hippo** giggled.*

D.2. COGS: Structural Recombination Categories

D.2.1. Object modification → Subject modification:

(ID: obj_pp_to_subj_pp) Object noun phrases containing prepositional phrases in training appear instead as subject noun phrases with similar PPs in the generalisation set.
Train.: *Noah ate the **cake on the plate**.*
Gen.: *The **cake on the table** burned.*

D.2.2. Depth generalization: Sentential complements:

(ID: cp_recursion) Sentences displaying deeper nested sentential complements in the generalisation set than in the training set.
Train.: *Emma said that Noah knew that the cat danced.*
Gen.: *Emma said that Noah knew that Lucas saw that the cat danced.*

D.2.3. Depth generalization: PP modifiers:

(ID: pp_recursion) Sentences displaying deeper nested prepositional phrases in the generalisation set than in the training set.
Train.: *Ava saw the ball **in the bottle on the table**.*
Gen.: *Ava saw the ball **in the bottle on the table on the floor**.*

D.3. SLOG

D.3.1. Prepositional phrases (PP) max depth 4 → depth 5–12

(ID: PP_5-12) Prepositional phrase nesting seen to a depth of 4 in training is extended to depths 5–12 in the generalisation set.
Train.: *Ava saw the ball **in the bottle on the table**.*
Gen.: *Ava saw the **cat in the box on the mat on the bed on the floor in the room**.*

D.3.2. Tail CP recursion max depth 4 → depth 5–12

(ID: CP_5-12) Recursive sentential complements (CPs) nested to a depth of 4 in training are extended to 5–12 in the generalisation set.
Train.: *Ava believed that Emma said that a fish froze.*
Gen.: *Ava said that Emma liked that Max believed that Noah found that Liam saw that the cat slept.*

D.3.3. Center embedding max depth 4 → depth 5–12

(ID: center_embed_5-12) Center-embedded relative clauses nested to a depth of 4 in training are extended to 5–12 in the generalisation set.
Train.: *Eva saw the **cat that the horse that the dog liked** chased.*
Gen.: *Ava held the dress that a store that a girl that a boy that a cat that a man drew saw loved knew sold.*

D.3.4. PP max depth 4 → depth 3

(ID: PP_3) Prepositional phrase nesting seen to a depth of 4 in training is reduced to depth 3 in the generalisation set.
Train.: *Emma saw the **ball in the bottle on the table on the floor in the office**.*
Gen.: *Ava saw the **cat on the mat on the floor in the office**.*

D.3.5. Tail CP recursion max depth 4 → depth 3

(ID: `CP_3`) Tail-recursive CPs trained at depth 4, tested at shallower depth 3.
Train.: *Ava believed that Emma said that Max found that a cat saw that a fish froze.*
Gen.: *Ava said that Emma liked that Max believed that the cat slept.*

D.3.6. Center embedding max depth 4 → depth 3

(ID: `center_embed_3`) Center embedded RCs trained at a depth of 4 are presented at a depth of 3 in the generalisation set.
Train.: *Eva saw the cat that the horse that the dog that the man that the girl loved found* liked chased.
Gen.: *Emma bought the dress that the store that the woman that Mike knew liked sold.*

D.3.7. PP in direct object NPs → PP in subject NPs

(ID: `PP_modif_subj`) Prepositional phrase modifiers that appear in direct object noun phrases during training are presented in subject noun phrases in the generalisation set.
Train.: *Noah ate the **cake on the plate**.*
Gen.: *The **cake on the table** burned.*

D.3.8. PP in direct object NPs → PP in indirect object NPs

(ID: `PP_modif_iobj`) Prepositional phrase modifiers that appear in direct objects during training are presented in indirect objects in the generalisation set.
Train.: *Noah ate the **cake on the plate**.*
Gen.: *Max gave a fish to a **cat on a table**.*

D.3.9. RC in direct object NPs → RC in subject NPs

(ID: `RC_modif_subj`) Relative clauses modifying direct objects during training are presented modifying subject noun phrases.
Train.: *Noah saw the **cat that froze**.*
Gen.: *The **cat that froze** smiled.*

D.3.10. RC in direct object NPs → RC in indirect object NPs

(ID: `RC_modif_iobj`) Relative clauses modifying direct objects during training are presented modifying indirect object noun phrases.
Train.: *Noah saw the **cat that froze**.*
Gen.: *Max gave a fish to a **cat that ran**.*

D.3.11. Subject, direct object-extracted RC → Indirect object-extracted RC

(ID: `RC_iobj_extracted`) Relative clauses introduce subjects and direct objects in training, the generalisation set presents relative clauses introducing indirect objects.
Train.: *Noah saw the cat that gave a fish to Liam*. | Noah saw the cat that Liam liked _.
Gen.: *Noah saw the cat that Emma gave a cake to _*.

D.3.12. Subject, direct object *wh*-questions → Indirect object *wh*-questions

(ID: `Q_iobj_ditransV`) Generalisation from subject *wh*-questions to ones interrogating indirect object arguments.
Train.: *Who saw the cat?* | *What did Emma see _?*
Gen.: *Who did Noah give the cake to _?*

D.3.13. Subject, object *wh*-q of simple transitives → Active subject *wh*-questions

(ID: `Q_subj_active`) Transitive predicates seen in training in non-interrogative contexts are presented in the generalisation set as part of *wh*-questions interrogating the active subject.
Train.: ***Who saw** the cat?* | *Emma **wanted** to sleep.*
Gen.: ***Who wanted** to sleep?*

D.3.14. Subject, object *wh*-q of simple transitives → Passive subject *wh*-questions

(ID: `Q_subj_passive`) Transitive predicates seen in training in non-interrogative contexts are presented in the generalisation set as part of *wh*-questions interrogating the passive subject.
Train.: ***Who** did Emma see _?* | *The boy **was found** by Emma*
Gen.: ***Who was helped** by Emma?*

D.3.15. Subject, object *wh*-q of simple transitives → Direct object *wh*-questions with ditransitive verbs

(ID: `Q_dobj_ditransV`) Generalisation from transitive *wh*-questions in training to ditransitive ones in the generalisation set.
Train.: ***What** did Emma **see** _?* | *Emma **gave** a fish **to** the cat*
Gen.: ***What** did Emma **give** _ to the cat?*

D.3.16. Subject, object *wh*-q of simple transitives → *Wh*-questions with modified NPs

(ID: `Q_modified_NPs`) Generalisation from *wh*-questions with simple nouns in training to ones with modifying PPs in the generalisation set.
Train.: ***What*** *did* ***the cat*** *see _?* | *the cat* ***on the mat***
Gen.: *What did* ***the cat on a table*** *see _?*

D.3.17. Subject, object *wh*-q of simple transitives → *Wh*-questions long movement

(ID: `Q_long_mv`) *Wh*-dependencies span longer distances than in training, testing generalisation to long-distance movement.
Train.: ***What*** *did the cat* ***see*** *_?* | *Emma* ***said that*** *the cat saw a fish.*
Gen.: ***What*** *did Emma* ***say that*** *the cat* ***found*** *_?*