Evaluating Compositionality in Neural Models Using Arithmetic Expressions

Anonymous ACL submission

Abstract

We introduce CobA, a dataset designed to evaluate the compositional properties of neural models. The dataset consists of simple arithmetic expressions combining natural integers with addition and multiplication operators. For example, $(5 + 4) \times 2$. We distinguish four aspects of compositionality: localism, substitutivity, productivity, and systematicity. We generate partitions of the dataset with specific in-domain and generalization sets, designed to evaluate the model's ability for each compositional aspect. By carefully selecting expressions from the in-domain and generalization sets, we introduce controlled differences between the two sets. We show that models achieve competitive performance on a random partition, for which there is no controlled difference. Yet, for partitions requiring compositional extrapolation, performances drastically decrease for most encoder architectures. We observe distinctions among architectures, in particular fixed-length context transformers, sequential or tree-structured LSTM.

1 Introduction

Compositionality is thought to be a key feature of human language. Symbolic generative theories of language indeed imply the possibility to produce an infinite number of grammatical phrases and sentences using only finite means (Chomsky, 1957; Montague, 1970; Hauser et al., 2010). Humans derive phrase meaning by composing syntactic and semantic components using compositional rules (Partee et al., 1984; Cann, 1993; Dowty, 2007).

On the other hand, language models are trained using self-supervised objectives with no direct linguistically oriented supervision. Nonetheless, recent large foundation models have shown striking abilities to process human language. They overperform humans on many benchmarks (Devlin et al., 2019; Brown et al., 2020). They also exhibit strong consistency on agreements (subject-verb, noun-adverb, verb-verb) which are determined by abstract structures and not just linear order of words (Linzen et al., 2016; Gulordava et al., 2018; Marvin and Linzen, 2018; Newman et al., 2021). Yet, many studies point that their compositional abilities are surprisingly limited and that they struggle to generalize to specific out-of-domain examples (Lake and Baroni, 2018; Kim and Linzen, 2020; Hupkes et al., 2020).

The ability of language models to process language without inducing exhaustive symbolic composition rules is not yet fully understood. Baroni (2019) suggests neural networks may process language using partially or different rules than humans. They emphasize human language is not fully characterized by algebraic rules. Language models might rely on less systematic phenomena such as semi-lexicalized constraints in syntax or irregular inflections. Tenenbaum (2018) explores the possibility that language models overcome their lack of compositional abilities with an exposition to huge amounts of data.

It is undoubtedly possible to train efficient language models without prior or posterior compositional properties. Yet, building more compositional models is an active subject of research. In particular using extensive pre-training, specific architectures or inductive biases (Russin et al., 2019; Furrer et al., 2020; Ontañón et al., 2021). Such methods seek to improve transformer models at learning compositional rules. The gain might be to increase the model robustness toward out-of-domain examples, that is examples with statistically different properties but generated with the same set of rules.

Evaluating model compositional properties is notoriously hard. First—as for every other language evaluation benchmark—creating the data is a critical step. Labeling raw data is time-consuming and it is difficult to control precisely the property of the examples. On the other hand, generating artificial data may lead to poor lexical or structural

diversity. Additionally, studies show that models might use lexical biases or shallow heuristics in the data to achieve the task (Linzen and Leonard, 2018). Popular benchmarks require to generate the answer, which makes it difficult to disentangle the effect of the encoding and decoding parts. Indeed, some errors might be related to the decoding part. This makes it difficult to precisely assess model compositional properties.

We introduce **CobA**, a **Co**mpositionality **b**enchmark using **A**rithmetic. Arithmetic indeed defines a self-contained universe which can be described using a limited set of symbols and composition rules. This makes it easy to build specific examples with isolated properties. Moreover, we formulate the benchmark as a classification task. The task can be solved using a simple encoder, without the need to decode the answer. Thus, our setup minimizes complex interactions between encoding and decoding.

We organize our paper as follows: we first review the related work in Section 2. In Section 3, we detail the data generation process and the distinct dataset partitions. In Section 4, we present our training and evaluation setup. We also present our main results on the CobA generalization set. Finally, in Section 5, we perform an in-depth study to better analyze how the choice of hyperparameters might leverage specific performances and how the expression complexity impacts model performances.

2 Related work

Evaluating compositionality Specific datasets exist to assess compositionality of language models: SCAN (Lake and Baroni, 2018), PCFG (Hupkes et al., 2020), CFQ (Keysers et al., 2020) and COGS (Kim and Linzen, 2020). All use a text-to-text setup: models take raw text as input and should map it to a semantic form. For the SCAN dataset, raw sentences should be mapped to a sequence of instructions, CFQ maps sentences to Sparql queries, and COGS to semantic forms.

Other setups exist: Dasgupta et al. (2018) measure compositionality for sentence embeddings using a natural language inference task. They construct pairs of examples that may be solved using compositional operations. Andreas (2019) propose a formalism to measure compositionality using similarity metrics through a communication game.

The work of Bowman et al. (2015) is perhaps the closest to ours. They analyze model compositional abilities by inferring logical relations between pairs of sentences. Such sentences are artificially generated using an artificial language based on logical statements. They compare the impact of structured models to encode these sentences with explicit latent recursive structures. In our work, we try here to better characterize the effect of encoder architectures, given the various compositional aspects.

Enhancing model compositional ability is addressed through multiple means. Some models propose to integrate structural biases within the architecture: in particular Tree-LSTM (Tai et al., 2015) or latter in transformers with structured attention (Russin et al., 2019). Some methods propose also to adapt the pre-training or fine-tuning procedure (Furrer et al., 2020). Finally, other methods propose to complete models with modules dedicated for compositional operations (Liu et al., 2020; Ontañón et al., 2021).

Using arithmetic with neural networks Neural networks may be properly trained to solve mathematical expressions. A line of work outlines that pre-trained language models or static word embeddings capture scales and notions of numeracy (Wallace et al., 2019; Naik et al., 2019; Sundararaman et al., 2020; Zhang et al., 2020; Thawani et al., 2021). Beyond representing numbers, further work also analyzes the ability of models to perform basic mathematics reasoning (Saxton et al., 2019; Dua et al., 2019; Geva et al., 2020) or solve mathematical expressions (Lample and Charton, 2020).

3 Dataset description

The key contribution in our work is to build a dataset of arithmetic expressions to evaluate neural model compositional abilities. Indeed, numerically evaluating formal expressions theoretically requires capturing the formal rules of arithmetic. CobA focuses on aspects of compositionality that may also be declined for language (Hupkes et al., 2020). But contrary to previous work, we do not mix raw text and numeracy. We derive many arithmetic expressions, using limited formal explicit rules and symbols. In this section, we detail the generation of the dataset.

3.1 Generation procedure

Using an automatic procedure, we generate arithmetic expressions. First, we generate a natural

Partitions	Random	Systematicity	Productivity	Localism	Substitutivity
Mean value	66.3 / 66.2	67.0 / 68.2	66.4 / 65.3	66.1 / 67.4	66.1 / 66.4
Min number of operators	0 / 0	2 / 2	**0 / 4**	2 / 2	2 / 2
Mean number of operators	2.8 / 2.8	2.6 / 2.5	**2.8 / 6.5**	2.8 / 2.7	2.8 / 2.7
Max number of operators	3 / 3	3 / 3	**3 / 9**	3 / 3	3 / 3
Expressions with odds and evens (%)	85.6 / 85.4	**0.0 / 100.0**	85.6 / 98.3	85.7 / 85.3	85.7 / 85.1
Swapped-expressions in-domain (%)	1.3 / 1.6	10.0 / 0.0	1.2 / 0.0	1.2 / 1.5	**0.0 / 0.0**
Sub-expressions in-domain (%)	2.2 / 2.3	4.2 / 1.3	2.1 / 0.7	**0.0 / 00**	1.0 / 1.1

Table 1: Dataset key statistics given each partition. For each statistic, we report the figures for the in-domain / generalization set. We express the "Expressions with odds and evens", "Swapped-expressions in-domain" and "Sub-expressions in-domain" as the proportion of expressions verifying the property for each set. The statistics that are determinants for the aspect studied in a given partition appear in **bold**.

integer between 1 and 100, for example, 34. We then decompose it as the addition or multiplication of two other integers, for example, 2×17. We then recursively decompose each integer in the new expression as the product or sum of two integers or keep it unchanged, with a probability p[1].

As in Lample and Charton (2020), we use prefix notation (also known as normal Polish notation). The arithmetic expression $2 \times (14 + 3)$ is represented as the sequence $\times 2 + 14\ 3$. This notation avoids the use of parenthesis and therefore leads to shorter expressions. We assign each symbol—natural integer or operator sign—to a given token. For each expression, we make the distinction between two distinct left and right hand sides of an operator. For example, we make the distinction between the expression $14 + 3$ and $3 + 14$. Given this procedure, we generate over 2.5M unique expressions with between 0 and 9 operators.

3.2 Partitioning the dataset

Our dataset aims at evaluating model compositional properties. We take inspiration from the procedure proposed in SCAN (Lake and Baroni, 2018; Loula et al., 2018). We carefully select expressions to create partitions (in-domain and generalization sets) from the dataset. We split them such that in-domain and generalization sets have different distributions. It is not possible to infer generalization examples without fully capturing the properties ruling this specific aspect in the in-domain set. We thus compare the ability of models to perform *out-of-domain* generalization. We make the distinction between model learning shallow heuristics such as local pattern matching and the one learning true compositional operations.

We build partitions given the work from Hupkes et al. (2020), which distinguishes sub-properties within compositionality. **Localism**, **Substitutivity**, **Productivity** and **Systematicity**[2]. Each of the partitions detailed below has a key statistic distribution and is designed to evaluate a model's performance along a given aspect. Each partition contains an in-domain set of 24,000 expressions and a generalization set of 12,000 expressions. We present other key statistics for the partitions in Table 1.

Random is a regular training procedure. We split the dataset randomly without any specific control during the selection of the expressions. While in-domain and generalization examples are all distinct they share the same distributions and have similar underlying characteristics.

Systematicity evaluates the recombination of known parts to form new sequences. We build the partition using the distinction between odd and even natural integers. The training set contains expressions with only either odd or even numbers. For examples $2 \times 4 + 8.$ or $3 + 5 + 7.$ The test set contains expressions with both even and odd numbers such as $3 + 2 \times 5 + 4$.

Productivity evaluates the extrapolation to longer sequences. We train the model on expression with up to 3 operators. We then evaluate the model on longer expressions with up to 9 operators.

Substitutivity evaluates the robustness towards the introduction of synonyms. In our work, we interpret this definition as the robustness towards paraphrases and evaluate the ability of models to perceive an operator's commutative property. We

[1]We implement specific rules for numbers that are prime and cannot be decomposed with multiplication. We set the probability p to expand, by default, at 0.5.

[2]Hupkes et al. (2020) also enumerate the overgeneralisation aspect which evaluate the accommodation to exceptions. However, we find it complex to adapt this property for our specific dataset and therefore discard it in this work.

organize our dataset as a collection of "swapped expressions". Swapped expressions are tuples of expressions with the same value and that only differ by swapping the left and right hand sides of each operator. We illustrate this swapping organization in Figure 1. During training, we only expose the model to a single expression per tuple. Therefore the model cannot learn the commutative property from shallow pattern matching. During the evaluation, we evaluate the model's commutative ability by comparing couples of predicted values for expressions from the same tuple.

Localism evaluates the recursive evaluation of smaller constituents before larger constituents. We also organize our dataset as a collection of "sub-expressions". Sub expressions are tuples of expressions with the same value that only differ by the level of decomposition of the expressions as illustrated in Figure 1. We use the same training and evaluation protocol as for Substitutivity.

4 Experiments

We train the model using a classification objective. Given an arithmetic expression, the model predicts its value among the 100 possible integers. This setup avoids the use of a complex decoder module ; a simple probe is sufficient. The architecture is decomposed as follows: first, an encoder maps the arithmetic expression to an embedding vector. Then, a two-layer perceptron followed by a softmax outputs a probability distribution. We train the model by minimizing a cross-entropy loss.

4.1 Encoder architectures

We list below the encoders architectures we use.

Bow The expression's embedding vector is the sum of its embeddings from each symbol. This representation accounts for neither the order nor the structure of the expression.

Sequential LSTM We use unidirectional or bidirectional LSTM (Hochreiter and Schmidhuber, 1997). We use the last token hidden state as embedding for the expression.

Tree LTSM We represent each arithmetic expression as a binary tree where each node is either a natural number or an operator sign. We encode the tree using a N-ary tree LSTM (Tai et al., 2015) and use the root node as expression embedding.

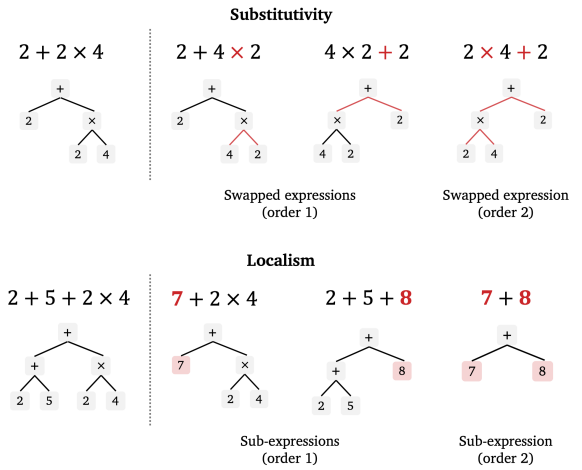


Figure 1: Generation of expressions tuples for probing substitutivity and localism[3]. For localism, we can deduce expressions from the expression seed by evaluating sub-components. For substitutivity, we can deduce expressions from each other by swapping left and right hand sides of operators.

Transformers We derive two simple encoders from the architectures of BERT (Devlin et al., 2019) and ALBERT (Lan et al., 2020). We use the [CLS] token hidden state from the last layer as expression embedding. We initialize our models randomly and train them from scratch. Their architectures are light compared with standard transformer scales: we use a hidden size of 128, 6 hidden layers, and 8 attention heads. This represents 1.2M parameters for BERT and 300k for ALBERT since parameters are tied across layers. As observed in Csordás et al. (2021) and Ontañón et al. (2021), transformers' positional encoding are particularly important for this task. We use the method from Wallace et al. (2019) and add some random padding at the beginning of the input so that the encoder does not solve the task by overfitting the absolute position of the symbols.

4.2 Training configuration

We design all encoders comparable, with roughly the same number of parameters (1.2M), as detailed in Table 2. We also use the same hidden and embedding size range for all encoders: 256 for LSTM-based encoders and 128 for transformer-based encoders. We use the same optimization procedure for each model. We train all models using the AdamW optimizer (Loshchilov and Hutter, 2019) with a $1e^{-3}$ learning rate, 1 epoch warm-up with polynomial decay and a batch size of 100. For each

[3]For the clarity of the illustration, we use the infix form for the expressions

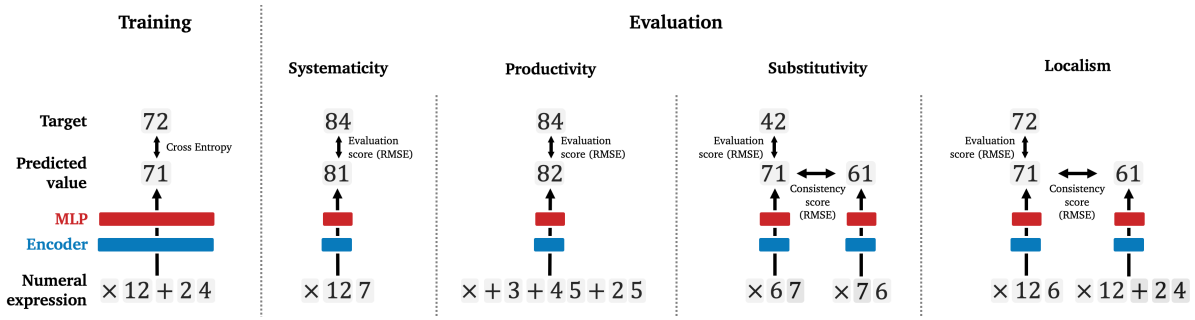


Figure 2: Illustration of the train and evaluation setup for assessing model compositional properties using the CobA dataset. We train the model to evaluate in-domain expressions. We then infer expressions from the generalization set. The dataset includes partitions for **Localism**, **Substitutivity**, **Productivity**, and **Systematicity**.

partition, we separate the in-domain set between a train and dev set using a random 90/10% split. We measure the RMSE between the expression predicted and the true value on the dev set. We stop the training when no improvement is made for 5 consecutive epochs or after 100 maximum epochs. We train all models on an Nvidia 2080 Ti GPU. The training time is around 10 minutes per partition and model. Since our setup is relatively light, we set parameters given the literature on the subject and do not perform hyper-parameters search[4].

Regarding the natural integer embeddings, We use the DICE method (Sundararaman et al., 2020). The method uses a deterministic approach to construct natural integer embeddings. It obtains state-of-the-art results on evaluation benchmarks (Wallace et al., 2019). We do not update natural integer embeddings during training. For operators and specific tokens such as CLS or SEP tokens, we initialize embeddings randomly (with the same scale) and update them during training.

4.3 Evaluation setup

We evaluate our models and report metrics from the generalization set. We illustrate the evaluation setup in Figure 2. For the random, systematicity, and productivity partitions, we compute an evaluation score as the mean RMSE between each expression's true and predicted value. For the localism and substitutivity partitions, we refine the evaluation procedure to take advantage of the additional paired structure of the partition described in section 3.2. We compute both an *agreement* score as the mean RMSE between the predicted values from the two expressions of each pair and an *evaluation* score as the RMSE between the predicted and true value of the expression. We report the harmonic mean between the *agreement* and *evaluation* scores. This score reflects the consistency of the model's predictions between two expressions as well as its ability to predict the true value. It for example discards trivial models which always predict the same value or model accurately evaluating one expression of the pair but failing for the other.

[4]Except for the random seed, since we observe it is a crucial parameter for transformers.

4.4 Results

Table 2 presents the results on the generalization set. We use two baselines: one that randomly predicts the value of any expression and the BoW model. We use the RMSE to compare the models. The lower it is, the better are predictions on average.

By a small margin, BoW outperforms the random guessing baseline. This suggests that the task requires accounting for the expression structure. Lexical information may provide insights for solving the task: an expression containing numbers such as a 56 and 43 is more likely of being equal to a high value such as 89 than an expression containing only a 2 and a 3. Yet, local information alone may not be sufficient to solve the task since expressions with a high overlap may greatly differ in value. For example, the expressions $3 + 3$ and 3×3 contain the same symbols but are not equal since they used different mathematical operators.

Models can generalize to examples with similar distributions. On the random partition, all the models indeed achieve low RMSE, significantly lower than the baselines. Models are also robust toward the introduction of paraphrases since scores on substitutivity and random partitions are similar. For other partitions, results are more contrasted.

In general, encoders relying on LSTM cells outperform transformers. For productivity and systematicity, sequential models strongly outper-

Encoders	Number of parameters (×100)	Random	Localism	Systematicity	Productivity	Substitutivity
Random	—	39.3 (0.1)	40.0 (0.1)	39.9 (0.2)	39.3 (0.1)	40.0 (0.1)
BoW	68	28.7 (8.1)	11.7 (10.8)	37.0 (0.2)	38.3 (5.4)	0.0 (0.0)
LSTM (uni)	595	3.5 (0.4)	6.7 (0.8)	2.7 (0.3)	14.1 (0.4)	2.4 (0.3)
LSTM (bi)	1,186	2.2 (0.2)	6.9 (0.8)	2.5 (0.3)	13.5 (1.1)	1.3 (0.1)
LSTM (tree)	1,012	5.4 (0.1)	8.6 (0.2)	7.4 (0.6)	15.0 (0.5)	4.8 (0.3)
Transformer (BERT)	1,290	3.6 (1.1)	10.1 (0.4)	20.6 (4.9)	30.2 (1.8)	2.2 (0.7)
Transformer (ALBERT)	315	6.9 (1.5)	12.2 (0.9)	16.6 (6.6)	25.9 (3.1)	4.6 (0.8)

Table 2: Compositionality evaluation. We report metrics from the generalization set. For the random, systematicity and productivity partitions, we report the evaluation score, which is the RMSE between the true and the predicted values. For the localism and ssubstitutivity partitions, we report the harmonic mean between the evaluation and consistency score. For each metric, we report the mean value over 4 runs (standard deviation in parentheses).

form models using fixed-length context. Surprisingly, sequential LSTMs constantly outperform tree LSTMs, despite having fewer structural biases.

Regarding transformers, the RMSE highly deteriorates for productivity. We confirm their known limitation in terms of productivity. We also observe transformers stumbling upon systematicity. Finally, we observe the benefit of tying parameters. ALBERT use the same architecture as BERT, except that weights are tied across layers. ALBERT achieve results comparable or above BERT despite using far less parameters.

5 In-depth analysis

Since we use generated arithmetic expressions, it is easy to control their fine-grained properties. We investigate further the influence of parameters for each partition. As part of our study, we observe how the complexity of the examples impacts compositional abilities and how we can enhance them by modifying model hidden size or exposing models to generalization examples during training.

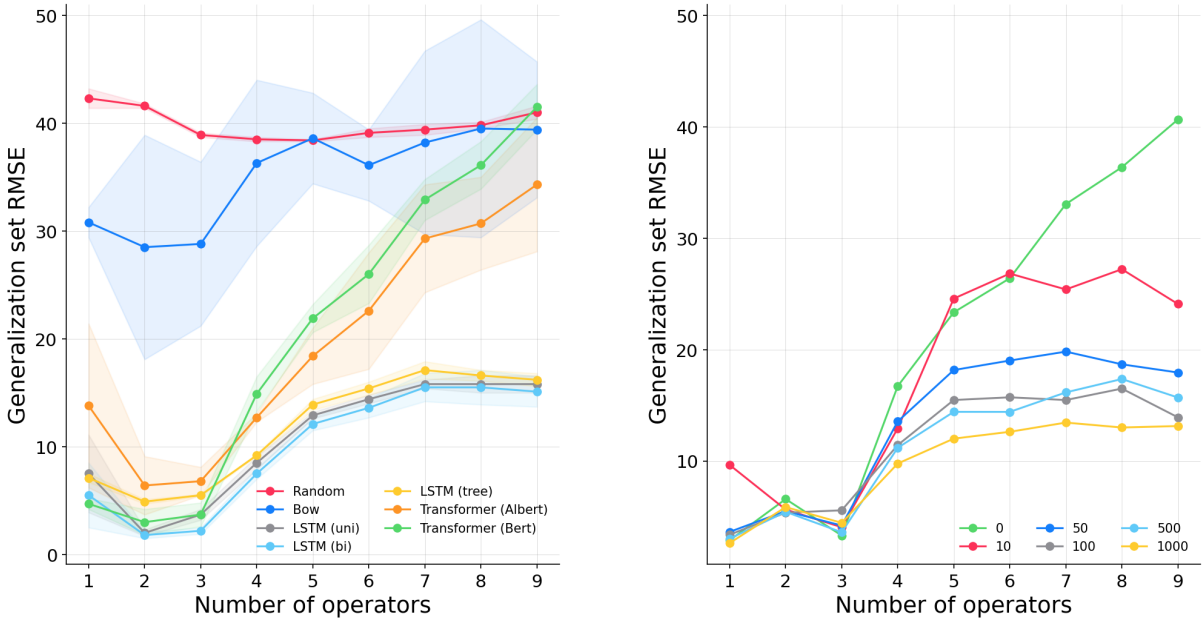
5.1 Impact of the expression's complexity

Complexity of compositional operations As observed in Table 2, all models perform reasonably well on the random partition. Yet, this performance might be heterogeneous across examples. We decompose the examples according to the type of operations involved. We consider expressions containing at least one addition sign (**Add**), at least one multiplication sign (**Mul**), only addition sign(s) (**Only Add**), only multiplication sign(s) (**Only Add**) and at least one multiplication and addition sign (**Add and Mul**). We present the performance given this stratification in Figure 4a.

Arithmetic expressions involving at least one addition operator obtain better results. Multiplications, on the other hand, tend to make the task harder. In expressions that involve addition and multiplication, there may be cases where multiplication should take precedence over addition, and for which computation order matters. Surprisingly, these expressions reach performance in line with expressions containing only one operator type.

Number of operators for productivity Productivity is notoriously hard (Kim and Linzen, 2020; Hupkes et al., 2020; Baroni, 2019). We also observe that neural networks struggle to generalize to longer expressions in our main results Table 2. In Figure 3a, we decompose the productivity generalization set according to the number of operators per expression and plot the evolution of the RMSE. In line with intuition, performance declines as the number of operators grows. This evolution is not uniform across architectures: LSTM architectures generalize better to long sequences.

Number of swaps For substitutivity, we organize the dataset given tuples of swapped expressions. During evaluation, we pair each expression with an expression from the same tuple and we compare the predicted value between the two. As illustrated in Figure 1, we can rank all expression pairs given the number of swaps necessary to generate one given the other. For example, given the expression $2 + 2 \times 4$ we can generate $2 + 4 \times 2$ with only one swap. We refer to this pair as level-1. We need to perform two swaps to generate $4 \times 2 + 2$: we refer to the pair as level-2. Figure 4c decomposes the results from the substitutivity partition given these levels. Encoders tend to reach better performance for expression pairs with only one swap.



(a) (b)

Figure 3: Evolution of the RMSE on the productivity generalization set given the number of operators. (a) We report the mean evolution over 4 runs for each encoder (standard deviation in light). As detailed in Table 1, expressions from the in-domain set have 2.8 operators on average. (b) For the BERT-based encoder, we expose the model to a proportion of generalization examples during training.

Complexity of local evaluations For the localism partition, we organize the dataset given tuples of sub-expressions. As illustrated in Figure 1, we can rank all expression pairs given the number of evaluated intermediate operators between the two. For example, given the expression $2 + 5 + 2 \times 4$, we evaluate only the first addition operator to generate $7 + 2 \times 4$. We refer to the expression pair as level-1. If also we evaluate the multiplication operator, we obtain $7 + 8$. We then refer to the expression pair as level-2. We compare the results on the Localism partition by decomposing generalization examples given this level. We aim to better quantify how local the encoder performs composition operations. Surprisingly, Figure 4b shows that level-2 expression pairs reach better scores for all encoders. We hypothesize that expressions with intermediate evaluated operators are shorter and therefore reach higher evaluation scores.

5.2 Enhancing model compositional abilities

Model hidden size The number of parameters indubitably boosts model performance. We analyze here whether the number of parameters can also leverage performance for out-of-domain generalization. We compare embedding and hidden sizes of 128, 256, and 512 and observe the impact on the out-of-domain generalization performances. We plot in Figure 4d the mean score for each partition given each encoder. For all encoders, we observe that the number of parameters benefits compositional generalization. On average, all models indeed reach the lowest RMSE on the generalization set with 512 hidden and embedding size than 128.

Exposition during training We study how to increase out-of-domain generalization for productivity. We consider exposing the model to a small number of out-of-domain examples during training. We randomly include between 10 and 1,000 expressions from the generalization set in the training samples. These expressions are then removed from the generalization set. In Figure 3b, we plot the evolution of RMSE on the productivity extrapolation set given the number of operators per expression for the transformer encoder.

With our dataset, only a minimal number of out-of-domain examples exposed during training may not be sufficient to trigger generalization during inference. Even by including a large portion, between 500 and 1,000 expressions, we still observe a significant performance drop for expressions with a large number of operators. With this configuration, the transformer falls short of the trend obtained with the sequential LSTM.

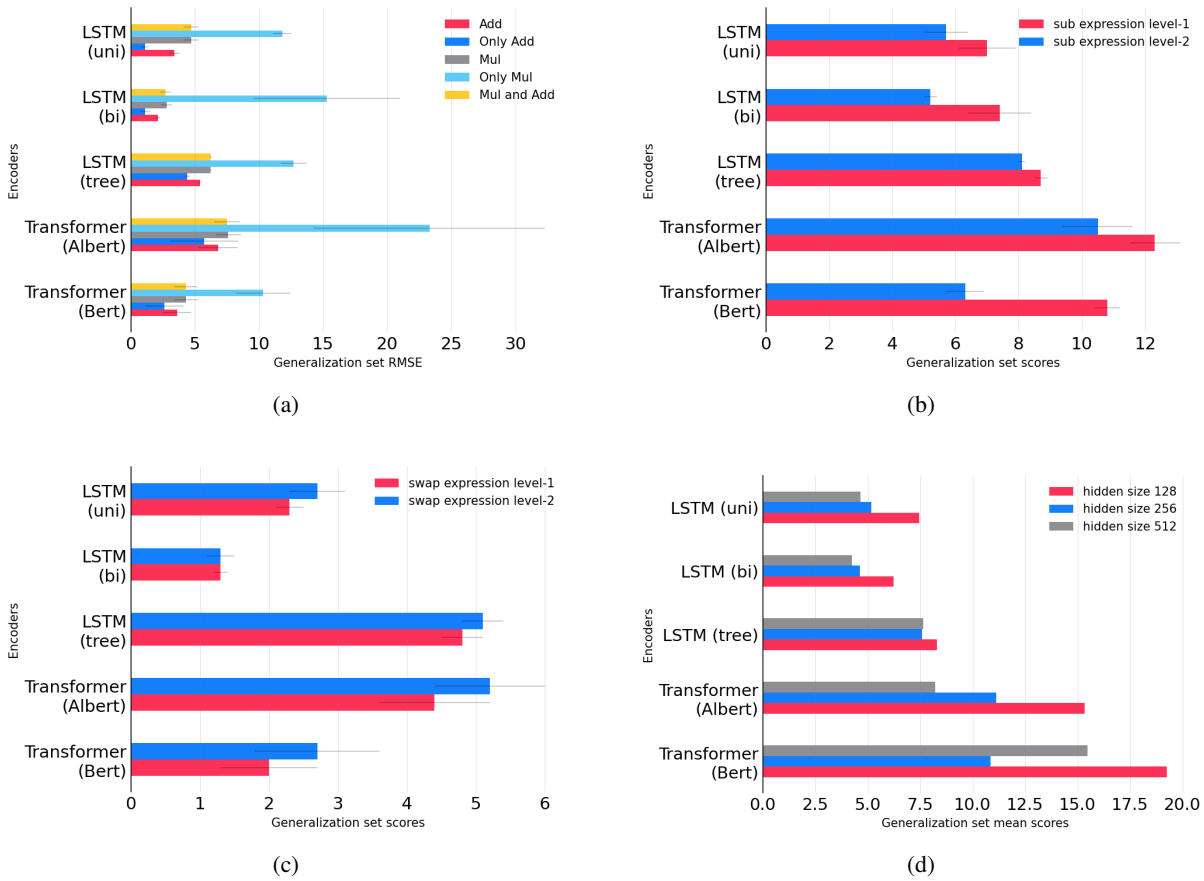


Figure 4: Impact of the expression complexity and model hidden size on the compositional performances. (a) We decompose the expressions from the random partition given the type of operators involved. (b) We decompose the expressions from the local partition given the number of sub-expression evaluated (c) We decompose the expressions from the substitutivity partition given the number of swaps (d) We compare the impact of the model hidden size on the average mean generalization score for each partition. For this specific analysis, we observe transformers might struggle to converge. We adapt the training procedure by increasing the warm-up to 1000 steps with no decay.

6 Conclusion

We introduced CobA, a dataset of arithmetic expressions. The dataset is specifically designed at evaluating model compositional properties. We partition the dataset into multiple in-domain and generalization sets. As arithmetic follows explicit and formal rules, we can precisely control the properties of the generated expressions and, consequently, the statistical properties of the sets. For each partition, we introduce controlled differences between the distribution of the two sets. We then study the ability of models to evaluate out-of-domain expressions from the generalization set, by learning compositional rules from the in-domain set. We build partitions to evaluate compositional properties (localism, substitutivity, productivity, and systematicity) that also apply to the study of human language. We hope our work may help better characterize compositional abilities to design more efficient encoders: either by adapting architectures or training methods.

We use the dataset to compare encoders with distinct structures: transformers, recurrent or tree-structured models. In general, models are robust toward the introduction of paraphrases (substitutivity) and can perform recursive evaluation of sub-components (localism). Yet, transformers struggle to generalize to longer sequences (productivity) or to combine known parts to form new sequences (systematicity). We perform an in-depth analysis and make observations in line with intuition. Models struggle with complex expressions, involving more symbols, more complex structure, or more operators types. We slightly enhance the compositionality degree by adapting the number of parameters or the exposition to generalization expressions. Yet the encoder architecture is the key parameter in our study.

Ethical considerations of the work

Our work aims at improving model compositional abilities. We hope it may enhance model out-of-domain generalization abilities. To extend, we hope this contributes to reducing the model's number of parameters or number of examples seen during training without loss of generality. Increased volume of data and parameters indeed require increased computational resources. Reducing this requirement at scale may therefore contribute to preserving a balance for a global sustainable environment. Better generalization abilities may also improve model robustness towards out-of-domain inference and therefore enhance their secure use.

References

Jacob Andreas. 2019. Measuring compositionality in representation learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Marco Baroni. 2019. Linguistic generalization and compositionality in modern artificial neural networks. *CoRR*, abs/1904.00157.

Samuel R. Bowman, Christopher D. Manning, and Christopher Potts. 2015. Tree-structured composition in neural networks without tree-structured architectures. In *Proceedings of the NIPS Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches co-located with the 29th Annual Conference on Neural Information Processing Systems (NIPS 2015), Montreal, Canada, December 11-12, 2015*, volume 1583 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Ronnie Cann. 1993. *Formal semantics: An introduction*. Cambridge University Press, Cambridge.

N. Chomsky. 1957. *Syntactic structures.* Mouton.

Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. 2021. The devil is in the detail: Simple tricks improve systematic generalization of transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 619–634. Association for Computational Linguistics.

Ishita Dasgupta, Demi Guo, Andreas Stuhlmüller, Samuel Gershman, and Noah D. Goodman. 2018. Evaluating compositionality in sentence embeddings. In *Proceedings of the 40th Annual Meeting of the Cognitive Science Society, CogSci 2018, Madison, WI, USA, July 25-28, 2018*. cognitivesciencesociety.org.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

David Dowty. 2007. Compositionality as an empirical problem. In *In Chris Barker and Pauline Jacobson (eds.) Direct Compositionality*, pages 23–101.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2368–2378. Association for Computational Linguistics.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. 2020. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *CoRR*, abs/2007.08970.

Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting numerical reasoning skills into language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 946–958. Association for Computational Linguistics.

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. 2018. Colorless green recurrent networks dream hierarchically. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1195–1205. Association for Computational Linguistics.

Marc D. Hauser, Noam Chomsky, and W. Tecumseh Fitch. 2010. *The faculty of language: what is it, who has it, and how did it evolve?*, Approaches to the Evolution of Language, page 14–42. Cambridge University Press.

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

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. Compositionality decomposed: How do neural networks generalise? *J. Artif. Intell. Res.*, 67:757–795.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

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 2020, Online, November 16-20, 2020*, pages 9087–9105. Association for Computational Linguistics.

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, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2879–2888. PMLR.

Guillaume Lample and François Charton. 2020. Deep learning for symbolic mathematics. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of lstms to learn syntax-sensitive dependencies. *Trans. Assoc. Comput. Linguistics*, 4:521–535.

Tal Linzen and Brian Leonard. 2018. Distinct patterns of syntactic agreement errors in recurrent networks and humans. In *Proceedings of the 40th Annual Meeting of the Cognitive Science Society, CogSci 2018, Madison, WI, USA, July 25-28, 2018*. cognitivesciencesociety.org.

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 *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

João Loula, Marco Baroni, and Brenden M. Lake. 2018. Rearranging the familiar: Testing compositional generalization in recurrent networks. In *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pages 108–114. Association for Computational Linguistics.

Rebecca Marvin and Tal Linzen. 2018. Targeted syntactic evaluation of language models. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1192–1202. Association for Computational Linguistics.

Richard Montague. 1970. Universal grammar. *Theoria*, 36(3):373–398.

Aakanksha Naik, Abhilasha Ravichander, Carolyn Penstein Rosé, and Eduard H. Hovy. 2019. Exploring numeracy in word embeddings. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3374–3380. Association for Computational Linguistics.

Benjamin Newman, Kai-Siang Ang, Julia Gong, and John Hewitt. 2021. Refining targeted syntactic evaluation of language models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 3710–3723. Association for Computational Linguistics.

Santiago Ontañón, Joshua Ainslie, Vaclav Cvicek, and Zachary Fisher. 2021. Making transformers solve compositional tasks. *CoRR*, abs/2108.04378.

Barbara Partee et al. 1984. Compositionality. *Varieties of formal semantics*, 3:281–311.

Jake Russin, Jason Jo, Randall C. O'Reilly, and Yoshua Bengio. 2019. Compositional generalization in a deep seq2seq model by separating syntax and semantics. *CoRR*, abs/1904.09708.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Dhanasekar Sundararaman, Shijing Si, Vivek Subramanian, Guoyin Wang, Devamanyu Hazarika, and Lawrence Carin. 2020. Methods for numeracy-preserving word embeddings. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 4742–4753. Association for Computational Linguistics.

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 of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1556–1566. The Association for Computer Linguistics.

Josh Tenenbaum. 2018. Building machines that learn and think like people. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, page 5. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM.

Avijit Thawani, Jay Pujara, Filip Ilievski, and Pedro A. Szekely. 2021. Representing numbers in NLP: a survey and a vision. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 644–656. Association for Computational Linguistics.

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. 2019. Do NLP models know numbers? probing numeracy in embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 5306–5314. Association for Computational Linguistics.

Xikun Zhang, Deepak Ramachandran, Ian Tenney, Yanai Elazar, and Dan Roth. 2020. Do language embeddings capture scales? In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2020, Online, November 2020*, pages 292–299. Association for Computational Linguistics.