Diverse Demonstrations Improve In-context Compositional Generalization

Itay Levy*   **Ben Bogin***   **Jonathan Berant**

The Blavatnik School of Computer Science, Tel-Aviv University

{itay.levy,ben.bogin,joberant}@cs.tau.ac.il

Abstract

In-context learning has shown great success in i.i.d semantic parsing splits, where the training and test sets are drawn from the same distribution. In this setup, models are typically prompted with demonstrations that are *similar* to the input utterance. However, in the setup of compositional generalization, where models are tested on outputs with structures that are absent from the training set, selecting similar demonstrations is insufficient, as often no example will be similar enough to the input. In this work, we propose a method to select *diverse* demonstrations that aims to collectively *cover* all of the structures required in the output program, in order to encourage the model to generalize to new structures from these demonstrations. We empirically show that combining diverse demonstrations with in-context learning substantially improves performance across three compositional generalization semantic parsing datasets in the pure in-context learning setup and when combined with finetuning.[1]

1 Introduction

Despite strong performance of pretrained language models (LMs) across many tasks, they have been shown to struggle in a compositional generalization setting (Lake and Baroni, 2018; Furrer et al., 2020; Shaw et al., 2021), when tested on their ability to process and generate novel combinations of previously observed elements. For example, a model might fail to interpret the request *"Book a meeting with Jake's supervisor"* even when *"Book a meeting with Jake"* and *"Who is Jake's supervisor?"* were observed during training. In semantic parsing, the task of mapping natural language utterances to formal queries, such generalization is important (especially in a real-world setting), since models are required to interpret new combinations that are
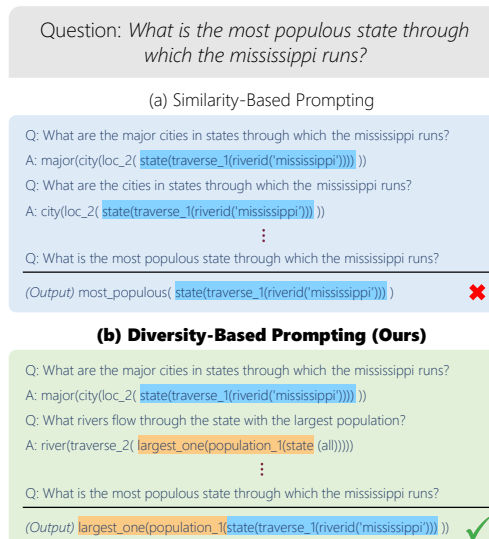

Figure 1: Compositional generalization setup: (a) Selecting demonstrations by considering only similarity to the input yields repetitive demonstrations that do not cover the structures in the target program. (b) However, choosing diverse demonstrations enables better coverage and leads to a correct prediction.

not covered by the annotated training data (Herzig and Berant, 2019; Yin et al., 2021).

Recently, large LMs have shown impressive performance on downstream tasks by conditioning on a text-based prompt that contains a few training examples. This type of few-shot inference is known as *in-context learning* (ICL, Brown et al., 2020). A core component of in-context learning is the set of examples in the prompt, often termed task *demonstrations*. With the right demonstrations, ICL can be an effective approach to improving LMs' compositional generalization abilities (Qiu et al., 2022b).

Selecting a relevant set of demonstrations is crucial for generalization. However, most past work only considered the relevance of each example *in isolation*, ignoring the quality of the entire set of examples (Liu et al., 2022). For instance, a retriever can be used to select the examples most similar to the input (Rubin et al., 2022). A set of demonstra-
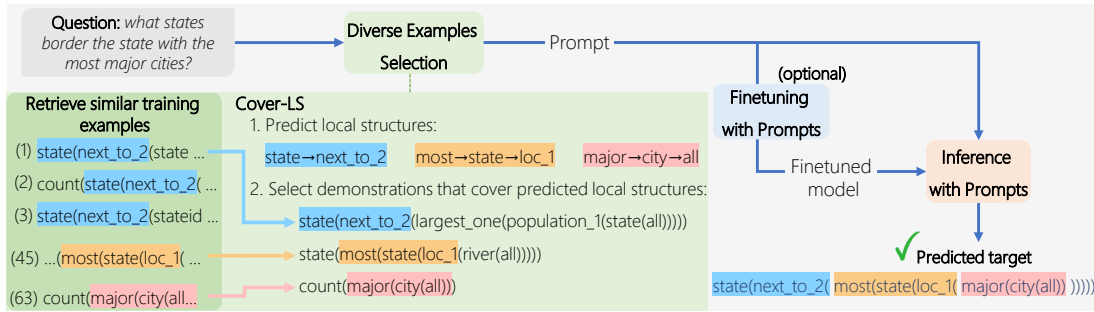

Figure 2: Overview of our framework. Given an utterance, we construct a prompt by selecting a set of diverse demonstrations. Feeding the prompt to the model yields the predicted target. Optionally, models can be finetuned (FT setup). In the bottom left corner, we see how *Cover-LS* selects diverse examples: predicting and covering *local structures*, thereby enabling the selection of complementary examples.

tions that are all highly relevant but highly similar to one another may not be as effective as a more **diverse** set. In compositional splits, where no single demonstration is sufficiently similar to the input, choosing diverse demonstrations can be especially beneficial since it leads to better coverage of structures in the target program (Fig. 1).

In this paper, we study how to leverage ICL to improve compositional generalization for semantic parsing, by optimizing the entire set of demonstrations and increasing the diversity of examples in this set. We investigate two approaches for increasing diversity: (a) a *coverage-based* approach, where we define a set of elements conditioned on the input utterance, and select examples that cover those elements (e.g., covering potential substructures in the output program), and (b) a second approach, where we select a subset of examples that are most dissimilar from one another, such that diversity is independent of the input utterance. Empirically, we find that coverage-based diversity results in better performance.

Our method can be used in the "pure" in-context learning setup without finetuning, which leverages the ability of large LMs, such as Codex (Chen et al., 2021), to generalize from the selected diverse demonstrations. Furthermore, it can be combined with finetuning by training a model with demonstrations as part of the input. This can be viewed as meta-learning, where the model learns to use demonstrations during training and build new structures based on them during inference (Finn et al., 2017; Lake, 2019; Conklin et al., 2021; Min et al., 2022; Chen et al., 2022). It can, however, lead to an over-reliance on demonstrations, especially in compositional splits. We address this by using "noisy" demonstrations during training.

We empirically test our method on three compositional generalization semantic parsing datasets. We show that diverse demonstrations, both with and without finetuning, improve performance by up to 23 absolute points (e.g., $50.3 \rightarrow 73.5$ on SMCalFlow-CS) compared to a baseline that retrieves demonstrations according to similarity alone, and lead to state-of-the-art results in multiple compositional setups. Finally, we show that our method reduces the number of demonstrations needed for generalization and improves test performance on hard examples.

2   Diversity for Compositional Generalization

In semantic parsing, we define compositional splits of datasets as splits where train and test programs do not overlap (Finegan-Dollak et al., 2018). Recent work has shown that increasing the number of different program structures a model sees during training improves performance on compositional splits. This can be done by augmenting the training set (Qiu et al., 2022a) or through efficient sampling of diverse examples (Oren et al., 2021; Bogin et al., 2022; Gupta et al., 2022). While past work focused on increasing structure diversity in the *training set*, we focus on diversity in the *demonstration set* within an ICL setup.

Increasing diversity is important as we want the demonstrations to *cover* all structures of the expected output program. In the few-shot setting, where the model is unfamiliar with the formal language of the output programs, increasing coverage also improves generalization simply since otherwise the model will be unaware of the required program symbols (predicates and logical operators). However, selecting demonstrations that cover

larger *structures* (sub-trees of the program tree) are potentially more beneficial, for two reasons: (1) it reduces the amount of new structures that the model needs to produce, making demonstration fusion easier, and (2) it exposes the model to structure compositions in different contexts, providing the model with valuable information about how structures can be composed in the data.

3 Diverse Demonstrations Selection

Problem setup Given a training set $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^n$ containing utterance-program pairs and a test utterance x_{test}, our objective is to select a subset of training examples $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^k \subset \mathcal{T}$, where $k \ll n$, termed demonstrations. Those demonstrations are then formatted as a text-based prompt P. When feeding the concatenation of the prompt and the test utterance ($[P; x_{\text{test}}]$) to the model, the desired output is y_{test}.

Overview Fig. 2 provides an overview of our framework for obtaining and leveraging diverse demonstrations for better compositional generalization. Given an input utterance, x_{test}, we propose two approaches for selecting demonstrations. In the first (§3.1), we optimize *coverage*: we define a set of elements that we want our demonstrations to cover (either structures in the program or utterance words), and then iteratively select examples that contain these elements. The second approach (§3.2) increases diversity by selecting a subset of examples with minimal similarity. Fig. 2 shows an example of the former approach (*Cover-LS*), where we predict and then attempt to cover *local structures* (LS), i.e., sub-trees of the output program. Local structures were shown to be key for compositional generalization in Bogin et al. (2022).

Having selected demonstrations, we use them to construct a prompt (§3.3). We show that our method can be combined with finetuning to meta-train the model to learn in-context (§3.4).

3.1 Coverage-based Selection

Bogin et al. (2022) have recently shown, in the context of finetuning semantic parsers, that models fail to generalize to programs with local structures that were not observed at training time, where local structures of a program are defined to be a set of its sub-trees. Inspired by this observation, we propose **Cover-LS**, an algorithm that given the test utterance x_{test}, attempts to choose examples that collectively cover as many local structures as possible from the set $\mathcal{S}_{y_{\text{test}}}$ of local structures of the program y_{test}. Since we have no access to y_{test} at test time, we predict what local structures are likely using an auxiliary model, assuming that predicting local structures is *easier* than predicting the entire program. Then, we iteratively select examples that cover the predicted local structures.

Local structures definition We follow the definition of Bogin et al. (2022), and given a program y, convert it to its abstract syntax tree, where each tree node is a program symbol and parent-child edges connect functions to their arguments. In addition, we add "sibling" edges between consecutive arguments. The local structures, $\mathcal{S}_{y_{\text{test}}}$, are a subset of all of the connected sub-graphs in the abstract syntax tree (e.g., state→next_to_2 and most→state→loc_1 in Fig. 2, see more examples in Tab. 8), as defined in App. B. Unlike Bogin et al. (2022), we consider local structures with any number of nodes. In addition, we anonymize programs by replacing values such as strings and numbers with constants (string and number), since such values are usually not relevant for program coverage.

Predicting local structures As mentioned, we assume predicting local structures is easier than predicting an entire program. Thus, we train an auxiliary model by finetuning T5 (Raffel et al., 2020) on the training set in the standard manner, training it to output anonymized programs given input utterances with no demonstrations. Then, for each test utterance, x_{test}, we use beam search to output B candidate programs $\{\tilde{y}_b\}_{b=1}^B$ and define the set of local structures as $\mathcal{S}_{\tilde{y}_{\text{test}}} = \bigcup_{b=1}^B \mathcal{S}_{\tilde{y}_b}$.

Covering local structures Our goal is to choose a set of demonstrations, $\mathcal{D}$, that covers the local structures in $\mathcal{S}_{\tilde{y}_{\text{test}}}$. Choosing an example for each local structure is infeasible due to prompt length limitations, and thus we propose Alg. 1, whose goal is to choose a small set of demonstrations that are (a) similar to the test utterance x_{test} and (b) cover as many local structures in $\mathcal{S}_{\tilde{y}_{\text{test}}}$ as possible.

We sort the LSs based on their size (number of nodes) in descending order (line 2). By first selecting training examples with programs that contain *larger* LSs from $\mathcal{S}_{\tilde{y}_{\text{test}}}$, we are more likely to include training examples similar to the test utterance, which should improve few-shot performance. Then, we iterate over all LSs, and for each local structure s we *retrieve* the most similar training example that contains s (line 6), and add it to $\mathcal{D}$

Algorithm 1: Cover-LS Algorithm

Input : List of candidate local structures to cover $\mathcal{S}$;
Pool of training examples $\mathcal{T}$; Retriever R ;
Desired number of output examples k

Output : Set of training examples $\mathcal{D}$

1 $\mathcal{D} = \emptyset$
2 Sort $\mathcal{S}$ from largest to smallest
3 **while** $|\mathcal{D}| < k$ **do**
4    $\mathcal{S}_{\text{uncovered}} = \mathcal{S}$
5   **for** *each* $s \in \mathcal{S}_{\text{uncovered}}$ **do**
6   Retrieve with R an example $e \in \mathcal{T}$ that contains s
7   Add e to $\mathcal{D}$
8   Remove from $\mathcal{S}_{\text{uncovered}}$ LSs that appear in e
9   Remove from $\mathcal{T}$ all examples with same anonymized program as e
10   **if** $|\mathcal{D}| == k$ **then**
11   break

(line 7). We then update the pool of LSs such that it will include only LSs that are not yet covered (line 8). To further encourage diversity, we remove from our example pool all examples that share the same template (program after anonymization) as the chosen examples (line 9). We keep choosing examples until reaching the desired amount of demonstrations, which might result in choosing more than one example for each local structure (lines 3-4).

We assume (line 6) access to a retriever that takes as input an utterance and returns similar training examples, from which we filter only examples that contain the desired structure. A variety of retrievers can be used, such as BM25 (Robertson and Zaragoza, 2009) or SBERT (Reimers and Gurevych, 2019).

We observe that in our setup, the running time of Cover-LS is negligible compared to the decoding time of the LMs.

Utterance coverage We propose a simpler variant that does not require predicting a set of local structures with an auxiliary model. This variant, termed **Cover-Utt**, uses the same coverage-oriented algorithm, but covers *words* in the input utterance, rather than predicted local structures. This is beneficial when the quality of the auxiliary model, and consequently predicted LSs, is low.

3.2 Diversity without Coverage

The primary challenge with coverage-based approaches is identifying the elements that need to be covered. An alternative approach is to define diversity more explicitly and select a subset of demonstrations that are dissimilar from one another (while being relevant for the input utterance).

A natural approach for choosing a subset of high-quality and diverse demonstrations from the training set is Determinantal Point Process (DPP) (Kulesza and Taskar, 2012), a probabilistic model that defines a probability distribution over subsets of items, giving high probability to subsets that contain *relevant* and *diverse* items. DPP requires a *relevance score* for each item and a *similarity score* between pairs of items. In our case, we define the relevance of a demonstration through its *retriever score* for the input test utterance. To compute the similarity between demonstration pairs, we first extract LSs and compute tf-idf vectors for each demonstration. The similarity of each pair is then the cosine similarity between their tf-idf vectors. Full implementation details are in App. E.

3.3 Prompt Construction

We order the chosen demonstrations according to their retriever score with respect to the input utterance in ascending order, in accordance to common practices (Liu et al., 2022). When finetuning the model (§3.4), demonstrations are shuffled. Demonstrations are formatted to a prompt according to the format in App. D, concatenated with the test utterance, and fed to the model.

3.4 Finetuning with Prompts

Despite the success of "pure" in-context learning, where model parameters are frozen, it has been by and large restricted to very large LMs. Conversely, finetuning requires more training data, but performs well even with smaller models. In-context learning can be easily integrated with finetuning by training a model with demonstrations as part of the input. This paradigm can be considered as meta-learning, where the model learns how to use demonstrations during training (Min et al., 2022).

When meta-learning is used in the i.i.d. setup, where the training and test examples are drawn from the same distribution, one can use the same procedure to select demonstrations at both training time and test time. However, in a compositional generalization setup, this does not work: at training time, the model will observe demonstrations that are similar to the target output and will learn to heavily rely on demonstrations and copy large chunks of them. Thus, the model will not learn to compose demonstration parts and will struggle with examples drawn from a different distribution.

To address this phenomenon, which we term *over-copying*, past work (Pasupat et al., 2021;

Dataset	Example

SMCalFlow-CS	*Can you make a meeting with David Lax's reports ?*
	`(Yield :output (CreateCommitEventWrapper :event (CreatePreflightEventWrapper`
	`:constraint (Constraint[Event] :attendees (AttendeeListHasPeople`
	`:people (FindReports :recipient (Execute :intension (refer (extensionConstraint`
	`(RecipientWithNameLike :constraint (Constraint[Recipient]) :name`
	`# (PersonName "David Lax"))))))))))))`

SMCalFlow-CS Simple (natural)	`CreateEvent (with_attendee (FindReports (recipient= refer (Recipient?` `(name= LIKE (David Lax))))))`
GeoQuery (natural)	*What is the most populous state through which the mississippi runs ?* `largest_one (population_1 (state (traverse_1 (riverid ("mississippi")))))`
COVR-10 (synthetic)	*What is the color of square dog ?* `query_attr[color] (filter (square, find (dog)))`

Table 1: An example utterance-program pair for each of the datasets.

Zemlyanskiy et al., 2022) used *sampling* to add noise to the demonstrations. Here, we also reduce the similarity of demonstrations to the input utterance, but with a simpler approach. Recall that our Cover-LS algorithm picks similar examples by (a) finding demonstrations that share *large* LSs with the predicted program (lines 2-6 in Alg. 1), and (b) using a retriever to find the most similar examples among these. To address over-copying, we modify this: at training time, we only consider LSs of size 1, i.e., program symbols, and for each such LS we randomly choose an example that contains this symbol rather than use a powerful retriever.

4 Experiments

We present our experimental setup and results on different compositional semantic parsing tasks, with finetuning (FT) and without (NoFT).

4.1 Datasets

We evaluate our methods on three datasets (examples in Tab. 1).

SMCalFlow-CS is a few-shot compositional generalization dataset proposed by Yin et al. (2021) derived from SMCalFlow (Andreas et al., 2020). It contains single-turn natural sentences involving two domains (organization structure and event creation), each having its own set of program symbols. The test set of the compositional splits contains only cross-domain examples, where both domains appear. We show results for a few-shot setting (split k-C, where $k \in \{8, 16, 32\}$) where the training set includes only k cross-domain examples, and a zero-shot setting (split 0-C). We also eval-

uate on an i.i.d. split[2] where the test set contains only single-domain examples. Prior studies on the dataset employed LISP and LISPRESS program formats, resulting in v1 and v2 versions, respectively (see an example in Tab. 9). We default to using v1, unless otherwise specified.

For our FT experiments, we use **SMCalFlow-CS Simple**, which contains the same utterances as SMCalFlow-CS, but with programs that use a simplified syntax provided by Meron (2022). We opt for this version because programs are much shorter, leading to a smaller memory footprint and accelerating training and inference.

GeoQuery (Zelle and Mooney, 1996; Tang and Mooney, 2001) contains 880 natural language questions about US geography. We use the standard (i.i.d.) and compositional splits created by Shaw et al. (2021): (1) template split, where target programs are anonymized into templates and then the templates are randomly split between training and test sets (Finegan-Dollak et al., 2018); (2) TMCD split, which makes the distributions of compounds in training and test sets as divergent as possible (Keysers et al., 2020); and (3) length split, where test sequences are longer than training ones. Similar to prior work, we average results across three TMCD and template splits to reduce variance caused by the small dataset size.

COVR-10 COVR (Bogin et al., 2022) is a synthetic dataset based on a variable-free functional language. COVR-10 contains 10 compositional grammar splits, in which each test set includes programs featuring a particular set of local structures not observed at training time. Results are averaged

[2]The split we use for the i.i.d. setup is 8-S.

	GeoQuery				SMCalFlow-CS					COVR-10
	i.i.d.	Templ.	TMCD	Len.	i.i.d.	0-C	8-C	16-C	32-C	
T5 (fine tuned w/o prompts)	90.3	85.9	75.4	36.0	88.5	0.0	34.5	39.0	50.0	21.5
Random	53.7	49.7	42.0	30.7	43.0	1.3	0.3	0.7	2.0	69.4
Top-K	86.3	78.0	71.8	64.3	81.7	17.0	34.0	35.7	50.3	61.8
Cover-Utt (ours)	**89.0**	82.1	77.8	73.7	83.3	**35.3**	51.0	51.3	69.7	**78.1**
DPP (ours)	87.0	81.2	77.8	**74.3**	79.3	34.7	44.0	50.0	59.7	62.7
Cover-LS (ours)	88.7	**85.3**	**79.4**	72.7	**86.0**	0.3	**53.3**	**58.3**	**73.5**	64.4
Top-K (Oracle)	86.3	74.5	76.2	55.7	85.0	0.0	33.0	54.0	59.6	35.4
Cover-LS (Oracle)	86.3	81.2	82.8	74.0	84.3	40.7	77.3	73.5	75.3	83.2

Table 2: **Main results, NoFT setup.** We show results of the Codex model on a random subset of 100 test examples across 3 seeds, with the results of a finetuned T5 model for comparison.

across the 10 splits.

4.2 Experimental setup

Models We use Codex (code-davinci-002) (Chen et al., 2021; Ouyang et al., 2022) for all NoFT experiments, and T5-large (Raffel et al., 2020) for FT experiments. T5-large is used to predict LSs in both the NoFT and FT setups.

Evaluation Like prior work, we use exact match accuracy as the main metric for evaluation. Results are averaged over 3 random seeds unless stated otherwise. In the FT setup, we use the entire test set for evaluation. In the NoFT setup, we use 100 test examples due to rate limits of the Codex inference API (and another 100 development examples for hyperparameter tuning).

Prompt We use a prompt size of $k = 24$ for NoFT experiments and $k = 3$ for FT experiments, unless stated otherwise. A prompt is truncated when its length exceeds the model's context length (excluding the tokens reserved for generation). In FT experiments, we included only the programs in our demonstrations and discarded their utterances, due to limitations of memory and sequence length (preliminary experiments with utterances showed this does not affect accuracy).

Retrievers In NoFT setup, we use BM25 over lower-cased utterance words. In FT setup, we use BM25 over predicted program symbols in $\mathcal{S}_{\tilde{y}_{\text{test}}}$ (predicted using T5). In Cover-LS experiments we use a random retriever at training time to avoid over-copying. We analyze other possible retriever choices in §4.5.

Hyperparameter tuning and model selection We train two types of models in this work: (a) models for predicting LSs, and (b) models finetuned with prompts. For both cases, we use the development set whenever it is available for model selection, otherwise, we use the last checkpoint. Similarly, we use the development set to tune the number of beam candidates B when predicting local structures, and if there is no development set, we set $B = 1$. We detail finetuning hyperparameters in App. F.

Local structure size In some experiments, we limit the maximum size of local structures (the number of nodes they contain). A subscript notation (Cover-LS$_d$ or DPP$_d$) indicates a limit up to size d.

4.3 Baselines

Finetuning without prompts Vanilla-finetuned T5 model which is trained without demonstrations, similar to the one used to predict LSs (§3.1), except that it is trained on non-anonymized programs.

Top-K We construct the prompt with the top-k examples that are most similar to x_{test} according to the retriever score.

Random We construct a prompt by randomly sampling k training examples without repetition.

We also conduct oracle experiments, where at test time we have access to y_{test} both for retrieval and LS coverage. The retriever takes as input the gold program and scores demonstrations using BM25 over the gold program symbols. In oracle Cover-LS, we cover local structures from $\mathcal{S}_{y_{\text{test}}}$ without predicting them with a model.

4.4 Main Results

NoFT We observe (Tab. 2) that all methods for increasing diversity (Cover-Utt, DPP and Cover-LS) outperform Top-K, which selects similar demonstrations without accounting for diversity, in 7 out of 8 compositional splits. In fact, all non-oracle diversity methods outperform an *oracle* Top-K in

	GeoQuery				SMCalFlow-CS				
	i.i.d.	Templ.	TMCD	Len.	i.i.d.	0-C	8-C	16-C	32-C
T5 Base (FT, **Qiu et al. 2022a**)	93.3	84.8	69.2	41.8	84.7 / -	-	34.7 / -	44.7 / -	59.0 / -
T5 Base + CSL-Aug (FT, **Qiu et al. 2022a**)	93.3	**89.3**	74.9	67.8	83.5 / -	-	51.6 / -	**61.4** / -	70.4 / -
T5 Base (FT, **Qiu et al. 2022b**)	92.9	84.8	69.2	40.0	- / 82.8	-	- / 21.7	- / 43.6	- / 58.9
T5 11B (Prompt Tuning, **Qiu et al. 2022b**)	**93.6**	87.7	**81.2**	41.5	- / 83.1	-	- / 0.0	- / 10.0	- / 23.6
PaLM 62B (FT, **Qiu et al. 2022b**)	92.5	85.1	72.7	44.2	- / 82.2	-	- / 26.9	- / 34.7	- / 51.1
PaLM 540B (ICL, **Qiu et al. 2022b**)	86.8	76.6	63.6	57.9	- / 58.3	-	- / 4.7	- / 5.0	- / 11.7
T5 Large (fine tuned w/o prompts)	92.5	83.8	73.5	37.2	85.3 / 83.3	0.0 / 0.0	34.3 / 6.9	43.0 / 33.6	56.1 / 53.6
Top-K (NoFT)	88.9	74.7	69.4	65.8	79.3 / 69.7	**19.8** / **13.6**	32.7 / 25.8	37.7 / 33.6	49.6 / 43.9
Cover-LS (NoFT)	91.4	81.6	76.3	**70.0**	82.2 / 73.6	0.0 / 0.0	**52.5** / **36.7**	60.9 / **60.3**	**75.1** / **64.7**

Table 3: **NoFT setup compared to past approaches** on the entire test set (single seed). Since past work reported results on different versions of SMCalFlow-CS, we report accuracy for both versions (v1 / v2).

7 out of 8 compositional splits, suggesting that retrieval methods that only consider similarity are sub-optimal even in an oracle setup. Similarly, all diversity methods improve performance compared to a finetuned T5 model in all compositional splits except GeoQuery's template splits. Furthermore, sampling random examples (Random baseline) results in poor performance in GeoQuery and SMCalFlow-CS, but achieves high accuracy in COVR-10, beating all methods except Cover-Utt. This can be explained by the synthetic nature and small vocabulary of COVR-10.

Comparing diversity methods, Cover-LS and Cover-Utt are better than DPP in 7 out of 10 splits, showing that covering the target input/program goes beyond simply picking diverse examples. Cover-Utt, which covers utterance words, works surprisingly well considering its simplicity. Coverage-based methods also outperform Top-K in i.i.d splits. One noticeable failure of Cover-LS is the 0-C split, where it fails to generalize, due to the poor T5 performance on this split (T5 baseline gets 0 accuracy). This emphasizes that if one cannot reasonably predict LSs, then covering input words is a viable alternative. Lastly, oracle methods outperform their non-oracle counterparts in most settings, but not always. This occurs because our oracle method, which has access to the gold program, does not guarantee the selection of the optimal set of demonstrations, a phenomenon also observed in Qiu et al. (2022b).

Tab. 3 shows accuracy on the entire test set (NoFT setup). Since the underlying models differ substantially, a fair comparison to previous work is impossible. Nevertheless, a comparison still provides a high-level overview for the state of these tasks. Results show that using Codex with Cover-LS outperforms a T5 finetuned with augmentation (Qiu et al., 2022a) in 4 compositional splits out of 6
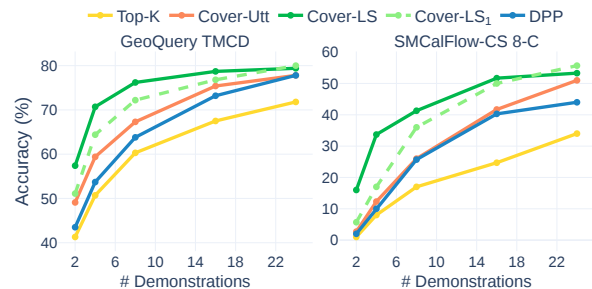

Figure 3: Comparing model accuracy (NoFT setup) based on the number of demonstrations, with multiple methods for selecting demonstrations.

(TMCD, Length, 8-C and 32-C), and outperforms non-finetuned PaLM 540B, where demonstrations are selected using BM25, in all splits.

Number of demonstrations (NoFT) We examine how performance is affected by the number of demonstrations in Fig. 3. Cover-LS outperforms Top-K by a large margin across all prompt sizes. Moreover, Cover-LS requires just four demonstrations in order to obtain roughly the same results as Top-K with 24 demonstrations. The gap between Cover-LS and Cover-Utt or Cover-LS$_1$ shows the importance of covering structures rather than just program symbols or utterance words, especially for small demonstration sets.

FT Finetuning results are shown in Tab. 4, where we detail separately the method used for demonstration selection at both training time and test time, as those may diverge to avoid over-copying.

First, using random demonstrations at test time, without controlling for diversity or using any retriever, is better compared to using no demonstrations at all. Our main method constructs prompts with Cover-LS at test time, but during training, prompts are retrieved with Cover-LS$_1$, that only covers program symbols, but not local structures, to avoid over-copying (see §3.4). This combination

Training Method	Test Method	GeoQuery				SMCalFlow-CS Simple				COVR-10
		i.i.d.	Templ.	TMCD	Len.	i.i.d.	8-C	16-C	32-C	
T5 (FT, w/o prompts)	-	92.5	83.8	73.5	37.2	83.7	9.7	37.5	59.4	19.4
Random	Random	**93.2**	85.0	76.8	39.8	83.5	28.3	46.4	58.0	23.2
Random	Top-K	93.0	84.6	75.9	39.8	83.4	24.4	40.6	54.8	22.8
Top-K	Top-K	90.7	54.7	57.4	20.8	83.2	8.8	22.1	46.1	19.6
Cover-LS$_1$	Cover-LS$_1$	92.9	85.3	76.6	41.9	83.9	**31.0**	**51.3**	**62.6**	**29.8**
Cover-LS$_1$	Cover-LS	93.1	**85.9**	**77.6**	**42.7**	**84.1**	30.5	50.6	61.5	28.6
Cover-LS$_2$	Cover-LS	92.6	84.9	75.6	39.8	83.7	28.8	46.3	60.5	28.8
Cover-LS	Cover-LS	91.8	80.7	69.4	37.7	82.9	21.2	34.1	53.8	13.6
Cover-LS$_1$	Cover-LS (Oracle)	93.7	87.7	79.8	48.9	87.4	48.0	64.1	73.5	41.1

Table 4: **FT results** using T5. We detail the method used for demonstration selection at both training time and test time as those may differ to avoid over-copying.
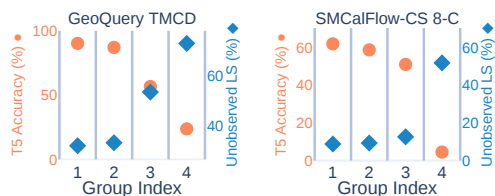

Figure 4: Properties of test example groups, where grouping is based on NoFT prediction outcome: (1) Top-K succeeds; (2) Cover-LS succeeds; (3) only Cover-LS succeeds; and (4) both fail.

leads to higher performance in all compositional splits compared to baselines that use Top-K or random sampling. Interestingly, using Top-K at both training time and test time yields low accuracy in compositional splits, but high results in i.i.d. splits. This corroborates our assumption that diversity is needed in compositional setups. Finally, A variant of our method, where Cover-LS$_1$ is used both during training and test time, is comparable to our main method across all splits.

We observe that limiting coverage at training time to program symbols is crucial: accuracy drops in all splits if we limit Cover-LS to structures up to size 2 (Cover-LS$_2$) instead of 1, or if we have no such limitation at all. The oracle Cover-LS outperforms all non-oracle models (unlike in NoFT, where this is not always the case).

4.5 Analysis

Stratified analysis Our main results show that Cover-LS outperforms Top-K in most compositional splits. But what examples does it perform better on? We analyze properties of test example groups, where grouping is based on NoFT prediction outcome: (1) Top-K succeeds; (2) Cover-LS succeeds; (3) only Cover-LS succeeds; and (4)

both fail. For each group we estimate difficulty by measuring the average accuracy achieved by a T5 model (finetuned without prompts), and also compute the percentage of examples that have an *unobserved local structure* (ULS) with respect to the training set. This measure is central to determining whether generalization to a test instance is hard, as shown in Bogin et al. (2022).[3]

We see (Fig. 4) that as the group index increases, T5 accuracy decreases and ULS rate increases. This finding confirms the claim in Bogin et al. (2022) that a test instance containing an ULS is hard. Examining groups 1 and 3, we observe that the group for which Cover-LS performs better than Top-K, is also tougher for T5 and has more ULS. Both methods fail on examples with low T5 accuracy and high ULS scores (group 4). This is also an evidence that T5 and Codex agree on the difficulty of examples, despite their different training and inference schemes. We provide error analysis in App. A.

Prompt metrics We analyze the characteristics of prompts constructed with different demonstration selection methods in Tab. 5. Symbol Coverage shows the average fraction of symbols in y_{test} that are covered by the demonstration set, and similarly LS Coverage the fraction of covered LSs. While symbol coverage is generally high across all methods when using 24 demonstrations, LS coverage is significantly higher in Cover-LS, suggesting that only covering relevant symbols in prompts isn't as efficient as covering LSs. Utterance Similarity measures average cosine similarity between SBERT embeddings of the test utterance and prompt utterances, which is highest for Top-K as expected.

---

[3]To comply with Bogin et al. (2022), we measure ULS only for structures up to size 4.

Prompt Metrics	GeoQuery TMCD			SMCalFlow-CS 8-C		
	Top-K	Cover-LS	DPP	Top-K	Cover-LS	DPP
Symbol Coverage	97.2	99.3	99.2	93.1	95.0	96.6
LS Coverage	69.2	73.0	71.0	70.0	86.8	76.1
Utterance Sim.	0.46	0.42	0.43	0.50	0.47	0.48
No. Unique LSs	306	505	484	2139	3647	4212

Table 5: Prompt metrics: coverage, similarity, and diversity in prompts with 24 demonstrations.

To approximate diversity between demonstrations, we calculate the average number of unique LSs in demonstrations, and observe it is substantially higher in Cover-LS and DPP compared to Top-K. This implies structural coverage and diversity are more important than input similarity in compositional splits.

Robustness to retrieval methods To assess our method's robustness, we test how sensitive it is to the chosen retriever in the NoFT setup. First, we use our default retrievers, which are BM25 over utterance words (BM25-Utterance), and BM25 over predicted program symbols (BM25-Predicted). We add a random retriever that is identical to the RANDOM baseline introduced in §4.3 when combined with Top-K. We also evaluate the SBERT retriever (Reimers and Gurevych, 2019), which encodes input utterances and measures the cosine similarity between pairs of encodings. As seen in Fig. 5, Cover-LS outperforms Top-K in all settings by a significant margin. Moreover, while BM25-Utterance performs best, variance across retrievers is low for Cover-LS, but higher for Top-K.

5 Related Work

Example selection One of the central issues in in-context learning is the selection of examples, which can either be based on parameter-free retrievers (Wang et al., 2022; Zemlyanskiy et al., 2022) or neural-based retrievers (Pasupat et al., 2021; Liu et al., 2022; Rubin et al., 2022). These studies consider each example separately, which often leads to a lack of coverage and diversity.

Our approach is similar to the retrieval procedure in Zemlyanskiy et al. (2022), which makes a preliminary prediction and retrieves demonstrations with similar programs. However, while they use classic tf-idf with predicted tokens, we use predicted local structures and aim to cover them.

Some studies encourage diverse example selection regardless of prompting. To address multi-answer retrieval, Nandigam et al. (2022) employ
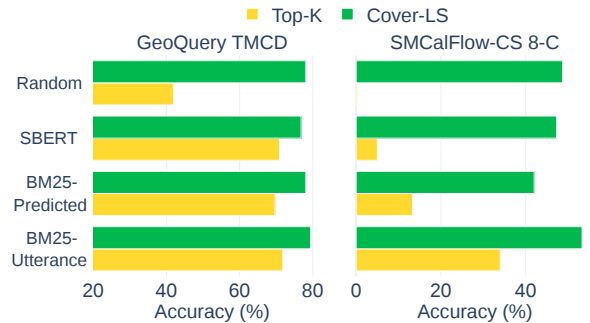

Figure 5: Comparing model accuracy across different retrievers, with demonstrations selected using Top-K or Cover-LS.

DPP, and Min et al. (2021) autoregressively select instances based on previous selections. Other works include Su et al. (2022), which selects instances with varying confidence scores for annotation and (concurrent work) Ye et al. (2022) who propose a MMR-based selection strategy.

In-context learning for compositional generalization There have been previous attempts to address compositional generalization problems using LLMs equipped with demonstrations. When selecting demonstrations, some also consider target coverage or structure similarity, but only in oracle setups (Hosseini et al., 2022; Qiu et al., 2022b). Drozdov et al. (2022) try to cover the syntactic parse tree constituents with demonstrations but rely heavily on manually-picked examples.

6 Conclusion

In this paper, we studied how to leverage ICL to improve compositional generalization in semantic parsing, by increasing diversity among demonstrations. We found that choosing demonstrations that cover the structures required in the output program substantially improves performance across three compositional semantic parsing datasets in the pure in-context learning setup and when combined with finetuning. We further demonstrated that by aiming for structural coverage, we can reduce the number of demonstrations needed for generalization, and improve test performance on hard examples. Our approach can be applied to a wide range of NLP tasks where demonstrations should cover complementary aspects of the task, and we hope it will encourage further exploration of our method to improve generalization across diverse applications.

Limitations

Demonstration selection methods We assume that diversity can be obtained by choosing demonstrations with different program structures. This is based on previous work that demonstrated the importance of diversifying program structures in semantic parsing tasks (Oren et al., 2021; Bogin et al., 2022; Gupta et al., 2022). We also try to diversify utterance words or program symbols but do not consider more complex utterance features that could be applied to a wider range of language understating tasks.

We also assume that recall matters more than precision when designing Cover-LS algorithm. That means we aim to choose a set of demonstrations that covers every predicted local structure in $S_{\tilde{y}_{test}}$, since it has the potential to be a correct one. We do not predict whether a specific structure should be covered. Furthermore, our approach for increasing gold structure coverage by using additional beam candidates could be improved by employing search methods specifically targeted for diversity (Meister et al., 2021; Narayan et al., 2022).

Retrievers We used different retrievers for NoFT and FT setups based on the retriever that worked best on the development set. Future research should be conducted to understand why different retrievers are preferred in different setups. A potential method could be to consider both input utterances and programs for retrieval, as suggested in Zemlyanskiy et al. (2022).

Ethics Statement

In this work, we studied methods for choosing diverse demonstrations to improve in-context compositional generalization in semantic parsing. We have only evaluated our methods on semantic parsing datasets in English. It is our hope, however, that improvements in compositional generalization will eventually allow systems to generalize better to languages that are not well represented in small training sets.

Acknowledgements

References

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571.

Ben Bogin, Shivanshu Gupta, and Jonathan Berant. 2022. Unobserved local structures make compositional generalization hard. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2731–2747, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Dorian Brown. 2020. Rank-BM25: A Collection of BM25 Algorithms in Python.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher 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 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.*

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, S. Arun Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan

Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *ArXiv preprint*, abs/2107.03374.

Yanda Chen, Ruiqi Zhong, Sheng Zha, George Karypis, and He He. 2022. Meta-learning via language model in-context tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 719–730, Dublin, Ireland. Association for Computational Linguistics.

Henry Conklin, Bailin Wang, Kenny Smith, and Ivan Titov. 2021. Meta-learning to compositionally generalize. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3322–3335, Online. Association for Computational Linguistics.

Andrew Drozdov, Nathanael Scharli, Ekin Akyuurek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. 2022. Compositional semantic parsing with large language models. *ArXiv preprint*, abs/2209.15003.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR.

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

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6, Melbourne, Australia. Association for Computational Linguistics.

Shivanshu Gupta, Sameer Singh, and Matt Gardner. 2022. Structurally diverse sampling for sample-efficient training and comprehensive evaluation. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 4966–4979, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2019. Don't paraphrase, detect! rapid and effective data collection for semantic parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3810–3820, Hong Kong, China. Association for Computational Linguistics.

Arian Hosseini, Ankit Vani, Dzmitry Bahdanau, Alessandro Sordoni, and Aaron Courville. 2022. On the compositional generalization gap of in-context learning. In *Proceedings of the Fifth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 272–280, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

Dieuwke Hupkes, Mario Giulianelli, Verna Dankers, Mikel Artetxe, Yanai Elazar, Tiago Pimentel, Christos Christodoulopoulos, Karim Lasri, Naomi Saphra, Arabella Sinclair, Dennis Ulmer, Florian Schottmann, Khuyagbaatar Batsuren, Kaiser Sun, Koustuv Sinha, Leila Khalatbari, Maria Ryskina, Rita Frieske, Ryan Cotterell, and Zhijing Jin. 2022. State-of-the-art generalisation research in NLP: a taxonomy and review. *ArXiv preprint*, abs/2210.03050.

Vishal Kaushal, Ganesh Ramakrishnan, and Rishabh K. Iyer. 2022. Submodlib: A submodular optimization library. *ArXiv preprint*, abs/2202.10680.

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.

Alex Kulesza and Ben Taskar. 2012. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286.

Brenden M. Lake. 2019. Compositional generalization through meta sequence-to-sequence learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9788–9798.

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.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2022. What makes good in-context examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 100–114, Dublin, Ireland and Online. Association for Computational Linguistics.

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.

Clara Meister, Martina Forster, and Ryan Cotterell. 2021. Determinantal beam search. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6551–6562, Online. Association for Computational Linguistics.

Joram Meron. 2022. Simplifying semantic annotations of SMCalFlow. In *Proceedings of the 18th Joint ACL - ISO Workshop on Interoperable Semantic Annotation within LREC2022*, pages 81–85, Marseille, France. European Language Resources Association.

Sewon Min, Kenton Lee, Ming-Wei Chang, Kristina Toutanova, and Hannaneh Hajishirzi. 2021. Joint passage ranking for diverse multi-answer retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6997–7008, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2022. MetaICL: Learning to learn in context. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2791–2809, Seattle, United States. Association for Computational Linguistics.

Poojitha Nandigam, Nikhil Rayaprolu, and Manish Shrivastava. 2022. Diverse multi-answer retrieval with determinantal point processes. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 2220–2225, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Shashi Narayan, Gonçalo Simões, Yao Zhao, Joshua Maynez, Dipanjan Das, Michael Collins, and Mirella Lapata. 2022. A well-composed text is half done! composition sampling for diverse conditional generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1319–1339, Dublin, Ireland. Association for Computational Linguistics.

Inbar Oren, Jonathan Herzig, and Jonathan Berant. 2021. Finding needles in a haystack: Sampling structurally-diverse training sets from synthetic data for compositional generalization. In *Proceedings of the 2021*

Conference on Empirical Methods in Natural Language Processing, pages 10793–10809, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. 2022. Training language models to follow instructions with human feedback. *ArXiv preprint*, abs/2203.02155.

Panupong Pasupat, Yuan Zhang, and Kelvin Guu. 2021. Controllable semantic parsing via retrieval augmentation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7683–7698, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Pawel Nowak, Tal Linzen, Fei Sha, and Kristina Toutanova. 2022a. Improving compositional generalization with latent structure and data augmentation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4341–4362, Seattle, United States. Association for Computational Linguistics.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Tianze Shi, Jonathan Herzig, Emily Pitler, Fei Sha, and Kristina Toutanova. 2022b. Evaluating the impact of model scale for compositional generalization in semantic parsing. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9157–9179, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2022. Learning to retrieve prompts for in-context learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2671, Seattle, United States. Association for Computational Linguistics.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 922–938, Online. Association for Computational Linguistics.

Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2022. Selective annotation makes language models better few-shot learners. *ArXiv preprint*, abs/2209.01975.

Lappoon R. Tang and Raymond J. Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *ECML*.

Shuohang Wang, Yichong Xu, Yuwei Fang, Yang Liu, Siqi Sun, Ruochen Xu, Chenguang Zhu, and Michael Zeng. 2022. Training data is more valuable than you think: A simple and effective method by retrieving from training data. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3170–3179, Dublin, Ireland. Association for Computational Linguistics.

Xi Ye, Srini Iyer, Asli Celikyilmaz, Ves Stoyanov, Greg Durrett, and Ramakanth Pasunuru. 2022. Complementary explanations for effective in-context learning. *ArXiv preprint*, abs/2211.13892.

Pengcheng Yin, Hao Fang, Graham Neubig, Adam Pauls, Emmanouil Antonios Platanios, Yu Su, Sam Thomson, and Jacob Andreas. 2021. Compositional generalization for neural semantic parsing via span-level supervised attention. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2810–2823, Online. Association for Computational Linguistics.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI, Vol. 2*.

Yury Zemlyanskiy, Michiel de Jong, Joshua Ainslie, Panupong Pasupat, Peter Shaw, Linlu Qiu, Sumit Sanghai, and Fei Sha. 2022. Generate-and-retrieve: Use your predictions to improve retrieval for semantic parsing. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 4946–4951, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

A Additional Analysis

Error analysis We analyze errors (NoFT setup) and show results in Tab. 6. Inspired by the metrics in Qiu et al. (2022b), we automatically compute statistics for the following cases when the prediction is wrong: (1) Syntax Errors, when the model produces a program with invalid parentheses; (2) Over-Copying, when the entire prediction has the same anonymized form as one of the demonstrations; (3) OOV (out-of-vocabulary) Hallucination, where the anonymized predicted program contains a symbol missing from the gold program or any prompt demonstration; and (4) Missing Symbol(s), where the predicted program is missing at least one symbol.

The distribution of errors is similar across demonstration selection methods. Syntax errors are rare in both datasets. Many predictions are over-copied, especially in SMCalFlow-CS, but when diversity is increased with DPP, this number decreases significantly. Surprisingly, despite having a smaller vocabulary, GeoQuery has more out-of-vocabulary hallucinations. Almost all incorrect predictions have a missing symbol, but Top-K predictions are especially prone to this type of error.

Change of retriever in FT setup Tab. 7 shows results for the FT setup when using BM25 over lower-cased utterance words as retriever, instead of BM25 over predicted program symbols.

B Local Structures

We follow the definition of local structures from Bogin et al. (2022), which were defined for structures of sizes 2-4, and extend them to local structures of any size. Given a program y, we parse it into a tree $T = (\mathcal{V}, \mathcal{E})$, such that each node $v \in \mathcal{V}$ is labeled by the program symbol (function or value) that it represents in y (or a special symbol for the root node), and the set of edges $\mathcal{E} = \{(p, c)\}$ expresses **parent-child** relations between the nodes.

We capture sibling relations by defining a graph based on the tree T that contains an edge set $\mathcal{E}_{\text{sib}}$ of **sibling** edges: $G = (\mathcal{V}, \mathcal{E} \cup \mathcal{E}_{\text{sib}})$. Specifically, for each parent node p, the program y induces an order over the children of p: $(c_1^p, ..., c_{N_p}^p)$, where N_p is the number of children. We then define $\mathcal{E}_{\text{sib}} = \bigcup_p \{c_i^p, c_{i+1}^p\}_{i=1}^{N_p}$, that is, all *consecutive* siblings will be connected by edges.

We define a local structure of size n as the subset G_{LS} of all connected sub-graphs of size n in G

Error Types	GeoQuery TMCD			SMCalFlow-CS 8-C		
	Top-K	Cover-LS	DPP	Top-K	Cover-LS	DPP
Syntax Error	1.0	0.0	0.9	5.0	2.9	9.5
Over-Copying	19.8	16.9	15.8	41.4	41.4	10.7
OOV Hallucination	20.0	17.8	22.9	8.0	3.5	5.4
Missing Symbol(s)	88.7	75.2	77.9	87.4	77.7	79.8

Table 6: Error analysis. We automatically compute the fraction of different error types.

such that for every pair (x, y) of nodes in G_{LS} it holds that $(x, y) \in \mathcal{E}_{\text{sib}}$ iff x and y are both leaves in G_{LS}. That is, informally, the relations between nodes in the the sub-graph include parent-child and siblings, but not e.g. cousins or uncles. All program symbols are local structures of size 1. Tab. 8 shows a partial list of local structures for a given program.

B.1 Fixes for Local Structure Extraction

We try to fix syntax errors in the predictions made using the auxiliary model to enable parsing them to ASTs and extraction of LSs. We add or remove closing parentheses based on the number of missing or redundant parentheses at the end of the program.

C Dataset Details

We provide representative examples of the datasets used in this work in Tab. 1 and Tab. 9. We report dataset sizes in Tab. 10. Due to conversion errors, SMCalFlow-CS Simple has fewer training examples than SMCalFlow-CS. However, those missing examples are not cross-domain examples.

We used publicly available datasets from previous peer-reviewed studies. Those datasets do not contain any information that uniquely identifies individual people or offensive content. The COVR-10 dataset is completely synthetic. The GeoQuery dataset contains only basic information about U.S. geography. SMCalflow-CS contains crowd-sourced queries collected in a simulated environment.

D Prompt Format and Examples

We add special prefixes "source:" and "target:" for retrieved source-target pairs and separate them with break lines. Tab. 11 shows prompt examples for different demonstration selection methods, where the only prompt that contains all the required program symbols and produces the correct prediction is Cover-LS's prompt.

Training Method	Test Method	GeoQuery				SMCalFlow-CS Simple				COVR-10
		i.i.d.	Templ.	TMCD	Len.	i.i.d.	8-C	16-C	32-C	
Random	Top-K	93.0	84.9	76.1	40.3	82.9	26.7	41.0	53.9	23.1
Cover-LS$_1$	Cover-LS$_1$	**93.3**	85.7	76.3	42.2	**83.2**	**31.9**	**48.6**	**61.5**	28.3
Cover-LS$_1$	Cover-LS	93.2	**85.8**	**76.6**	**42.4**	**83.2**	28.3	46.6	60.9	**30.1**
Cover-LS$_2$	Cover-LS	92.5	85.2	75.1	39.7	83.9	27.2	45.5	59.5	29.8
Cover-LS	Cover-LS	91.4	81.0	69.1	39.2	82.7	17.5	31.5	55.1	12.3

Table 7: **FT results** using T5. Same setup as in Tab. 4, except we use BM25 over lower-cased utterance words.

E   DPP Details

DPPs are probabilistic models that are effective at modeling a distribution on all the subsets of the ground set $\mathcal{T}$ jointly considering the quality and diversity. A subset $\mathcal{D}$ is drawn according to the probability distribution $\mathcal{P}$:

$$\mathcal{P}(\mathcal{D} \subset \mathcal{T}; L) \propto \det(L_{\mathcal{D}}) \qquad (1)$$

Where $L \in \mathbb{R}^{n \times n}$ is a PSD matrix and $L_{\mathcal{D}}$ is the submatrix of L indexed by items in $\mathcal{D}$. L matrix takes into account the quality of each training example and its similarity to other training examples through:

$$L_{ij} = q_i \phi_i^\top \phi_j q_j \qquad (2)$$

with $q \in \mathbb{R}^n$ being normalized retriever scores that model the quality of each example; and $\{\phi_i\}_{i=1}^n$ denoting normalized tf-idf vectors over LSs, which model the different aspects that are contained within each training example. The dot product of those vectors is used to model the similarity between two train examples.

$\log \det(L_{\mathcal{D}})$ is a submodular function which satisfies the diminishing marginal returns property. Therefore, we can find a subset of training examples $\mathcal{D} \subset \mathcal{T}, |\mathcal{D}| = k$ that maximizes it in a feasible manner using a greedy optimizer (Kaushal et al., 2022). Specifically, we used the Naive Greedy optimizer. We used scikit-learn (Pedregosa et al., 2011) for calculating tf-idf vectors.

F   Finetuning Details

We provide implementation details for finetuning experiments (we use the same configuration for all FT experiments and training of the auxiliary model). We finetune the T5-large model (770 million parameters) with the AdamW optimizer (Loshchilov and Hutter, 2019) and a learning rate of $1e^{-5}$. We use a polynomial decay learning rate with an ending rate of $1e^{-6}$, and 100 warmup

steps. We train for 250/50/70 epochs and evaluate on the validation set every 3/5/10 epochs for Geo/SMCalFlow (both versions)/COVR respectively. We use batches of size 8 for all datasets (and gradient accumulation in case batch cannot fit in memory). We used a single GPU for each T5-large finetuning experiment: Nvidia GeForce RTX 3090 when training on GeoQuery and COVR-10, and A100 (80GB) for SMCalFlow-CS and SMCalFlow-CS Simple. GeoQuery experiments with prompts trained for an average of 2 hours, COVR for 8 hours, and SMCalFlow-CS Simple for 41 hours.

We use the AllenNLP library (Gardner et al., 2018) for training and evaluation. We use Rank-BM25 (Brown, 2020) as a BM25 implementation.

Standard deviation   We report standard deviation results in the FT setup in Tab. 13. Results are computed across 3 random seeds.

G   NoFT Details

All NoFT experiments were conducted using the OpenAI inference API with the sampling temperature set to 0. Our setup requires a single API call per test instance. The total number of API calls is estimated at 160K.

Standard deviation   We report standard deviation results in NoFT setup in Tab. 12. Results are computed using 3 random seeds for a subset of 100 test examples.

Tuning the number of beam candidates   We use the development set to tune the number of beam candidates B when predicting local structures. Tab. 14 shows the results of using different values of B in NoFT setup on a random subset of 100 development examples. Prompts are constructed using Cover-LS with k = 8 demonstrations.

H   Artifact Licensing

We include license information for all artifacts used in this work in Tab. 15. Our use of artifacts was

consistent with their intended purpose when it was specified.

I GenBench Evaluation Card

Our GenBench (Hupkes et al., 2022) evaluation card is presented in Fig. 6.

Dataset	SMCalFlow-CS Simple
Utterance	*Create a new meeting on Friday called Work on Project.*
Program	CreateEvent (AND (has_subject ("Work on Project"), starts_at (NextDOW ("Friday"))))
Anonymized Program	CreateEvent (AND (has_subject (string), starts_at (NextDOW (string))))

Size	Local structures
1	CreateEvent AND has_subject string starts_at NextDOW
2	<root> → CreateEvent CreateEvent → AND AND → has_subject AND → starts_at has_subject ↔ starts_at has_subject → string starts_at → NextDOW NextDOW → string
3	<root> → CreateEvent → AND CreateEvent → AND → has_subject CreateEvent → AND → starts_at AND → has_subject ↔ starts_at AND → has_subject → string AND → starts_at → NextDOW starts_at → NextDOW → string
	⋮
6	<root> → CreateEvent → AND → starts_at → NextDOW → string

Table 8: Local structures of different sizes for a specific example (→ denotes parent-child relations, ↔ denotes sibling relations)

Utterance	*Can you make a meeting with David Lax's reports ?*

Version	Program
v1 (LISP)	(Yield :output (CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event] :attendees (AttendeeListHasPeople :people (FindReports :recipient (Execute :intension (refer (extensionConstraint (RecipientWithNameLike :constraint (Constraint[Recipient]) :name # (PersonName "David Lax")))))))))))
v2 (LISPRESS)	(Yield (CreateCommitEventWrapper (CreatePreflightEventWrapper (Event.attendees_? (AttendeeListHasPeople (FindReports (Execute (refer (extensionConstraint (RecipientWithNameLike (^ (Recipient) EmptyStructConstraint) (PersonName.apply "David Lax")))))))))))

Table 9: An example from each version of SMCalFlow-CS dataset.

Dataset	Split	Train	Development	Test
GeoQuery	Standard	600	-	280
	Template1	438	110	332
	Template2	439	110	331
	Template3	440	110	330
	TMCD1	440	110	330
	TMCD2	440	110	330
	TMCD3	440	110	330
	Length	440	110	330
SMCalFlow-CS v1	8-S	25412	662	662
	0-C	25404	662	663
	8-C	25412	662	663
	16-C	25420	662	663
	32-C	25436	662	663
SMCalFlow-CS v2	8-S	20965	360	360
	0-C	20957	360	360
	8-C	20965	360	360
	16-C	20973	360	360
	32-C	20989	360	360
SMCalFlow-CS Simple	8-S	25402	662	662
	8-C	25402	662	663
	16-C	25410	662	663
	32-C	25426	662	662
COVR-10	Each split	3000	-	500

Table 10: Dataset sizes

Motivation			
Practical	*Cognitive*	*Intrinsic*	*Fairness*
All			
Generalisation type					
Compositional	*Structural*	*Cross Task*	*Cross Language*	*Cross Domain*	*Robustness*
All					
Shift type			
Covariate	*Label*	*Full*	*Assumed*
All			
Shift source			
Naturally occurring	*Partitioned natural*	*Generated shift*	*Fully generated*
	GeoQuery		COVR-10
	SMCalFlow-CS		
Shift locus			
Train–test	*Finetune train–test*	*Pretrain–train*	*Pretrain–test*
	All		

Figure 6: GenBench (Hupkes et al., 2022) evaluation card.

Dataset	GeoQuery
Utterance	*through which states does the longest river* `in texas` *run*
Gold Program	`answer (state (traverse_1 (longest (river` `(loc_2 (stateid (string)))))))`

Selection Method	Prompt
Top-K	source: *which states does the mississippi river run through*
	target: `answer (state (traverse_1 (river (riverid (string)))))`
	source: *which states does the colorado river run through*
	target: `answer (state (traverse_1 (river (riverid (string)))))`
	source: *which states does the missouri river run through*
	target: `answer (state (traverse_1 (river (riverid (string)))))`
	source: *which states does the longest river run through*
	target: `answer (state (traverse_1 (longest (river` `(all)))))`
	source: *through which states does the longest river in texas run*
	target:
DPP	source: *what states does the shortest river run through*
	target: answer (state (traverse_1 (shortest (river (all)))))
	source: *which states does the mississippi run through*
	target: answer (state (traverse_1 (riverid (string)))))
	source: *which states does the missouri river run through*
	target: `answer (state (traverse_1 (river (riverid (string)))))`
	source: *which states does the longest river run through*
	target: `answer (state (traverse_1 (longest (river` `(all)))))`
	source: *through which states does the longest river in texas run*
	target:
Cover-LS	source: *what state borders the least states excluding alaska and excluding hawaii*
	target: answer (fewest (state (next_to_2 (exclude (exclude (state (all)), stateid (string)), stateid (string))))))
	source: *what is the longest river* `in texas`
	target: answer (longest (river `(loc_2 (stateid (string)))))`
	source: *which states does the missouri river run through*
	target: `answer (state (traverse_1 (river (riverid (string)))))`
	source: *which states does the longest river run through*
	target: `answer (state (traverse_1 (longest (river` `(all)))))`
	source: *through which states does the longest river in texas run*
	target:

Table 11: Prompts produced with different demonstration selection methods for a specific test example. Each prompt contains $k = 4$ demonstrations.

	GeoQuery				SMCalFlow-CS					COVR-10
	i.i.d.	Templ.	TMCD	Len.	i.i.d.	0-C	8-C	16-C	32-C	
Random	1.5	6.6	2.5	5.0	4.6	0.6	0.6	0.6	3.5	3.1
Top-K	1.5	1.8	1.0	1.1	0.6	1.0	1.0	1.1	1.1	4.6
Cover-Utt	1.0	1.2	1.2	2.1	1.5	1.5	1.0	1.2	2.1	1.9
DPP	0.0	0.5	1.7	1.5	1.2	0.6	1.0	1.0	3.1	2.0
Cover-LS	1.5	1.1	2.4	2.1	1.4	0.6	1.1	0.6	3.5	4.2

Table 12: Standard deviation results in NoFT setup. Results are computed on a random subset of 100 test examples across 3 random seeds.

Training Method	Test Method	GeoQuery				SMCalFlow-CS Simple				COVR-10
		i.i.d.	Templ.	TMCD	Len.	i.i.d.	8-C	16-C	32-C	
T5 (fine tuned w/o prompts)	-	0.2	0.8	1.6	0.5	0.7	1.4	4.6	1.5	1.7
Random	Random	0.0	1.2	1.0	0.9	0.3	3.2	2.7	0.4	2.7
Random	Top-K	0.2	1.4	1.3	2.3	0.4	3.3	1.2	1.2	2.7
Top-K	Top-K	0.6	3.5	2.1	0.7	0.3	1.9	1.9	1.3	3.9
Cover-LS$_1$	Cover-LS$_1$	0.6	0.8	0.9	2.6	0.5	2.0	0.2	1.7	4.8
Cover-LS$_1$	Cover-LS	0.5	0.4	0.9	4.2	0.4	1.4	0.8	0.8	6.5
Cover-LS$_1$	Cover-LS (Oracle)	0.2	0.7	0.9	2.6	0.3	0.6	0.6	0.8	12.1

Table 13: Standard deviation results in FT setup. Results are computed across 3 random seeds.

	GeoQuery							SMCalFlow-CS				
B	Templ. 1	Templ. 2	Templ. 3	TMCD 1	TMCD 2	TMCD 3	Len.	i.i.d.	0-C	8-C	16-C	32-C
1	85	74	77	66	65	84	62	73	0	36	47	63
3	85	75	75	69	59	88	60	65	0	42	49	67
5	84	76	72	69	64	87	60	64	1	44	51	68

Table 14: The effect of number of beam candidates B on accuracy in NoFT setup. Prompts are constructed using Cover-LS with $k = 8$ demonstrations. Results are computed on a random subset of 100 development examples (single seed).

Artifact	License	Reference
Models		
T5	Apache 2.0	HF model card
Codex	API usage policy	API documentation
Dataset		
GeoQuery	GPL 2.0	Official website
GeoQuery compositional splits	Apache 2.0	Github repository
SMCalFlow-CS	MIT	Github repository
SMCalFlow Simple	MIT	Github repository
COVR-10	MIT	Github repository
Tools		
AllenNLP	Apache 2.0	Github repository
Rank-BM25	Apache 2.0	Github repository
SBERT	Apache 2.0	Github repository
DPP optimization	MIT	Github repository

Table 15: License information for all artifacts

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Limitations

☑ A2. Did you discuss any potential risks of your work?
Ethics Statement

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Abstract and section 1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Section 4

☑ B1. Did you cite the creators of artifacts you used?
Section 4

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Appendix H

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Appendix H

☑ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Appendix C

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Ethics Statement, Section 4.1

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Appendix C

C ☑ Did you run computational experiments?

Section 4

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Appendices F - G

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 4, Appendices F - G

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Section 4, Appendices F - G

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Appendices D - G

D   ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.