Compositional Generalization Across Distributional Shifts with Sparse Tree Operations

Paul Soulos[*]
Johns Hopkins University
psoulos1@jh.edu

Henry Conklin[*]
University of Edinburgh

Mattia Opper[*]
University of Edinburgh

Paul Smolensky
Johns Hopkins University
Microsoft Research

Jianfeng Gao
Microsoft Research

Roland Fernandez
Microsoft Research

Abstract

Neural networks continue to struggle with compositional generalization, and this issue is exacerbated by a lack of massive pre-training. One successful approach for developing neural systems which exhibit human-like compositional generalization is *hybrid* neurosymbolic techniques. However, these techniques run into the core issues that plague symbolic approaches to AI: scalability and flexibility. The reason for this failure is that at their core, hybrid neurosymbolic models perform symbolic computation and relegate the scalable and flexible neural computation to parameterizing a symbolic system. We investigate a *unified* neurosymbolic system where transformations in the network can be interpreted simultaneously as both symbolic and neural computation. We extend a unified neurosymbolic architecture called the Differentiable Tree Machine in two central ways. First, we significantly increase the model's efficiency through the use of sparse vector representations of symbolic structures. Second, we enable its application beyond the restricted set of tree2tree problems to the more general class of seq2seq problems. The improved model retains its prior generalization capabilities and, since there is a fully neural path through the network, avoids the pitfalls of other neurosymbolic techniques that elevate symbolic computation over neural computation.

1 Introduction

Note: this is an abbreviated version simultaneously published at the NeurIPS 2024 main conference. Please see that version for the full details. Deep learning models achieve remarkable performance across a broad range of natural language tasks [65], despite having difficulty generalizing outside of their training data, struggling with new words [34], known words in new contexts [27], and novel syntactic structures, like longer sequences with greater recursive depth [28, 37]. Increasingly this problem is addressed through data augmentation, which tries to make it less likely a model will encounter something unlike what it sees during training — reducing the degree by which it has to gener-

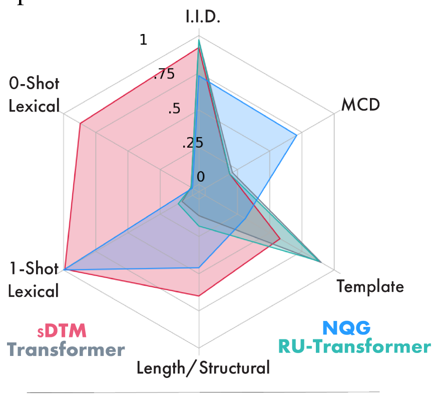

Figure 1: Generalization ability of our approach (sDTM) compared with baselines across various out-of-distribution shifts, averaged over different datasets. See §4.

[*]Work partially completed while at Microsoft Research.

alize [1, 13, 24]. However, even models trained on vast quantities of data struggle when evaluated on examples unlike those seen during training [30].

This stands in contrast to how humans process language, which enables robust generalization [47]. By breaking novel sentences into known parts, we can readily interpret phrases and constructions that we have never encountered before (e.g. 'At the airport I smiled myself an upgrade', [20]). Why do models trained on orders of magnitude more language data than a human hears in 200 lifetimes [23] still fail to acquire some of language's most essential properties?

Central to language's generalizability is compositional structure [45] where contentful units, like words, fit together in a structure, like a syntactic tree. Many classical approaches in NLP and Machine Learning attempt to induce a grammar from data in the hope of leveraging the same kinds of generalization seen in natural language [e.g. 32, 31, 62]. However, structured representations are not first-order primitives in most neural networks [41, 57]. Despite theoretical appeal, the strictures of purely discrete symbolic approaches have made them difficult to apply to the breadth of tasks and domains where deep learning models have proven successful [16]. In contrast, purely connectionist models — like Transformers [65] — struggle with the kinds of sample efficiency and robust generalization ubiquitous to human learning.

Neurosymbolic methods attempt to integrate neural and symbolic techniques to arrive at a system that is both compositional and flexible [3, 17, 18, 57]. While some neurosymbolic architectures achieve impressive compositional generalization, they are often brittle due to the symbolic core of their computation [53]. These methods are hybrid neurosymbolic systems, where the primary computation is symbolic, and the neural network serves to parameterize the symbolic space. We take a different approach, one where symbolic operations happen in vector space. In our system, neural and symbolic computations are **unified** into a single space; we multiply and add vector-embedded symbolic structures instead of multiplying and adding individual neurons. Related Work is discussed in Appendix A.1.

We introduce a new technique for representing trees in vector space called Sparse Coordinate Trees (SCT). SCT allows us to perform structural operations: transformations which change the structure of an object without changing the content. This is a crucial aspect of compositionality, where the structure and content can be transformed independently. We extend a previous system which operates over binary trees in vector space, the Differentiable Tree Machine (`DTM`), to improve performance and applicability to a larger variety of tasks[2]. While `DTM` processes vector-embedded binary trees as the primitive unit of computation, the order of operations and argument selection is governed by a Transformer. We present results showing that this unified approach retains many of the desirable properties of more brittle symbolic models with regards to generalization, while remaining flexible enough to work across a far wider set of tasks. While fully neural architectures or hybrid neurosymbolic techniques excel at certain types of generalization, we find that `DTM`, with its unified approach, excels across the widest array of shifts.

The main contributions from this paper are:

- Sparse Coordinate Trees (SCT), a method for representing binary trees in vector space. (§2)
- Bit-Shift Operating — systematic and parallelized tree operations for SCT. (§2.1)
- The introduction of Sparse Differentiable Tree Machine (`sDTM`), architectural improvements to the `DTM` to leverage SCT and drastically reduce parameter and memory usage. (§3)
- Techniques to apply `DTM` to seq2seq tasks by converting sequences into trees. (§3.5)
- Empirical comparisons between `sDTM` and various baselines showing `sDTM`'s strong generalization across a wide variety of tasks. (§4)

2 Differentiable Tree Operations Over Sparse Coordinate Trees

Representing trees in vector space enables us to perform differentiable structural operations on them. Soulos et al. [61] used Tensor Product Representations (TPRs) [58] for this purpose. TPRs use the tensor (or outer) product to represent trees in vector space (§A.2). Use of an outer product leads to a representation dimensionality that is multiplicative with both the embedding dimensionality and the

[2]Code and data available at publication.

number of possible tree nodes. Additionally, the number of nodes is itself an exponential function of the supported depth. This makes TPRs difficult to use in practice, given available memory is quickly exceeded as tree depth increases.

In this section, we introduce Sparse Coordinate Trees (SCT), a new schema for representing trees in vector space. We then define a library of parallelized tree operations and how to perform these operations on SCT.

Like TPRs, we want an encoding for trees that factorizes the representation into subspaces for *structure* and *content* respectively. This approach to representational spaces differs from models like an RNN and Transformer, which represent structure and content jointly in an unfactorized manner. By separating the structure and content subspaces a priori, we can operate over these two spaces independently. This decision is motivated by the fact that distinct treatment of these spaces is an essential aspect of compositionality.

We derive our tree representation scheme from the sparse coordinate list (COO) format. COO stores tensor data in tuples of (indices [integers], values [any format], size [integers]). The indices are N-dimensional to simulate a tensor of arbitrary shape (e.g. including dimensions such as batch or length). When an index is not indicated in indices, it is assumed that the corresponding value is 0.

We give structural meaning to COO representations by defining one dimension of indices as the tree position occupied by a value vector. Our tree addressing scheme is based on Gorn addresses [21]: to get the tree position from an index, convert the index to binary and read from right to left. A left-branch is indicated by a 0 and a right branch by a 1. To distinguish between leading 0s and left-branches (e.g. 010 vs 10), we start our addressing scheme at 1 instead of 0. This indicates that all 0s to the left of the most-significant 1 are unfilled and not left-branches. Figure 2 shows an example encoding of a tree with this approach. SCT can be viewed as a TPR with certain constraints, and Section A.2 defines this equivalence and describes the memory savings.

Section 4.1 discusses the performance, memory, and parameter comparison between DTM models which use TPRs and SCT.

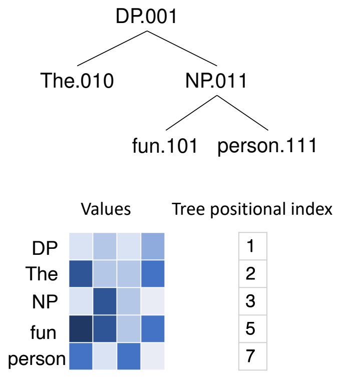

Figure 2: An example representation using Sparse Coordinate Trees (SCT). The values are N-dimensional vectors, and the tree positional indices are integer representations of positions in the tree. The absent child nodes of "The" (indices 4 and 6) are skipped with SCT.

2.1 Differentiable Tree Operations

To operate on the trees defined in the previous section, we need a set of functions. We use a small library of only three: left-child (`left`), right-child (`right`), and **cons**truct (`cons`) a new tree from a left and right subtree.[3] Although these three functions are simple, along with the control operations of conditional branching and equality-checking, these five functions are Turing complete [42].

In addition to saving memory, SCT also provides a more efficient method for performing differentiable tree operations. The operations defined in Soulos et al. [61] require precomputing, storing, and applying linear transformations for `left`, `right`, and `cons`. Since our values and tree positional indices are kept separate, we can compute the results of `left`, `right`, and `cons` dramatically more efficiently using indexing, bit-shifts, and addition.

Figure 3 shows how we can perform `left` directly on SCT. `left` is performed by indexing the even indices (i.e. those with a 0 in the least significant bit, which targets all of the nodes left of the root) and their corresponding values, then performing a right bit-shift on the indices. `right` is symmetrical, except that we index for the odd positional indices and ignore position 1 in order to remove the previous root node. `cons` is performed by left bit-shifting the positional indices from the left- and right-subtree arguments, then adding 1 to the newly shifted indices for the right argument. A new value s can be provided for the root node.

[3]In LISP and expert systems literature, `left` is referred to as `car`, and `right` is referred to as `cdr`.

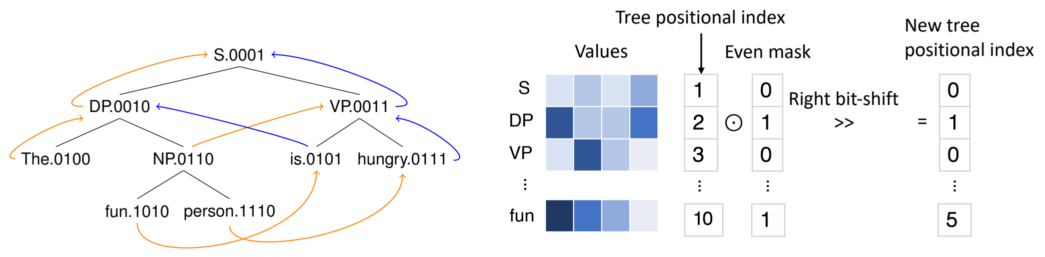

Figure 3: Left: Performing `left` (orange) and `right` (blue). Right: visualizing the `left` transformation which results in DP being placed at the root. Tree positional indices of 0 and their corresponding values are discarded.

Our network also needs to learn a *program* over multiple operations differentiably. This involves the aforementioned structured operations, as well as differentiable selection of which operation to perform and on which trees. We take weighted sums over the three operations `left`, `right`, and `cons`, as well as over potential trees. Specific details are discussed in the next section. The result of our weighted sum is coalesced, which removes duplicate positional indices by summing together all of the values that share a specific index. Formally, define the trees over which to perform `left` T_L, `right` T_R, and `cons` T_{CL} & T_{CR}; $\vec{T} = [T_L; T_R; T_{CL}; T_{CR}]$. We also take a new value $s \in \mathbb{R}^d$ to be inserted ($\otimes$) at the new root node of the `cons` operation, and a vector of operation weights $\vec{w} = (w_L, w_R, w_C)$ which sum to 1.

$$O(\vec{w}, \vec{T}, s) = w_L \texttt{left}(T_L) + w_R \texttt{right}(T_R) + w_C(\texttt{cons}(T_{CL}, T_{CR}) + s \otimes r_1) \tag{1}$$

3 The Sparse Differentiable Tree Machine (`sDTM`)

Our work extends the Differentiable Tree Machine (`DTM`) introduced in Soulos et al. [61] with the Sparse Differentiable Tree Machine (`sDTM`). While similar to the original at a computational level, `sDTM` represents a different implementation of these concepts that make it dramatically more parameter and memory efficient. We also introduce techniques to apply `sDTM` to tasks with sequence input and output (seq2seq).

3.1 Overview

`sDTM` uses our Sparse Coordinate Trees schema across its components. Like the original `DTM`, our model is comprised of an agent, interpreter, and memory (illustrated in Figure 4). The Interpreter performs Equation 1 by applying the bit-shifting tree operations from Section 2.1 and weighting the result. The output from the interpreter is written to the next available memory slot, and the last memory slot is taken as the output.

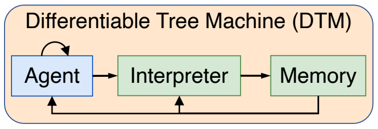

Figure 4: A schematic of how the three core components of the `DTM` (agent, interpreter, and memory) relate to each other. Adapted from Soulos et al. [61].

The Agent is a Transformer encoder that takes an encoding of the memory as input and produces the inputs for Equation 1: $\vec{w}$, $\vec{T}$, and s. Two special tokens, <OP> and <ROOT>, are fed into the Agent to represent $\vec{w}$ and s. Each time a tree is written to memory, a fixed-dimensional encoding of that tree is produced and fed as a new token to the agent (§3.2). The agent soft-selects tree arguments for the interpreter, $\vec{T}$, by performing a weighted sum over the trees in memory. Figure 6 in the Appendix contains a detailed diagram showing the flow of information for one layer of `sDTM`.

3.2 Pooling by attention

Each tree in memory needs to have a fixed-dimensional encoding to feed into the agent regardless of how many nodes are filled. Commonly this is done via pooling, like taking the means of the elements

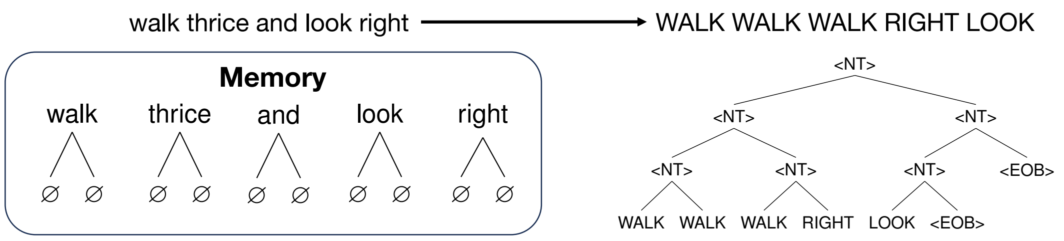
walk thrice and look right
WALK WALK WALK RIGHT LOOK
Memory
walk
thrice
and
look
right
∅ ∅
<NT>
<EOB>
WALK WALK WALK RIGHT LOOK <EOB>

Figure 5: Left: The memory state is initialized as a sequence of trees where only the root node contains a token. Right: An output sequence is embedded in a tree using the left-aligned uniform-depth (LAUD) scheme. <NT> and <EOB> are special tokens not in the original output sequence.

in the tree, or a linear transformation in the case of the original DTM. Instead, we use Pooling by Multi-headed Attention (PMA) [36], which performs a weighted sum over the elements, where the weight is derived based on query-key attention.

Attention is permutation invariant, but it is important that our pooling considers tree position information. To enforce this, we convert the position indices to their binary vector representation $\vec{b}$. This leads to an asymmetrical vector with only positive values, so instead we represent left branches as -1 and keep right branches as $+1$. For example, position $5 \rightarrow [0,0,0,0,0,1,0,1] \rightarrow [0,0,0,0,0,1,-1,1]$. The input to our pooling function is the concatenation of this positional encoding $\vec{b}$ with the token embedding $\vec{x}$ at that position: $[\vec{x};\vec{b}]$. This method for integrating token and node position is similar to tree positional encoding from Shiv and Quirk [54], except that we use concatenation and a linear transformation to mix the content and position information instead of addition.

Unlike standard self attention, we use a separate, single learnable parameter for our query vector $\vec{q} \in \mathbb{R}^{\text{num\_heads} \times \text{key\_dim}}$. We pass $[\vec{x};\vec{b}]$ through linear transformations to generate keys $\vec{k} \in \mathbb{R}^{\text{num\_heads} \times \text{key\_dim}}$ and values $\vec{v} \in \mathbb{R}^{\text{num\_heads} \times \text{value\_dim}}$. The result of this computation is always $z \in \mathbb{R}^{\text{num\_heads} \times \text{value\_dim}}$ given that $\vec{q}$ is fixed and does not depend on the input. The rest of the computation is identical to a Transformer with pre-layer normalization [67].

3.3 Tree Pruning

While Sparse Coordinate Trees mean that trees with fewer filled nodes take up less memory, the way our model blends operations results in trees becoming dense. The interpreter returns a blend of all three operations at each step, including the cons operation which increases the size of the representation by combining two trees. In practice even as the entropy of the blending distribution drops, the probability of any operation never becomes fully 0. This means that over many steps, trees start to become dense due to repeated use of cons. In order to keep our trees sparse, we use pruning: only keeping the top-k nodes as measured by magnitude. k is a hyper-parameter that can be set along with the batch size depending on available memory.

3.4 Lexical Regularization

To aid lexical generalization, we add noise to our token embeddings. Before feeding an embedded batch into the model, we sample from a multi-variate standard normal for each position in each tree, adding the noise to the embeddings as a form of regularization [4]. Ablation results showing the importance of this regularization are available in Appendix A.4.

3.5 Handling Sequential Inputs and Outputs

seq2tree The original DTM can only be applied to tasks where a tree structure is known for both inputs and outputs. Here we provide an extension to allow DTM to process sequence inputs. To do this we treat each input token as a tree with only the root node occupied by the token embedding. We then initialize the tree memory with N trees, one for each token in the input sequence. Figure 5 left depicts the initial memory state for a sequence. The agent's attention mechanism is permutation-invariant, so in order to distinguish between two sequences which contain the same tokens but in different orders, we apply random sinusoidal positional encodings to the first N tokens passed to the agent [38, 50]. Random positional encodings sample a set of increasing integers from left-to-right instead of assigning a fixed position to each token. The purpose of `left` and `right` is to extract subtrees. Since in our seq2tree setting the input sequence is processed in a completely bottom-up manner, we restrict the agent and interpreter to only have a single operation: `cons`. Use of a single operation to construct new trees from subtrees aligns the `DTM` theoretically with the Minimalist Program [9], which addresses natural language's compositionality in terms of a single operation: merge.

seq2seq To handle sequence inputs and outputs we convert the output sequence to a tree. One method to convert the output sequence into a tree is to use a parser. Alternatively, when a parser is not available, we can embed a sequence as the **l**eft-**a**ligned leaves at **u**niform **d**epth (LAUD). Figure 5 right shows how an output sequence can be embedded using LAUD. Since all of the non-terminal nodes are the same, we can hardcode the root argument to `cons`. We insert a special token <EOB> to signify the end of a branch, similar to an <EOS> token.

4 Results

We consider models that are trained from scratch on the datasets they're evaluated on; while the compositional capabilities of large pre-trained models are under active debate [30], we are interested in the compositional abilities of the underlying architecture — rather than those that may result from a pre-training objective. We compare `DTM` to two fully neural models, a standard Transformer [65] as well as a relative universal Transformer (RU-Transformer) which was previously shown to improve systematic generalization on a variety of tasks [11]. We also compare our model to a hybrid neurosymbolic system, NQG [53, 63], a model which uses a neural network to learn a quasi-synchronous context-free grammar [56]. NQG was introduced alongside NQG-T5, which is a modular system that uses NQG when the grammar produces an answer and falls back to a fine-tuned large language model T5 [49]. As mentioned at the beginning of this section, we only compare to NQG in this paper since we want to evaluate models that have not undergone significant pre-training.[4] Details related to data preprocessing (§A.5), model training (§A.7, §A.8), and compute resources (§A.9) are available in the Appendix.

For each task, we test whether models generalize to samples drawn from various data distributions. Independent and identically distributed (**IID**) samples are drawn from a distribution shared with training data. We evaluate several out-of-distribution (OOD) shifts. **One-shot** lexical samples, while drawn from the same distribution as the training data, contain a word that was only seen in a single training sample. Similarly, **Zero-shot** lexical samples are those where the model is not exposed to a word at all during training. **Structural/length** generalization tests whether models can generalize to longer sequences (length) or nodes not encountered during training (structural). **Template** generalization withholds an abstract n-gram sequence during training, and then each test sample fits the template. Finally, maximum compound divergence (**MCD**) generates train and test sets with identical uni-gram distributions but maximally divergent n-grams frequencies [27]. Although models are often tested on a single type of generalization, we believe evaluating a model across a broad array of distributional shifts is essential for characterizing the robustness of its generalization performance.

4.1 Performance Regression (Active$\leftrightarrow$Logical)

Active$\leftrightarrow$Logical is a tree2tree task containing input and output trees in active voice and logical forms [61]. Transforming a tree in active voice to its logical form simulates semantic parsing, and transforming a logical form tree to active voice simulates natural language generation. For this dataset, there are three test sets: IID, 0-shot lexical, and structural. In addition to the baselines listed in the previous section, we also compare our modified `sDTM` to the original `DTM` to see the changes in parameter count and memory. The results are show in Table 1.

The various `DTM` models and Transformers all perform perfectly on the IID test set. NQG struggles to learn the Active$\leftrightarrow$Logical task, an example of the brittleness of hybrid neurosymbolic systems. Only the `DTM` variants succeed on the OOD test sets. As anticipated, the RU-Transformer performs better than the standard Transformer with regards to structural generalization.

[4]NQG uses pre-trained BERT embeddings [13]; it is unknown how much this pre-training helps the method.

Table 1: **Active↔Logical accuracy.** Results are the best performance over five runs. The test sets are divided into IID, and OOD sets 0-shot lexical and structural. Parameter and memory usage is shown for the original `DTM` with TPRs and our proposed sparse `DTM` with and without pruning. Our modifications reduce the parameter count by almost two orders of magnitude. *NQG was trained on a seq2seq version without parantheses because it was not able to learn the tree2tree training set.

Model	Split				
	IID	0-Shot Lexical	Structural		
Transformer	1.0	0.0	0.0		
RU-Transformer	1.0	0.0	.12		
NQG*	.45	0.0	0.0		
				Parameters	Memory (GB)
Original `DTM`	1.0	1.0	1.0	72M	12.3
`sDTM`	1.0	1.0	1.0	1M	9.7
`sDTM` (pruned k=1024)	1.0	1.0	.95	1M	1.8

Comparing the original `DTM` to `sDTM` without pruning, we see a 70x reduction in parameter count from pooling by attention, as well as a 20% reduction in memory usage from fewer parameters and SCT. We are able to gain even further memory savings due to the pruning method. The final two rows show that the pruning method has no impact on lexical generalization and a minor impact on structural generalization, while reducing memory usage by 5x. The results from this experiment confirm that `sDTM` is capable of matching `DTM` performance on a previous baseline. Next we turn to tasks where the original `DTM` could not be used.

4.2 Scalability (FOR2LAM)

FOR2LAM is a tree2tree program translation task to translate an abstract syntax tree (AST) in an imperative language FOR to an AST in a functional language LAM [6]. This makes FOR2LAM a good dataset to test the scalability of `sDTM` to more complex samples. We augment the FOR2LAM dataset with a 0-shot lexical test set. During training, only two variable names appear: 'x' and 'y'. For the 0-shot test, we replace all occurrences of x in the test set with a new token 'z'. We are unable to test `DTM` on FOR2LAM because a batch of 1 does not fit into memory due to the depth of the trees.

Results on FOR2LAM are shown on the left side of Table 2. NQG suffers with scale (see A.8), and we were unable to include results for it on FOR2LAM due to training and evaluation exceeding 7 days. All other models do well on the in-distribution test set, but only `DTM` is able to achieve substantive accuracy on the 0-shot lexical test. `DTM`'s performance is impressive given work on data augmentation has shown the difficulty of few-shot generalization is inversely proportional to vocabulary size [46], with smaller vocabulary tasks being more challenging. This 0-shot challenge is from 2 variables (x, y) to 3 (x, y, z), making it difficult enough that both transformer variants score 0.03%. ta

4.3 Seq2Tree (GeoQuery)

GeoQuery is a natural language to SQL dataset [69] where a model needs to map a question stated in natural language to a correctly formatted SQL query, including parentheses to mark functions and arguments. We use the parentheses and function argument relationship as the tree structure for our output. In this format, GeoQuery is a seq2tree task, and we follow the description from Section 3.5. We use the same preprocessing and data as Shaw et al. [53]. The TMCD split for GeoQuery [53] extends MCD to natural language datasets instead of synthetic languages. GeoQuery is a very small dataset, with a training set containing between 440 and 600 samples, depending on the split. We are unable to test `DTM` on GeoQuery because a batch size of 1 does not fit into memory.

Results for GeoQuery are shown on the right side of Table 2. This is the most difficult task that we test because of the small training set, and the natural language input is not draw from a synthetic grammar. Given this, a potential symbolic solution to this task might be quite complex. We find that both NQG and `DTM` perform worse than the two Transformer variants on the IID test set. This also

	FOR2LAM		**GeoQuery**			
Model	IID	0-shot lexical	IID	Length	Template	TMCD
Transformer	1.0	.03	.88	.26	.79	.40
RU-Transformer	1.0	.03	.87	.25	.77	.37
NQG†	–	–	.76	.37/.26*	.62	.41
`sDTM`	1.0	.61	.73	.18	.20	.35

Table 2: **Accuracies on FOR2LAM and GeoQuery.** Results are the best performance over five runs. †Results taken from Shaw et al. [53]. *We report the results from a replication study of NQG where the result on the Length split differed substantially from the original result [63].

Table 3: **SCAN accuracy.** Results are the best performance over five runs. MCD scores are calculated as the average of the three MCD splits. †Results from Shaw et al. [53]. *Results from Sun et al. [63].

Model	Split					
	IID	1-shot lexical	0-shot lexical	Length	Template	MCD
Transformer	1.0	.08	0.0	.07	1.0	.02
RU-Transformer	1.0	.11	0.0	.19	1.0	.01
NQG†	1.0*	1.0	0.0	1.0	0.0*	1.0
`sDTM` (parse trees)	1.0	.99	.99	.75	.95	.03
`sDTM` (LAUD trees)	1.0	.87	.98	.06	.98	0.0

holds true for the Template split, where Transformers outperform the neurosymbolic models. On the Length and TMCD splits, all of the baselines achieve roughly the same performance while `DTM` performs slightly worse — the degree of variation in the input space and small training set appear to make it difficult for `sDTM` to find a compositional solution.

4.4 Seq2Seq (SCAN)

SCAN is a synthetic seq2seq task with training and test variations to examine OOD generalization [34]. To process seq2seq samples, we follow the description in Section 3.5. We compare two methods for embedding the output sequence into a tree by writing a parser for SCAN's output and comparing this to the left-aligned uniform-depth trees (LAUD). In addition to the standard test splits from SCAN, we introduce a 0-shot lexical test set as well.

All models perform well on the IID test set, showing that they have learned the training distribution well. Transformer variants perform poorly on lexical, length, and MCD splits. The Transformers and `sDTM` perform well on the Template split while NQG completely fails. Along with the results from GeoQuery, which showed weak `sDTM` performance on the Template split and strong performance from both Transformers, it seems that the Transformer architecture is robust under template shifts between training and testing. `sDTM` is the only model to perform well on the 0-shot lexical test set, whereas NQG is the only model able to perform well on the MCD test set. The two `sDTM` rows compare models trained with output trees from a parser or LAUD encoding. The main performance difference is on the Length split, where the structurally relevant information in the parse trees is necessary for `sDTM` to perform well. It is not necessary to have structured input for the model to perform well on length generalization as long as the output is structured.

5 Conclusions

We introduced the Sparse Differentiable Tree Machine (`sDTM`) and a novel schema for efficiently representing trees in vector space: Sparse Coordinate Trees (SCT). While not perfect — `sDTM` struggles with MCD and Template shifts, as well as the extremely small GeoQuery dataset — the model generalizes robustly across the *widest variety* of distributional shifts. `sDTM` is also uniquely capable of zero-shot lexical generalization, likely enabled by its factorization of content and structure. While these capacities for generalization are shared with the original `DTM`, our instantiation is computationally efficient (representing a 75x reduction in parameters) and can be applied to seq2seq, seq2tree, and tree2tree tasks. Our work reaffirms the ability of neurosymbolic approaches to bridge the flexibility of connectionist models with the generalization of symbolic systems. We believe continued focus on efficient neurosymbolic implementations can lead to architectures with the kinds of robust generalization, scalability, and flexibility characteristic of human intelligence.

References

[1] Jacob Andreas. Good-Enough Compositional Data Augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7556–7566, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.676. URL `https://www.aclweb.org/anthology/2020.acl-main.676`.

[2] Yonatan Belinkov and James Glass. Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72, 2019.

[3] Tarek R. Besold, Artur S. d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro M. Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luís C. Lamb, Daniel Lowd, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. *CoRR*, abs/1711.03902, 2017. URL `http://arxiv.org/abs/1711.03902`.

[4] Chris M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7(1):108–116, 1995. doi: 10.1162/neco.1995.7.1.108.

[5] Terra Blevins, Omer Levy, and Luke Zettlemoyer. Deep rnns encode soft hierarchical syntax. *arXiv preprint arXiv:1805.04218*, 2018.

[6] Xinyun Chen, Chang Liu, and Dawn Song. Tree-to-tree neural networks for program translation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper_files/paper/2018/file/d759175de8ea5b1d9a2660e45554894f-Paper.pdf`.

[7] Xinyun Chen, Chen Liang, Adams Wei Yu, Dawn Song, and Denny Zhou. Compositional generalization via neural-symbolic stack machines. *Advances in Neural Information Processing Systems*, 33:1690–1701, 2020.

[8] Noam Chomsky. *Aspects of the theory of syntax*. Number no. 11 in Massachusetts Institute of Technology. Research Laboratory of Electronics. Special technical report. The MIT Press, Cambridge, Massachusetts, 50th anniversary edition edition, 1965. ISBN 978-0-262-52740-8.

[9] Noam Chomsky. *The Minimalist Program*. The MIT Press, 12 2014. ISBN 9780262327282. doi: 10.7551/mitpress/9780262527347.001.0001. URL `https://doi.org/10.7551/mitpress/9780262527347.001.0001`.

[10] Henry Conklin, Bailin Wang, Kenny Smith, and Ivan Titov. Meta-learning to compositionally generalize. *arXiv preprint arXiv:2106.04252*, 2021.

[11] Róbert Csordás, Kazuki Irie, and Juergen Schmidhuber. The devil is in the detail: Simple tricks improve systematic generalization of transformers. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 619–634, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.49. URL `https://aclanthology.org/2021.emnlp-main.49`.

[12] Fernando Cuetos, Don C Mitchell, and Martin MB Corley. Parsing in different languages. In *Language processing in Spanish*, pages 163–208. Psychology Press, 2013.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019. URL `http://arxiv.org/abs/1810.04805`. arXiv: 1810.04805.

[14] Li Dong and Mirella Lapata. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*, 2016.

[15] Brian DuSell and David Chiang. Stack attention: Improving the ability of transformers to model hierarchical patterns. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=XVhm3X8Fum.

[16] Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *arXiv preprint arXiv:2007.08970*, 2020.

[17] Artur SD'Avila Garcez, Luis C Lamb, and Dov M Gabbay. *Neural-symbolic cognitive reasoning*. Springer Science & Business Media, 2008.

[18] Marta Garnelo and Murray Shanahan. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 29: 17–23, 2019. ISSN 2352-1546. doi: https://doi.org/10.1016/j.cobeha.2018.12.010. URL https://www.sciencedirect.com/science/article/pii/S2352154618301943. Artificial Intelligence.

[19] Ross W Gayler. Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience. In Peter Slezak, editor, *Proceedings of the ICCS/ASCS Joint International Conference on Cognitive Science (ICCS/ASCS 2003)*, pages 133–138, Sydney, NSW, AU, jul 2003. University of New South Wales. URL http://arxiv.org/abs/cs/0412059.

[20] Adele E Goldberg. *Constructions at work: the nature of generalization in language*. Oxford University Press, Oxford; New York, 2006. URL http://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=3052348. OCLC: 193697889.

[21] Saul Gorn. *Explicit Definitions and Linguistic Dominoes*, pages 77–115. University of Toronto Press, Toronto, 1967. ISBN 9781487592769. doi: doi:10.3138/9781487592769-008. URL https://doi.org/10.3138/9781487592769-008.

[22] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. *Advances in neural information processing systems*, 28, 2015.

[23] Thomas L. Griffiths. Understanding human intelligence through human limitations. *Trends in Cognitive Sciences*, 24:873–883, 2020. URL https://api.semanticscholar.org/CorpusID:221996148.

[24] Demi Guo, Yoon Kim, and Alexander M. Rush. Sequence-Level Mixed Sample Data Augmentation. *arXiv:2011.09039 [cs]*, November 2020. URL http://arxiv.org/abs/2011.09039. arXiv: 2011.09039.

[25] Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. *Advances in neural information processing systems*, 28, 2015.

[26] Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 1:139–159, 2009.

[27] 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. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SygcCnNKwr.

[28] Najoung Kim and Tal Linzen. COGS: A compositional generalization challenge based on semantic interpretation. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.731. URL https://aclanthology.org/2020.emnlp-main.731.

[29] Najoung Kim and Tal Linzen. COGS: A Compositional Generalization Challenge Based on Semantic Interpretation. *arXiv:2010.05465 [cs]*, October 2020. URL http://arxiv.org/abs/2010.05465. arXiv: 2010.05465.

[30] Najoung Kim, Tal Linzen, and Paul Smolensky. Uncontrolled lexical exposure leads to overestimation of compositional generalization in pretrained models. *arXiv preprint arXiv:2212.10769*, 2022.

[31] Yoon Kim, Chris Dyer, and Alexander M Rush. Compound probabilistic context-free grammars for grammar induction. *arXiv preprint arXiv:1906.10225*, 2019.

[32] Dan Klein and Christopher D Manning. A generative constituent-context model for improved grammar induction. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 128–135, 2002.

[33] Denis Kleyko, Dmitri A. Rachkovskij, Evgeny Osipov, and Abbas Rahimi. A survey on hyperdimensional computing aka vector symbolic architectures, part i: Models and data transformations. *ACM Comput. Surv.*, 55(6), dec 2022. ISSN 0360-0300. doi: 10.1145/3538531. URL `https://doi.org/10.1145/3538531`.

[34] Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. *35th International Conference on Machine Learning, ICML 2018*, 7:4487–4499, 2018. arXiv: 1711.00350 ISBN: 9781510867963.

[35] Brenden M Lake. Compositional generalization through meta sequence-to-sequence learning. *Advances in neural information processing systems*, 32, 2019.

[36] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/lee19d.html`.

[37] Bingzhi Li, Lucia Donatelli, Alexander Koller, Tal Linzen, Yuekun Yao, and Najoung Kim. SLOG: A Structural Generalization Benchmark for Semantic Parsing, October 2023. URL `http://arxiv.org/abs/2310.15040`. arXiv:2310.15040 [cs].

[38] Yuxuan Li and James McClelland. Representations and computations in transformers that support generalization on structured tasks. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL `https://openreview.net/forum?id=oFC2LAqS6Z`.

[39] Matthias Lindemann, Alexander Koller, and Ivan Titov. Compositional generalization without trees using multiset tagging and latent permutations. *arXiv preprint arXiv:2305.16954*, 2023.

[40] Adam Lopez. Statistical machine translation. *ACM Computing Surveys (CSUR)*, 40(3):1–49, 2008.

[41] Gary F. Marcus. *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. MIT Press, 2001.

[42] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, 3(4):184–195, 1960.

[43] R Thomas McCoy, Tal Linzen, Ewan Dunbar, and Paul Smolensky. Rnns implicitly implement tensor product representations. *arXiv preprint arXiv:1812.08718*, 2018.

[44] Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher D Manning. Characterizing intrinsic compositionality in transformers with tree projections. *arXiv preprint arXiv:2211.01288*, 2022.

[45] Barbara Partee et al. Lexical semantics and compositionality. *An invitation to cognitive science: Language*, 1:311–360, 1995.

[46] Arkil Patel, Satwik Bhattamishra, Phil Blunsom, and Navin Goyal. Revisiting the compositional generalization abilities of neural sequence models. *arXiv preprint arXiv:2203.07402*, 2022.

[47] Steven Pinker. *The language instinct: How the mind creates language*. Penguin uK, 2003.

[48] Tony A. Plate. *Holographic Reduced Representation: Distributed Representation for Cognitive Structures*. CSLI Publications, USA, 2003. ISBN 1575864290.

[49] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1), jan 2020. ISSN 1532-4435.

[50] Anian Ruoss, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Róbert Csordás, Mehdi Bennani, Shane Legg, and Joel Veness. Randomized positional encodings boost length generalization of transformers. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, ed-

itors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1889–1903, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-short.161. URL `https://aclanthology.org/2023.acl-short.161`.

[51] Jake Russin, Jason Jo, Randall C O'Reilly, and Yoshua Bengio. Compositional generalization in a deep seq2seq model by separating syntax and semantics. *arXiv preprint arXiv:1904.09708*, 2019.

[52] Itiroo Sakai. Syntax in universal translation. In *Proceedings of the International Conference on Machine Translation and Applied Language Analysis*, 1961.

[53] Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *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, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.75. URL `https://aclanthology.org/2021.acl-long.75`.

[54] Vighnesh Shiv and Chris Quirk. Novel positional encodings to enable tree-based transformers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper_files/paper/2019/file/6e0917469214d8fbd8c517dcdc6b8dcf-Paper.pdf`.

[55] Vighnesh Shiv and Chris Quirk. Novel positional encodings to enable tree-based transformers. *Advances in neural information processing systems*, 32, 2019.

[56] David Smith and Jason Eisner. Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In Philipp Koehn and Christof Monz, editors, *Proceedings on the Workshop on Statistical Machine Translation*, pages 23–30, New York City, June 2006. Association for Computational Linguistics. URL `https://aclanthology.org/W06-3104`.

[57] Paul Smolensky. On the proper treatment of connectionism. *Behavioral and Brain Sciences*, 11 (1):1–23, 1988. doi: 10.1017/S0140525X00052432.

[58] Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artif. Intell.*, 46:159–216, 1990.

[59] Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In Jun'ichi Tsujii, James Henderson, and Marius Paşca, editors, *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211, Jeju Island, Korea, July 2012. Association for Computational Linguistics. URL `https://aclanthology.org/D12-1110`.

[60] Paul Soulos, Tom McCoy, Tal Linzen, and Paul Smolensky. Discovering the compositional structure of vector representations with role learning networks. *arXiv preprint arXiv:1910.09113*, 2019.

[61] Paul Soulos, Edward Hu, Kate McCurdy, Yunmo Chen, Roland Fernandez, Paul Smolensky, and Jianfeng Gao. Differentiable Tree Operations Promote Compositional Generalization, June 2023. URL `http://arxiv.org/abs/2306.00751`. arXiv:2306.00751 [cs].

[62] Mark Steedman. Combinatory grammars and parasitic gaps. *Natural Language & Linguistic Theory*, 5(3):403–439, 1987.

[63] Kaiser Sun, Adina Williams, and Dieuwke Hupkes. A Replication Study of Compositional Generalization Works on Semantic Parsing. August 2023. URL `https://openreview.net/forum?id=MF9uv95psps`.

[64] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[66] Yau-Shian Wang, Hung-Yi Lee, and Yun-Nung Chen. Tree transformer: Integrating tree structures into self-attention. *arXiv preprint arXiv:1909.06639*, 2019.

[67] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR, 2020.

[68] Dani Yogatama, Yishu Miao, Gabor Melis, Wang Ling, Adhiguna Kuncoro, Chris Dyer, and Phil Blunsom. Memory architectures in recurrent neural network language models. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=SkFqf0lAZ`.

[69] John M Zelle and Raymond J Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055, 1996.

A Appendix

A.1 Related Work

Work leveraging the generalizability of tree structures has a long history across Computer Science, Linguistics, and Cognitive Science [8, 42, 52, 58, 62]. Much of classical NLP aims to extract structured representations from text like constituency or dependency parses [for overview: 12, 40]. More recent work has shown the representations learned by sequence-to-sequence models without structural supervision can recover constituency, dependency, and part of speech information from latent representations in machine translation and language models [2, 5]. While those analyses show structural information is encoded, they stop short of showing that the representations themselves are tree-structured. Analyses inspired by Tensor Product Representations [43, 60] and chart parsing [44] give an account of how representations become somewhat tree-structured over the course of training.

Despite the apparent emergence of semi-structured representations in transformers and LSTMs, these architectures still appear to struggle with the kinds of structural generalization that come easily to humans [34, 29, 27]. A variety of approaches try to tackle this problem through meta-learning [35, 10], data augmentation [1], or decomposing the task into separate parts [51, 39]. The Relative-Universal Transformer [11] combines relative positional embeddings with a recurrent component, in an effort to emphasize local structures while allowing for unbounded computation.

Explicitly tree structured network architectures have been introduced for RNNs [59], LSTMs [64, 14], and Transformers [66, 55]. However, these variants often do not outperform their unstructured counterpart on out-of-distribution challenges [61]. This may be because generalization requires both structured representations and operations that respect that structure. A separate line of work considers neural architectures that are used to parameterize components of a symbolic system [31, 7] or that leverage explicit stack operations [15, 22, 25, 68]. NQG from Shaw et al. [53] combines the outputs from neural and symbolic models by inducing a grammar, but deferring to T5 [49] when that grammar fails. However the grammar's induction method has polynomial complexity with both dataset size and sequence length, which limits its application to larger tasks.

Vector Symbolic Architectures (VSAs) implement symbolic algorithms while leveraging high dimensional spaces [48, 19, 26, 33]. VSAs are similar to uniform neurosymbolic approaches, although VSAs commonly lack a learning component. Our work extends that of Soulos et al. [61] which can be viewed as integrating Deep Learning and VSAs. They introduce the Differentiable Tree Machine for Tree-to-Tree transduction. Here we instantiate a sparse Sequence-to-Sequence version with far fewer parameters and improved memory efficiency.

A.2 Sparse Coordinate Trees as Tensor Product Representations

This section shows that Sparse Coordinate Trees is the same as a TPR with the constraint that the role basis is the standard basis. TPRs define structural positions as role vectors $r_i \in \mathbb{R}^{d_r}$, and the content that fills these positions is defined by filler vectors $f_i \in \mathbb{R}^{d_f}$. For a particular role and filler pair, the filler f_i is *bound* to the role r_i using the tensor/outer product: $f_i \otimes r_i \in \mathbb{R}^{d_f \times d_r}$. The representation of an entire structure is the sum over all N individual filler-role pairs: $T = \sum_{i=1}^{N} f_i \otimes r_i \in \mathbb{R}^{d_f \times d_r}$. As shown in the previous two equations, the dimensionality of a single filler-role pair is equal to the dimensionality of an entire structure: both have dimensionality $\mathbb{R}^{d_f \times d_r}$. This means that a tree with only a filled root node takes up the same memory as a dense tree with every node filled. An important requirement for TPRs is that the role vectors must be linearly independent; this ensures that a filler can be *unbound* from a role without introducing noise using the inner product: $f_j = T r_j^+$, where $\{r_i^+\}_i$ is the basis dual to $\{r_i\}_i$. Previous work typically used randomly initialized and frozen orthonormal vectors to define the role basis. By defining our role vectors in a sparse manner as opposed to random initialization, we can greatly reduce the memory used by TPRs.

Classic symbolic data structures grow in memory linearly with the number of filled positions. It is possible to replicate this behavior with TPRs by defining the role vectors to be the standard one-hot basis, which is orthonormal by definition. The i-th element of role vector r_i is 1, and the other elements are 0. When a filler and role vector are both dense, the resulting bound vector is also dense. When the role vector is one-hot, the resulting bound vector is 0 everywhere except for column i which corresponds to the value 1 in r_i. By using a sparse tensor representation that only keeps track of dimensions that are not equal to 0, we can reduce the memory usage of TPRs to linear

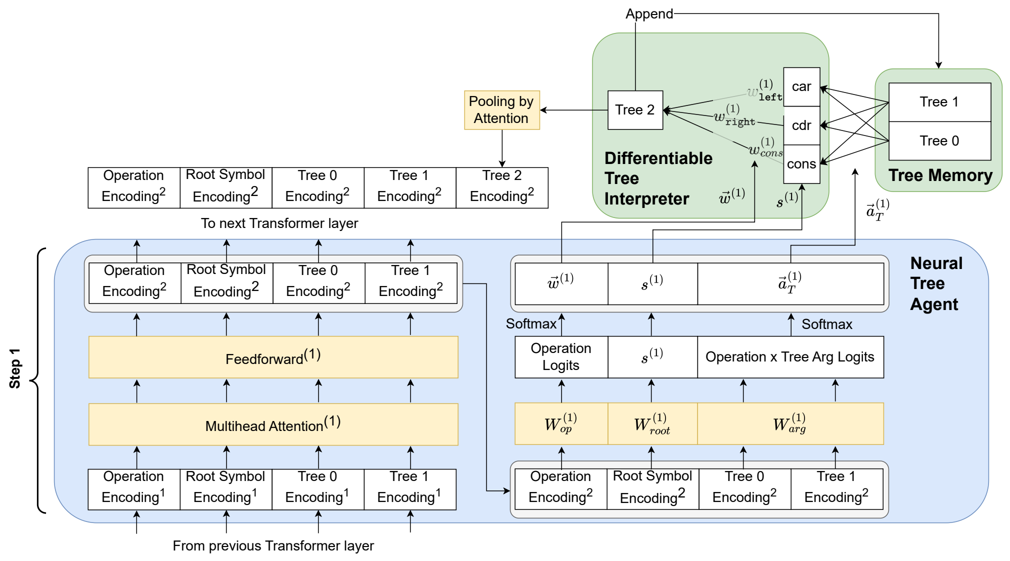

Figure 6: Adapted from Soulos et al. [61]. One step of DTM is expanded to show how the agent produces the input to the interpreter. The interpreter then writes the output to memory and encodes the output for the agent. Parts of the architecture with learnable parameters are indicated in yellow. The agent uses three linear transformations on top of a standard Transformer encoder layer to parameterize the inputs to the interpreter. The superscript indicates the layer number and refers to parameters and activations that are exclusive to this layer.

growth that scales with the number of filled positions, like a classical symbolic data structure. This however forgoes a motivating desideratum for the design of TPRs, that roles (and not just fillers) have similarity relations that support generalization across structural positions.

We can additionally improve the efficiency by refraining from performing the outer product. Since we are not performing a **tensor** product, this technique is only implicitly a **Tensor** Product Representation. Instead, we can keep the filler and role vectors in two aligned lists. A filler is bound to a role by sharing an index in our aligned lists. This is equivalent to the *binding* and *unbinding* from classical dense TPRs without having to perform multiplication.

Since we are not performing an outer product, instead of storing sparse role vectors, we can simply store a role integer, where the integer corresponds to the one-hot dimension. We derive a tree addressing scheme based on Gorn addresses [21]. In our scheme, addresses are read from right to left, giving the path from the root where a left-branch is indicated by a 0 and a right-branch is indicated by a 1. We need a way to distinguish between leading 0s and left-branches (e.g., 010 vs. 10), so we start our addressing scheme at 1 instead of 0. This indicates that all 0s to the left of the left-most 1 are unfilled and not left-branches; the left-most 1 and all preceding 0s are ignored when decoding the path-from-root. Figure 2 shows an example encoding of a tree in the sparse implicit approach.

We can compare the memory requirements of the Sparse Coordinate Tree encoding used in the sDTM to the memory requirements of the full TPRs used in the original DTM of Soulos et al. [61]. A TPR uses the same amount of memory regardless of the number of filled nodes. As with all sparse tensor formats, the memory savings arise when there are many zeros. In a dense tree where every node is occupied, the classical dense TPR approach is actually more efficient: the SCT's value list has the same total dimension as the classical TPR, but, in addition, the SCT encoding includes the list of filled-node addresses.

A.3 Agent Figure

See Figure 6.

Table 4: Comparing sDTM's accuracy on SCAN 1-shot lexical OOD generalization with and without lexical regularization. We use LAUD to embed the output sequence in a tree.

With noise	.87
Without noise	0.0

A.4 Lexical Regularization Ablation

To see the importance of adding noise to our input embeddings as defined in Section 3.4, we show the performance of sDTM with and without this regularization in Table 4.

A.5 Dataset Preprocessing

We preprocessed GeoQuery according to the steps from Shaw et al. [53]. FOR2LAM and GeoQuery both contain non-binary trees, which we convert to binary form using Chomsky normal form. When a new node is inserted to make a branch binary, we use the token <NT>. For output sequences with length one embedded according to left-aligned uniform-depth, we make the single token the left child of a new <NT> root node.

A.6 0-shot Lexical Test Generation

For both FOR2LAM and SCAN, we introduce 0-shot lexical tests. For FOR2LAM, we do this by replacing every occurrence of 'x' in the test set with a new token 'z'. For the SCAN 0-shot set, we start with the 1-shot lexical test set and remove the sample containing the 1-shot word 'jump'. We alter the output vocabulary to use the same tokens as the input vocabulary, since it is impossible for a word level model to translate between an input and output word without any exposure to that word.

A.7 DTM Training Details

When applicable, we adopt the hyperparameters from Soulos et al. [61]. Below we list the newly introduced hyperparameters and changes we made to existing parameters.

Soulos et al. [61] set the dimensionality of the embeddings to be equal to the size of vocabulary. This works for the datasets with small vocabulary examined in the original paper. We keep this setting for Active↔Logical, but set the embedding dimension to 64 for FOR2LAM, and 128 for GeoQuery and SCAN. We also changed the loss function from mean-squared error to cross entropy.

For each new task, we need to decide how many layers to use for sDTM. We followed the heuristic of doubling the max tree depth for the models with sequence input and quadrupling the number of layers for tree input. This leads to 56 layers for FOR2LAM, 22 layers for GeoQuery, and 14 layers for SCAN.

Pooling by multi-headed attention 3.2 introduces new hyperparameters such as number of pooling heads and pooling key dimensionality, and we set the value of these to be the same as the Transformer hyperparameters for the agent. Tree pruning 3.3 introduces a new hyperparameter k for the maximum number of nodes to keep. In general, a larger k is better but uses more memory. For Active↔Logical we set $k = 1024$, for FOR2LAM $k = 1024$, for GeoQuery $k = 2048$, and for SCAN $k = 256$. With the memory savings from SCT, pooling by multi-headed attention, and pruning, we increase the batch size from 16 to 64. We also increased the agent's model dimension to 256 with 8 heads of attention due to the memory savings except for Active↔Logical where we matched the original hyperparameters.

Random positional embeddings (RPE) also introduce a new hyperparameter for the max input integer, and we set this to be double the max input length. This leads to an RPE hyperparameter of 44 for GeoQuery and 18 for SCAN.

We noticed that randomly initializing and freezing our embedding vectors was essential for sDTM to achieve 0-shot generalization on SCAN.

For the results, we reported the best run of 5 random seeds. Like `DTM`, `sDTM` suffers from high variance. Some runs get stuck in local optima and fail to achieve moderate performance on the training set, which leads to poor performance on the test sets. This is a known issue with models that use superposition data structures, and reporting the best run over a number of random seeds has been previously used [68, 15].

A.8 Baseline Training Details

NQG: Active↔Logical rule induction used the following hyperparameters: sample size=training set size, terminal code length=8, allow repeated nts=True. The terminal code length setting was obtained via grid search over the values 1, 8, 32. For the actual training of the model we follow the hyperparameters utilised by [63, 53]. FOR2LAM used the same hyperparameters with the exception of sample size which had to be set to 1000 as additional increases became computationally intractable. Even under these settings rule induction took 42 hours on a machine with 64gb of ram. Writing the training set would take an additional week of processing time, which we considered computationally too expensive.

Transformer: We followed the same hyperparameters obtained via grid search from [61]. Specifically these are: 30,000 steps of which 1000 were warmup and linear learning rate decay; batch size 256; one encoder layer and three decoder layer each with a hidden dimension of 1024 and two attention heads; the optimizer was Adam.

RU-Transformer: We followed the hyperparameters reported by [11]. These are: 128 dimension hidden size with 256 feedforward; 8 attention heads; 3 layers; batch size 256; trained using Adam with learning rate 10^{-3}.

A.9 Compute resources

All reported `sDTM` runs could be processed on NVIDIA 16gb V100 GPUs. Depending on availability, we ran some seeds on 80gb H100 GPUs, but this is not necessary. The Transformer baselines were also run on NVIDIA 16gb V100 GPUs. NQG used NVIDIA 40gb A100 GPUs. The GPUs we used were hosted on an internal cluster.

Designing our architecture involved many preliminary experiments that are not reported in the paper.

A.10 Licenses

Baselines:

- DTM: Permissive 2.0
- Transformer: BSD-3 (Pytorch implementation)
- RU-Transformer: MIT Licence
- NQG: Apache 2.0

Datasets:

- GeoQuery: GLP 2.0
- SCAN: BSD
- Active↔Logical: Permissive 2.0
- FOR2LAM: Not public (no licence obtained through email request to original authors)