Improving Compositional Generalization in Semantic Parsing

Inbar Oren[1] **Jonathan Herzig**[*,1] **Nitish Gupta**[*,2] **Matt Gardner**[3] **Jonathan Berant**[1,3]

[1]School of Computer Science, Tel-Aviv University
[2]University of Pennsylvania
[3]Allen Institute for AI

{inbaroren@mail, jonathan.herzig@cs, joberant@cs}.tau.ac.il,
nitishg@seas.upenn.edu, mattg@allenai.org

Abstract

Generalization of models to out-of-distribution (OOD) data has captured tremendous attention recently. Specifically, *compositional generalization*, i.e., whether a model generalizes to new structures built of components observed during training, has sparked substantial interest. In this work, we investigate compositional generalization in semantic parsing, a natural test-bed for compositional generalization, as output programs are constructed from sub-components. We analyze a wide variety of models and propose multiple extensions to the attention module of the semantic parser, aiming to improve compositional generalization. We find that the following factors improve compositional generalization: (a) using contextual representations, such as ELMo and BERT, (b) informing the decoder what input tokens have previously been attended to, (c) training the decoder attention to agree with pre-computed token alignments, and (d) downsampling examples corresponding to frequent program templates. While we substantially reduce the gap between in-distribution and OOD generalization, performance on OOD compositions is still substantially lower.

1 Introduction

Neural models trained on large datasets have recently shown great performance on data sampled from the training distribution. However, generalization to out-of-distribution (OOD) scenarios has been dramatically lower (Sagawa et al., 2019; Gardner et al., 2020; Kaushik et al., 2020). A particularly interesting case of OOD generalization is *compositional generalization*, the ability to systematically generalize to test examples composed of components seen during training. For example, we expect a model that observed the questions *"What is the capital of France?"* and *"What is the population of Spain?"* at training time to generalize to questions such as *"What is the population of the capital of Spain?"*. While humans generalize systematically to such compositions (Fodor et al., 1988), models often fail to capture the structure underlying the problem, and thus miserably fail (Atzmon et al., 2016; Lake and Baroni, 2018; Loula et al., 2018; Bahdanau et al., 2019b; Ruis et al., 2020).

Semantic parsing, mapping natural language utterances to structured programs, is a task where compositional generalization is expected, as substructures in the input utterance and output program often align. For example, in *"What is the capital of the largest US state?"*, the span *"largest US state"* might correspond to an `argmax` clause in the output program. Nevertheless, prior work (Finegan-Dollak et al., 2018; Herzig and Berant, 2019; Keysers et al., 2020) has shown that data splits that require generalizing to new program templates result in drastic loss of performance. However, past work did not investigate how different modeling choices interact with compositional generalization.

In this paper, we thoroughly analyze the impact of different modeling choices on compositional generalization in 5 semantic parsing datasets—four that are text-to-SQL datasets, and DROP, a dataset for executing programs over text paragraphs. Following Finegan-Dollak et al. (2018), we examine performance on a *compositional split*, where target programs are partitioned into "program templates", and templates appearing at test time are unobserved at training time. We examine the effect of standard practices, such as contextualized representations (§3.1) and grammar-based decoding (§3.2). Moreover, we propose novel extensions to decoder attention (§3.3), the component responsible for aligning sub-structures in the question and program: (a) supervising attention based on pre-computed token alignments, (b) attending over constituency spans, and (c) encouraging the decoder

[*] The authors contributed equally.

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 2482–2495
November 16 - 20, 2020.

attention to cover the entire input utterance. Lastly, we also propose downsampling examples from frequent templates to reduce dataset bias (§3.4).

Our main findings are that (i) contextualized representations, (ii) supervising the decoder attention, (iii) informing the decoder on coverage of the input by the attention mechanism, and (iv) downsampling frequent program templates, all reduce the gap in generalization when comparing standard iid splits to compositional splits. For SQL, the gap in exact match accuracy between in-distribution and OOD is reduced from 84.6 → 62.2 and for DROP from 96.4 → 77.1. While this is a substantial improvement, the gap between in-distribution and OOD generalization is still significant. All our code and data are publicly available at http://github.com/inbaroren/improving-compgen-in-semparse.

2 Compositional Generalization

Natural language is compositional in a sense that complex utterances are interpreted by understanding the structure of the utterance and the meaning of its parts (Montague, 1973). For example, the meaning of *"a person below the tree"* can be composed from the meaning of *"a person"*, *"below"* and *"tree"*. By virtue of compositionality, an agent can derive the meaning of new utterances, even at first encounter. Thus, we expect our systems to model this compositional nature of language and generalize to new utterances, generated from subparts observed during training but composed in novel ways. This sort of model generalization is often called *compositional generalization*.

Recent work has proposed various benchmarks to measure different aspects of *compositional generalization*, showing that current models struggle in this setup. Lake and Baroni (2018) introduce a benchmark called SCAN for mapping a command to actions in a synthetic language, and proposed a data split that requires generalizing to commands that map to a longer sequence of actions than observed during training. Bahdanau et al. (2019a) study the impact of modularity in neural models on the ability to answer visual questions about pairs of objects that were not observed during training. Bahdanau et al. (2019b) assess the ability of models trained on CLEVR (Johnson et al., 2017) to interpret new referring expressions composed of parts observed at training time. Keysers et al. (2020) develop a benchmark of Freebase questions and propose a data split such that the test set contains new combinations of knowledge-base constants (entities and relations) that were not seen during training. Ruis et al. (2020) proposed gSCAN, which focuses on compositional generalization when mapping commands to actions in a situated environment.

Program	Question	iid split	Program split
`select distinct river.length from river where rive.name = "river_name0"`	*What length is* `river_name0`*?*	*train*	*train*
	How long is `river_name0`*?*	test	*train*
`select state.name from state where state.area = (select max (state.area) from state)`	*Give me the largest state*	*train*	test
	What state has the largest area?	test	test

Figure 1: An iid split of examples in semantic parsing leads to identical anonymized programs appearing at both training and test time. A program split prohibits anonymized programs from appearing in the same partition, and hence tests compositional generalization.

In this work, we focus on a specific kind of compositional data split, proposed by Finegan-Dollak et al. (2018), that despite its simplicity leads to large drops in performance. Finegan-Dollak et al. (2018) propose to split semantic parsing data such that a model cannot memorize a mapping from question templates to programs. To achieve this, they take question-program pairs, and *anonymize* the entities in the question-program pair with typed variables. Thus, questions that require the same abstract reasoning structure now get mapped to the same anonymized program, referred to as *program template*. For example, in the top two rows of Figure 1, after anonymizing the name of a river to the typed variable `river_name0`, two lexically-different questions map to the same program template. Similarly, in the bottom two rows we see two different questions that map to the same program even before anonymization.

The data is then split in a manner such that a program template and all its accompanying questions belong to the same set, called the *program split*. This ensures that all test-set program templates are unobserved during training. For example, in a *iid split* of the data, it is possible that the question *"what is the capital of France?"* will appear in the training set, and the question *"Name Spain's capital."* will appear in the test set. Thus, the model only needs to memorize a mapping from question

templates to program templates. However, in the *program split*, each program template is in either the training set or test set, and thus a model must generalize at test time to new combinations of predicates and entities (see Figure 1 - Program split).

We perform the compositional split proposed by Finegan-Dollak et al. (2018) on four text-to-SQL datasets from Finegan-Dollak et al. (2018) and one dataset for mapping questions to QDMR programs (Wolfson et al., 2020) on DROP (Dua et al., 2019). Exact experimental details are in §4.

3 Model

Finegan-Dollak et al. (2018) convincingly showed that a program split leads to low semantic parsing performance. However, they examined only a simple baseline parser, disregarding many standard variations that have been shown to improve in-distribution generalization, and might affect OOD generalization as well. In this section, we describe variants to both the model and training, and evaluate their effect on generalization in §5. We examine well-known choices, such as the effect of contextualized representations (§3.1) and grammar-based decoding (§3.2), as well as several novel extensions to the decoder attention (§3.3), which include (a) eliciting supervision (automatically) for the decoder attention distribution, (b) allowing attention over question spans, and (c) encouraging attention to cover all of the question tokens. For DROP, where the distribution over program templates is skewed, we also examine the effect of reducing this bias by downsampling frequent program templates (§3.4).

Baseline Semantic Parser A semantic parser maps an input question x into a program z, and in the supervised setup is trained from (x, z) pairs. Similar to Finegan-Dollak et al. (2018), our baseline semantic parser is a standard sequence-to-sequence model (Dong and Lapata, 2016) that encodes the question x with a BiLSTM encoder (Hochreiter and Schmidhuber, 1997) over GloVe embeddings (Pennington et al., 2014), and decodes the program z token-by-token from left to right with an attention-based LSTM decoder (Bahdanau et al., 2015).

3.1 Contextualized Representations

Pre-trained contextualized representations revolutionized natural language processing in recent years, and semantic parsing has been no exception (Guo et al., 2019; Wang et al., 2019). We hypothesize that better representations for question tokens should improve compositional generalization, because they reduce language variability and thus may help improve the mapping from input to output tokens. We evaluate the effect of using ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019) to represent question tokens.[1]

3.2 Grammar-Based Decoding

A unique property of semantic parsing, compared to other generation tasks, is that programs have a clear hierarchical structure that is based on the target formal language. Decoding the output program token-by-token from left to right (Dong and Lapata, 2016; Jia and Liang, 2016) can thus generate programs that are not syntactically valid, and the model must effectively learn the syntax of the target language at training time. Grammar-based decoding resolves this issue and has been shown to consistently improve in-distribution performance (Rabinovich et al., 2017; Krishnamurthy et al., 2017; Yin and Neubig, 2017). In grammar-based decoding, the decoder outputs the abstract syntax tree of the program based on a formal grammar of the target language. At each step, a production rule from the grammar is chosen, eventually outputting a top-down left-to-right linearization of the program tree. Because decoding is constrained by the grammar, the model outputs only valid programs. We refer the reader to the aforementioned papers for details on grammar-based decoding.

Compositional generalization involves combining known sub-structures in novel ways. In grammar-based decoding, the structure of the output program is explicitly generated, and this could potentially help compositional generalization. We discuss the grammars used in this work in §4.

3.3 Decoder Attention

Semantic parsers use attention-based decoding: at every decoding step, the model computes a distribution $(p_1 \dots p_n)$ over the question tokens $x = (x_1, \dots, x_n)$ and the decoder computes its next prediction based on the weighted average $\sum_{i=1}^{n} p_i \cdot h_i$, where h_i is the encoder representation of x_i. Attention has been shown to both improve in-distribution performance (Dong and Lapata, 2016) and also lead to better compositional

[1] We use fixed BERT embeddings without fine-tuning in the SQL datasets due to computational constraints.

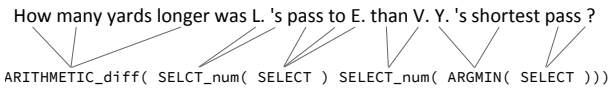

Figure 2: Example of an alignment between question and program tokens in DROP as predicted by FastAlign. *"Lossman"*, *"Evan"*, *"Vince"*, and *"Young"* are abbreviated to *"L."*, *"E."*, *"V."*, and *"Y."* for brevity.

generalization (Finegan-Dollak et al., 2018), by learning a soft alignment between question and program tokens. Since attention is the component in a sequence-to-sequence model that aligns parts of the input to parts of the output, we propose new extensions to the attention mechanism, and examine their effect on compositional generalization.

(a) Attention Supervision Intuitively, learning good alignments between question and program tokens should improve compositional generalization: a model that correctly aligns the token *largest* to the predicate `max` should output this predicate when encountering *largest* in novel contexts.

To encourage learning better alignments, we supervise the attention distribution computed by the decoder to attend to specific question tokens at each time-step (Liu et al., 2016). We use an off-the-shelf word aligner to produce a "gold" alignment between question and program tokens (where program tokens correspond to grammar rules in grammar-based decoding) for all training set examples. Then, at every decoding step where the next prediction symbol's "gold" alignment is to question tokens at indices $\mathcal{I}$, we add the term $-\log \sum_{i \in \mathcal{I}} p_i$ to the objective, pushing the model to put attention probability mass on the aligned tokens. We use the FastAlign word alignment package (Dyer et al., 2013), based on IBM model 2, which is a generative model that allows to extract word alignments from parallel corpus without any annotated data. Figure 2 shows an example question-program pair and the alignments induced by FastAlign.

(b) Attention over Spans Question spans can align to subtrees in the corresponding program. For example, in Fig. 1, *largest state* aligns to `state.area = (select max ... from state)`. Similarly, in a question such as *"What does Lionel Messi do for a living?"*, the multi-word phrase *"do for a living"* aligns to the KB relation `Profession`. Allowing the model to directly attend to multi-token phrases could induce more meaningful alignments that improve compositional generalization.

Here, rather than computing an attention distribution over input tokens $(x_1, \ldots x_n)$, we compute a distribution over the set of spans corresponding to all constituents (including all tokens) as predicted by an off-the-shelf constituency parser (Joshi et al., 2018). Spans are represented using a self-attention mechanism over the hidden representations of the tokens in the span, as in Lee et al. (2017).

(c) Coverage Questions at test time are sometimes similar to training questions, but include new information expressed by a few tokens. A model that memorizes a mapping from question templates to programs can ignore this new information, hampering compositional generalization. To encourage models to attend to the entire question, we add the attention-coverage mechanism from See et al. (2017) to our model. Specifically, at each decoding step the decoder holds a *coverage vector* $c = (c_1, \ldots, c_n)$, where c_i corresponds to the sum of attention probabilities over x_i in all previous time steps. The coverage vector is given as another input to the decoder, and a loss term is added that penalizes attending to tokens with high coverage: $\sum_{i=1}^{n} \min(c_i, p_i)$, encouraging the model to attend to tokens not yet attended to.

3.4 Downsampling Frequent Program Templates

Training a semantic parser can be hampered if the training data contains a highly skewed distribution over program templates, i.e., a large fraction of the training examples correspond to the same template. In such a biased environment, the model might memorize question-to-template mappings instead of modeling the underlying structure of the problem. We propose to downsample examples from frequent templates such that the resulting training data has a more balanced template distribution.

Our initial investigation showed that the distribution over program templates in DROP is highly skewed (20 templates out of 111 constitute 90% of the data), leading to difficulties to achieve *any* generalization to examples from the program split. Thus, in DROP, for any program template in the training set where there are more than 20 examples, we randomly sample 20 examples for training. Downsampling is related to AFLite (Sakaguchi et al., 2020; Bras et al., 2020), an algorithmic approach to bias reduction in datasets. AFLite is applied when bias is hard to define; as we have direct

access to a skewed program distribution, we can take a much simpler approach for reducing bias.

4 Datasets

We create iid and program splits for five datasets according to the procedure of Finegan-Dollak et al. (2018) as described in §2:[2] Four text-to-SQL datasets from Finegan-Dollak et al. (2018) and one dataset for mapping questions to QDMR programs (Wolfson et al., 2020) in DROP (Dua et al., 2019). Similar to prior work (Finegan-Dollak et al., 2018), we train and test models on anonymized programs, that is, entities are replaced with typed variables (§2). Table 1 gives an example question and program for each of these datasets.

- **ATIS**: questions for a flight-booking task (Price, 1990; Dahl et al., 1994).
- **GEOQUERY**: questions about US geography (Zelle and Mooney, 1996).
- **ADVISING**: questions about academic course information. (Finegan-Dollak et al., 2018).
- **SCHOLAR**: questions about academic publications (Iyer et al., 2017).
- **DROP**: questions on history and football games described in text paragraphs. We use annotated QDMR programs from Wolfson et al. (2020).

SQL Grammar: We adapt the SQL grammar developed for ATIS (Lin et al., 2019) to cover the four SQL datasets. To achieve that, additional data normalization steps were taken (see appendix), such as rewriting programs to have a consistent SQL style. The grammar uses the DB schema to produce domain-specific production rules, e.g., in ATIS `table_name` → `FLIGHTSalias0`, `column_name` → `FLIGHTSalias0.MEAL_DESCRIPTION`, and `value` → `class_type0`. At inference time, we enforce context-sensitive constraints that eliminate production rules that are invalid given the previous context. For example, in the `WHERE` clause, the set of `column_name` rules is limited to columns that are part of previously mentioned tables. These constraints reduce the number of syntactically invalid programs, but do not eliminate them completely.

DROP Grammar: We manually develop a grammar over QDMR programs to perform grammar-based decoding for DROP, similar to Gupta et al. (2020). This grammar contains typed operations required for answering questions, such as, `ARITHMETIC_diff(NUM, NUM)` → `NUM`, `SELECT_num(PassageSpan)` → `NUM`, and `SELECT` → `PassageSpan`. Because QDMR programs are executed over text paragraphs (rather than a KB), QDMR operators receive string arguments as inputs (analogous to KB constants), which we remove for anonymization (Table 1). This results in program templates that include only the logical operations required for finding the answer. While such programs cannot be executed as-is on a database, they are sufficient for the purpose of testing compositional generalization in semantic parsing, and can be used as "layouts" in a neural module network approach (Gupta et al., 2020).

5 Experiments

We now present our empirical evaluation of compositional generalization.

5.1 Experimental Setup

We create training/development/test splits using both an *iid split* and a *program split*, such that the number of examples is similar across splits. Table 2 presents exact statistics on the number of unique examples and program templates for all datasets. There are much fewer new templates in the development and test sets for the iid split than for the program split, thus the iid split requires less compositional generalization. In DROP, we report results for the downsampled dataset (§3.4), and analyze downsampling below.

Evaluation Metric We evaluate models using exact match (EM), that is, whether the predicted program is identical to the gold program. In addition, we report *relative gap*, defined as $1 - \frac{\text{EM}_{\text{program}}}{\text{EM}_{\text{iid}}}$, where $\text{EM}_{\text{program}}$ and EM_{iid} are the EM on the program and iid splits, respectively. This metric measures the gap between in-distribution generalization and OOD generalization, and our goal is to minimize it (while additionally maximizing EM_{iid}).

We select hyper-parameters by tuning the learning rate, batch size, dropout, hidden dimension, and use early-stopping w.r.t. development set EM (specific values are in the appendix). The results reported are averaged over 5 different random seeds.

Evaluated Models Our goal is to measure the impact of various modeling choices on compo-

[2] We do not use their original split because we remove duplicate question-program pairs and balance the number of examples between the iid and program splits.

Dataset: GEOQUERY
***x*:** *how many states border the state with the largest population?*
***z*:** `select count( border_info.border ) from border_info as border_info where border_info.state_name in ( select state.state_name from state as state where state.population = ( select max( state.population ) from state as state ) )`

Dataset: ATIS
***x*:** *what is the distance from airport_code0 airport to city_name0 ?*
***z*:** `select distinct airport_service.miles_distant from airport as airport , airport_service as airport_service , city as city where airport.airport_code = "airport_code0" and airport.airport_code = airport_service.airport_code and city.city_code = airport_service.city_code and city.city_name = "city_name0"`

Dataset: SCHOLAR
***x*:** *What papers has authorname0 written?*
***z*:** `select distinct paper.paperid from author as author , paper as paper , writes as writes where author.authorname = "authorname0" and writes.authorid = author.authorid and writes.paperid = paper.paperid`

Dataset: ADVISING
***x*:** *Can undergrads enroll in the course number0 ?*
***z*:** `select distinct course.advisory_requirement , course.enforced_requirement , course.name from course as course where course.department = "department0" and course.number = number0`

Dataset: DROP
***x*:** *How many yards longer was Johnson's longest touchdown compared to his shortest touchdown of the first quarter?*
***z*:** `ARITHMETIC_diff( SELECT_num( ARGMAX( SELECT ) ) SELECT_num( ARGMIN( FILTER( SELECT ) ) ) )`

Table 1: Examples for the different datasets, of a question (x) and its corresponding program (z).

Dataset	Split	# examples (train / dev / test)	# new templates (train / dev / test)
GEOQUERY	iid	409 / 103 / 95	192 / 32 / 24
	Prog.	424 / 91 / 91	148 / 49 / 47
ATIS	iid	3014 / 405 / 402	830 / 48 / 65
	Prog.	3061 / 373 / 375	645 / 140 / 148
SCHOLAR	iid	433 / 111 / 105	158 / 16 / 16
	Prog.	454 / 97 / 98	112 / 37 / 37
ADVISING	iid	3440 / 451 / 446	203 / 0 / 0
	Prog.	3492 / 421 / 414	163 / 20 / 17
DROP	iid	582 / 102 / 500	73 / 0 / 0
	Prog.	582 / 102 / 385	73 / 0 / 38

Table 2: Dataset statistics for the iid split and the program (prog.) split for all datasets. *# new templates* indicates the number of templates unseen during training time for the development and test sets, and the total number of templates for the training set.

Model	iid split	Program split	Rel. gap
SQL			
SEQ2SEQ	74.9	10.8	84.6
+ELMO	76.2	**15.9**	**77.9**
+BERT	**77.5**	10.5	85.7
GRAMMAR	70.1	14.1	78.1
+ELMO	65.5	11.2	81.4
+BERT	67.6	8.4	86.7
DROP			
SEQ2SEQ	45.4	1.6	96.4
+ELMO	53.2	2.1	96.0
+BERT	50.0	0.0	100
GRAMMAR	49.2	2.6	94.7
+ELMO	57.8	**13.2**	**77.1**
+BERT	**64.6**	3.9	93.9

Table 3: Test results for contextualized representations and grammar-based decoding.

sitional generalization. We term our baseline sequence-to-sequence semantic parser SEQ2SEQ, and denote the parser that uses grammar-based decoding by GRAMMAR (§3.2). Use of contextualized representations in these parsers is denoted by +ELMO and +BERT (§3.1). We also experiment with the proposed additions to the decoder attention (§3.3). In a parser, use of (a) auxiliary attention supervision obtained from FastAlign is denoted by +ATTNSUP, (b) use of attention over constituent spans by +ATTNSPAN, and (c) use of attention-coverage mechanism by +COVERAGE.

5.2 Main Results

Below we present the performance of our various models on the test set, and discuss the impact of these modeling choices. For SQL, we present results averaged across the four datasets, and report the exact numbers for each dataset in Table 9.

Model	iid split	Program split	Rel. gap
SQL			
SEQ2SEQ	74.9	10.8	84.6
+ATTNSUP	73.3	18.5	73.2
+ELMO	**76.2**	15.9	77.9
+ELMO+ATTNSUP	73.7	**20.3**	**70.6**
GRAMMAR	70.1	14.1	78.1
+ATTNSUP	73.3	15.8	75.3
+ELMO	65.5	11.2	81.4
+ELMO+ATTNSUP	69.1	11.8	81.6
DROP			
SEQ2SEQ	45.4	1.6	96.4
+ATTNSUP	49.4	1.3	97.3
+ELMO	53.2	2.1	96.0
+ELMO+ATTNSUP	58.2	2.6	95.5
GRAMMAR	49.2	2.6	94.7
+ATTNSUP	55.8	4.7	91.5
+ELMO	57.8	**13.2**	**77.1**
+ELMO+ATTNSUP	**59.8**	12.2	79.5

Table 4: Test results for auxiliary attention supervision.

Model	iid split	Program split	Rel. gap
SQL			
SEQ2SEQ	74.9	10.8	84.6
+COVERAGE	75.3	17	76.2
+ATTNSUP	72.4	23.5	65.8
+ELMO	76.2	15.9	77.9
+ELMO+COVERAGE	**76.2**	24.1	66.5
+ATTNSUP	72	**25.4**	**62.2**
DROP			
SEQ2SEQ	45.4	1.6	96.4
+COVERAGE	47.2	2.1	95.5
+ELMO	53.2	2.1	96.0
+ELMO+COVERAGE	**64.4**	**4.4**	**93.1**

Table 5: Test results for attention-coverage.

Baseline Performance The top-row in Table 3 shows the performance of our baseline SEQ2SEQ model using GloVe representations. In SQL, it achieves 74.9 EM on the iid split and 10.8 EM on the program split, and in DROP, 45.4 EM and a surprisingly low 1.6 EM on the iid and program splits, respectively. A possible reason for the low program split performance on DROP is that programs include only logical operations without any KB constants (§4), making generalization to new compositions harder than in SQL (see also analysis in §5.3). As observed by Finegan-Dollak et al. (2018), there is a large relative gap in performance on the iid vs. program split.

Contextualized Representations Table 3 shows that contextualized representations consistently improve absolute performance and reduce the relative gap in DROP. In SQL, contextualized representations improve absolute performance and reduce the relative gap in the SEQ2SEQ model, but not in the GRAMMAR model. The relative gap is reduced by roughly 7 points in SQL, and 17 points in DROP. As ELMO performs slightly better than BERT, we present results only for ELMO in some of the subsequent experiments, and report results for BERT in Table 9.

Model	iid split	Program split	Rel. gap
SQL			
SEQ2SEQ	74.9	10.8	84.6
+ATTNSPAN	73.8	14.3	79.5
+ELMO	**76.2**	15.9	77.9
+ELMO+ATTNSPAN	75.5	**16.3**	**77.2**
DROP			
SEQ2SEQ	45.4	1.6	96.4
+ATTNSPAN	48.6	**3.1**	**93.6**
+ELMO	53.2	2.1	96.0
+ELMO +ATTNSPAN	**56.2**	1.6	97.1

Table 6: Test results for attention over spans.

Model	iid split		Program split	
	w/o DS	w/ DS	w/o DS	w/ DS
SEQ2SEQ	**49.8**	45.4	0.0	**1.6**
GRAMMAR	**51.6**	49.2	0.0	**2.6**
+ELMO	52.8	**57.8**	0.8	**13.2**

Table 7: Reducing training data bias in DROP by downsampling examples for frequent templates leads to better compositional generalization in all models.

Grammar-based Decoding Table 3 shows that grammar-based decoding both increases accuracy and reduces the relative gap on DROP in all cases. In SQL, grammar-based decoding consistently decreases the absolute performance compared to SEQ2SEQ. We conjecture this is because our SQL grammar contains a large set of rules meant to support the normalized SQL structure of Finegan-Dollak et al. (2018), which makes decoding this structure challenging. We provide further in-depth comparison of performance in §5.3.

Attention Supervision Table 4 shows that attention supervision has a substantial positive effect on compositional generalization, especially in SQL. In SQL, adding auxiliary attention supervision to a SEQ2SEQ model improves the program split EM from 10.8 $\rightarrow$ 18.5, and combining with ELMO leads to an EM of 20.3. Overall, using ELMO and ATTNSUP reduces the relative gap from

84.6 → 70.6 compared to SEQ2SEQ. In DROP, attention supervision improves iid performance and reduces the relative gap for GRAMMAR using GloVe representations, but does not lead to additional improvements when combined with ELMO.

Attention-coverage Table 5 shows that attention-coverage improves absolute performance and compositional generalization in all cases. Interestingly, in SQL, best results are obtained without the attention coverage loss term, but still providing the coverage vector as additional input to the decoder. In SQL, adding attention-coverage improves program split EM from 10.8 → 17. Combining coverage with ELMO and ATTNSUP leads to our best results, where program split EM reaches 25.4, and the relative gap drops from 84.6 → 62.2 (with a slight drop in iid split EM). In DROP, using attention-coverage mechanism with auxiliary coverage loss improves iid performance from 53.2 → 64.6 and reduces the relative gap from 96 → 93.1.

Attention over Spans Table 6 shows that, without ELMO, attention over spans improves iid and program split EM in both SQL and DROP, but when combined with ELMO differences are small and inconsistent.

Downsampling Frequent Templates Table 7 shows that for DROP, where the distribution over program templates is extremely skewed, downsampling training examples for frequent templates leads to better compositional generalization in all models. For example, without downsampling (w/o DS), program split EM drops from 13.2 → 0.8 for the GRAMMAR+ELMO model.

Takeaways We find that contextualized representations, attention supervision, and attention coverage generally improve program split EM and reduce the relative gap, perhaps at a small cost to iid split EM. In DROP, grammar-based decoding is important, as well as downsampling of frequent templates. Overall the gap between in-distribution and OOD performance dropped from 84.6 → 62.2 for SQL, and from 96.4 → 77.1 for DROP. While this improvement is significant, it leaves much to be desired in terms of models and training procedures that truly close this gap.

5.3 Analysis

Error Analysis We analyze the errors of each model on the program split development set for all SQL datasets and label each example with one of three categories (Table 8): *Seen programs* are errors resulting from outputting program templates that appear in the training set, while *new programs* are wrong programs that were not observed in the training set. *Invalid syntax* errors are outputs that are syntactically invalid programs. Table 8 shows that for SEQ2SEQ models, those that improve compositional generalization also increase the frequency of *new programs* and *invalid syntax* errors. Grammar-based models output significantly more *new programs* than SEQ2SEQ models, and less *invalid syntax* errors.[3] Overall, the correlation between successful compositional generalization and the rate of *new programs* is inconsistent.

Model	Seen program	New program	Invalid syntax
SEQ2SEQ	75.7	19.6	**4.7**
+ELMO	64.9	26.2	8.9
+ATTNSUP	62.6	29	8.3
+ELMO	57.4	32.4	10.2
+COVERAGE	59.8	28.9	11.3
+ELMO	**40.5**	**41.3**	18.1
+ATTNSPAN	70.2	22.2	7.5
+ELMO	63.1	29.3	7.6
GRAMMAR	26.2	70.4	**3.4**
+ELMO	**22**	**71.7**	6.3
+ATTNSUP	25.7	68.6	5.7
+ELMO	26.8	69.3	3.9

Table 8: Analysis of program split development set results across all SQL datasets.

We further inspect 30 random predictions of multiple models on both the program split and the iid split (Table 10). *Semantically equivalent* errors are predictions that are equivalent to the target programs. *Semantically similar* is a relaxation of the former category (e.g., an output that represents *"flights that depart at time0"*, where the gold program represents *"flights that depart after time0"*). *Limited divergence* or *significant divergence* corresponds to invalid programs that slightly or significantly diverge from the target output, respectively.

Table 10 shows that adding attention-supervision, attention-coverage, and attention over spans increases the number of predictions that are semantically close to the target programs. We also find that the frequency of correct typed variables in predictions is significantly higher when using

[3]The grammar can still produce invalid outputs (see §4 - SQL Grammar), thus it does not eliminate these errors entirely.

Model	Advising			ATIS			GeoQuery			Scholar		
	iid split	Prog. split	Rel. gap	iid split	Prog. split	Rel. gap	iid split	Prog. split	Rel. gap	iid split	Prog. split	Rel. gap
SEQ2SEQ	90.0	0.1	99.9	70.5	12.3	82.6	70.1	19.1	72.8	69.1	11.6	83.2
+ELMO	91.7	1.9	97.9	71.6	21.1	70.5	73.7	27.9	62.1	68.0	12.9	81.0
+BERT	91.5	0.1	99.9	72.2	17.0	76.5	74.7	18.0	75.9	71.4	6.7	90.6
+ATTNSUP	87.4	1.1	98.7	69.6	28.3	59.3	72.4	25.9	64.2	64.0	18.8	70.6
+ELMO	89.1	0.4	99.6	71.4	28.3	60.4	71.6	**32.5**	54.6	62.7	20.2	67.8
+BERT	90.2	2.3	97.5	70.1	29.9	57.3	74.7	29.2	60.9	64.8	16.3	74.8
+COVERAGE	90.1	1.9	97.9	70.7	23.7	66.5	72.4	27.7	61.7	67.8	14.5	78.6
+ELMO	91.9	5.4	94.1	**74.5**	**34.4**	53.8	73.3	28.4	61.3	65.1	**28.2**	56.7
+BERT	**92.4**	5.0	94.6	73.6	31.7	56.9	**76.6**	28.6	62.7	**73.0**	23.9	67.3
+ATTNSUP	85.9	3.2	96.3	71.1	31.4	55.8	72.6	**34.7**	52.2	60	24.7	58.8
+ELMO	88.6	5	94.4	71	34.3	**51.7**	70.5	34.1	**51.6**	57.7	**28.2**	**51.1**
+BERT	89.1	4.9	94.5	71.8	33.6	53.2	73.9	31.6	57.2	63.2	27.6	56.3
+ATTNSPAN	89.3	3.4	96.2	70.4	17.9	74.6	70.5	22.2	68.5	65.1	13.9	78.6
+ELMO	92.2	4.8	94.8	72.4	23.5	67.5	69.5	24.8	64.3	67.8	12.2	82.0
+BERT	91.9	0.0	100.0	71.5	22.6	68.4	72.0	21.1	70.7	65.3	9.4	85.6
GRAMMAR	88.5	3.0	96.6	65.8	18.1	72.5	63.2	21.8	65.5	61.1	13.7	77.6
+ELMO	90.0	3.1	96.6	61.3	21.3	65.3	58.1	16.3	71.9	52.6	4.3	91.8
+BERT	90.7	2.3	97.5	62.4	7.1	88.6	63.2	20.0	68.4	54.1	4.1	92.4
+ATTNSUP	87.4	**5.9**	**93.2**	63.8	24.2	62.1	64.2	20.4	68.2	63.8	14.3	77.6
+ELMO	89.1	2.0	97.8	65.0	15.9	75.5	62.9	22.4	64.4	59.2	6.7	88.7
+BERT	89.8	3.5	96.1	61.4	3.5	94.3	66.5	12.5	81.2	54.3	3.9	92.8

Table 9: Test EM for all models and SQL datasets. All results are averages over 5 different random seeds.

Model	Semantically equivalent	Semantically similar	Limited divergence	Significant divergence
program split				
SEQ2SEQ+ELMO	4	7	4	15
+ATTNSUP	7	7	5	11
+COVERAGE	4	11	2	13
+ATTNSPAN	5	9	0	16
iid split				
SEQ2SEQ	6	8	4	12
GRAMMAR	6	11	7	6

Table 10: Manual categorization of 30 random predictions on the iid and program splits development sets.

attention-supervision and attention-coverage compared to the baseline model ($p < 0.05$). In addition, the errors of the GRAMMAR model tend to be closer to the target program compared to SEQ2SEQ.

Compositional Generalization in DROP Our results show that compositional generalization in DROP is harder than in the SQL datasets. We hypothesized that this could be due to the existence of KB relations in SQL programs after program anonymization, while QDMR programs do not contain any arguments. To assess that, we further anonymize the predicates in all SQL programs in all four datasets, such that the SQL programs do not contain any KB constants at all (similar to DROP). We split the data based on this anonymization, and term it the *KB-free split*. On the development set, when moving from a program split to a KB-free split, the average accuracy drops from $14.5 \rightarrow 9.8$. This demonstrates that indeed a KB-free split is harder than the program split from Finegan-Dollak et al. (2018), partially explaining the difference between SQL and DROP.

6 Conclusion

We presented a comprehensive evaluation of *compositional generalization* in semantic parsers by analyzing the performance of a wide variety of models across 5 different datasets. We experimented with well-known extensions to sequence-to-sequence models and also proposed novel extensions to the decoder's attention mechanism. Moreover, we proposed reducing dataset bias towards a heavily skewed program template distribution by downsampling examples from frequent templates.

We find that our proposed techniques improve generalization to OOD examples. However, the generalization gap between in-distribution and OOD data remains high. This suggests that future research in semantic parsing should consider more drastic changes to the prevailing encoder-decoder approach to address compositional generalization.

Acknowledgements

This research was supported by The Israel Science Foundation (grant 942/16), The Yandex Initiative for Machine Learning, The European Research Council (ERC) under the European Union Horizons 2020 research and innovation programme (grant ERC DELPHI 802800), and the Army Research Office (grant number W911NF-20-1-0080). The second author was supported by a Google PhD fellowship.

References

Yuval Atzmon, Jonathan Berant, Vahid Kezami, Amir Globerson, and Gal Chechik. 2016. Learning to generalize to new compositions in image understanding. *arXiv preprint arXiv:1608.07639*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron C. Courville. 2019a. Systematic generalization: What is required and can it be learned? In *ICLR*.

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

Ronan Le Bras, Swabha Swayamdipta, Chandra Bhagavatula, Rowan Zellers, Matthew E. Peters, Ashish Sabharwal, and Yejin Choi. 2020. Adversarial filters of dataset biases. *ArXiv*, abs/2002.04108.

Deborah A Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the workshop on Human Language Technology*, pages 43–48. Association for Computational Linguistics.

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, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

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 *NAACL-HLT*.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A simple, fast, and effective reparameterization of IBM model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, Atlanta, Georgia. Association for Computational Linguistics.

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.

Jerry A Fodor, Zenon W Pylyshyn, et al. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71.

Matt Gardner, Yoav Artzi, Victoria Basmova, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, et al. 2020. Evaluating nlp models via contrast sets. *arXiv preprint arXiv:2004.02709*.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. 2020. Neural module networks for reasoning over text. In *ICLR*.

J. Herzig and J. Berant. 2019. Don't paraphrase, detect! rapid and effective data collection for semantic parsing. In *Empirical Methods in Natural Language Processing (EMNLP)*.

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

S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Association for Computational Linguistics (ACL)*.

R. Jia and P. Liang. 2016. Data recombination for neural semantic parsing. In *Association for Computational Linguistics (ACL)*.

Johanna E. Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. 2017. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. *CVPR*.

Vidur Joshi, Matthew Peters, and Mark Hopkins. 2018. Extending a parser to distant domains using a few dozen partially annotated examples. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*,

pages 1190–1199, Melbourne, Australia. Association for Computational Linguistics.

Divyansh Kaushik, Eduard Hovy, and Zachary C Lipton. 2020. Learning the difference that makes a difference with counterfactually-augmented data. In *ICLR*.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *ICLR*.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *EMNLP*.

Brenden M. Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *ICML*.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end neural coreference resolution. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Kevin Lin, Ben Bogin, Mark Neumann, Jonathan Berant, and Matt Gardner. 2019. Grammar-based neural text-to-sql generation. *arXiv preprint arXiv:1905.13326*.

Lemao Liu, Masao Utiyama, Andrew Finch, and Eiichiro Sumita. 2016. Neural machine translation with supervised attention.

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

Richard Montague. 1973. The proper treatment of quantification in ordinary English. In *Approaches to Natural Language*.

J. Pennington, R. Socher, and C. D. Manning. 2014. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. 2018. Deep contextualized word representations. In *North American Association for Computational Linguistics (NAACL)*.

Patti Price. 1990. Evaluation of spoken language systems: The atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149, Vancouver, Canada. Association for Computational Linguistics.

Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M Lake. 2020. A benchmark for systematic generalization in grounded language understanding. *arXiv preprint arXiv:2003.05161*.

Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. 2019. Distributionally robust neural networks. In *International Conference on Learning Representations*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *AAAI*.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*.

Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. Break it down: A question understanding benchmark. *Transactions of the Association for Computational Linguistics*.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

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

A SQL Style

SQL programs vary in style across datasets. We address a specific difference concerning the syntax to neutralize an interaction with the models analyzed in this analysis, and allow comparability across models and datasets. We standardize the form `<table1> <join> <table2> ON <condition>` by replacing `<join>` with a comma and adding `<condition>` to the `WHERE` clause.

B SQL Grammar Development

Our SQL grammar is a context-free grammar. We fit an existing implementation for text-to-SQL (Lin et al., 2019) to the datasets we experimented with. Examples for grammar rules are in Table 12. At each step, a sequence of non-terminal or terminal expressions (right side) is derived from some non-terminal (left side).

The SQL programs in the text-to-SQL datasets have aliases for all tables, sub-queries, and custom fields. Also, each column in the program is preceded by an aliased table or a sub-query. To allow the model to generate all aliases, we add terminal rules based on the dataset schema. We modify the rules to create sub-queries and fields so that the use of aliases is enforced, and we add the alias patterns for custom field and tables. We add the table names in the schema, concatenated with the alias patterns, to *table_name*. We define *col_ref* as the concatenation of an aliased table and a column of this table. Additionally, we add valid combinations of aliased variables and schema entities.

To allow comparability with SEQ2SEQ models, we use only examples that are parsed by the grammar in the development and test sets, eliminating 39 examples from ADVISING, 18 from ATIS and one example from GEOQUERY. The grammar covers at least 95% of each train set.

During inference we enforce contextual rules. For example, forcing the derivation of *from_clause* to have the tables that were selected in *select_results*. We check validity by executing the programs against the dataset database in Mysql server 5.7. Some of the programs in our datasets were not executable due to inconsistent use of aliases, or partial column references. We were not able to automatically fix all the programs. We relaxed our constraints to allow the generation of all target programs, hence allowing some invalid outputs.

Model	ADVISING	ATIS	GEOQUERY	SCHOLAR
SEQ2SEQ	0.7	2.4	0.4	0.2
+ELMO	0.8	7.1	0.3	0.6
+BERT	1.7	3.3	0.3	0.5
+ATTNSUP	3.2	6.0	0.6	0.6
+ELMO	0.6	4.2	0.6	1.2
+COVERAGE	8.0	11.9	0.6	1.0
+ATTNSPAN	4.0	6.1	0.5	0.6
+ELMO	4.1	7.2	0.7	0.7
GRAMMAR	18.8	25.5	1.7	0.6
+ELMO	8.8	22.1	0.8	1.0
+BERT	20.7	36.0	1.5	1.4
+ATTNSUP	25.6	37.3	1.8	1.5
+ELMO	28.6	40.3	6.2	2.4

Table 11: Average training duration in hours for models trained on SQL datasets.

C Training

We implement and train our models using AllenNLP with PyTorch as backend, and conduct experiments on 2 machines each with 4 NVIDIA GeForce GTX 1080 GPUs and 16 Intel(R) Xeon(R) CPU E5 − 1660 v4 CPUs. The OS is Ubuntu 18.04 LTS. Averaged running time per model are detailed in Table 11.

SQL hyper-parameters We use Adam optimizer with learning rate selected from $\{0.001, 0.0001\}$. Batch size is selected from $\{1, 4\}$, and we use patience of 15 epochs. We use EM on the development set as a metric for early stopping and selecting the best hyper-parameters. For all models, we use pre-trained GloVe embeddings of size 100, and the target embedding dimension is 100.Encoder hidden size is selected from $\{200, 300\}$. Dropout is kept fixed at $p = 0.5$. We train each model with five random seeds. We perform a grid-search and use accuracy on the development set for model selection.

ELMO and BERT representations are concatenated to the trainable 100 dimension GloVe embeddings. For BERT we use the top layer of the bert-base-uncased model. ELMO and BERT based models are trained with Noam learning scheduler, with 800 600, or 400 warm-up steps. For the ATTNSUP and COVERAGEmodels, the additional loss term scaling hyper-parameter was tuned using the values $\{0.0, 0.1, 0.5, 1.0, 2.5, 5.0\}$. For our best performing models, SEQ2SEQ+COVERAGE+ELMO, on all datasets, we used an encoder-decoder hidden size of 300, with coverage loss parameter 0. Learning rate was set to 0.0001 for ATIS, and 0.001 for the other datasets.

Global structure	
query	*select_core*, *groupby_clause*, *orderby_clause*, *limit*
select_core	*select_with_distinct* , *select_results*, *from_clause*, `"WHERE"`, *where_clause*
Select clause	
select_results	*select_result*, `","`, *select_result*
select_result	*function*
From clause	
source	*single_source*, `","`, *source*
single_source	*source_subq*
source_subq	`"("`, *query*,`")"`, `"AS"`, *subq_alias*
source_table	`"TABLE_PLACEHOLDER"`, `"AS"`, *table_name*
Where clause	
where_clause	*expr*, `","`, *where_conj*
where_conj	`"AND"`, *where_clause*
Group by clause	
groupby_clause	`"GROUP BY"`, *group_clause*
group_clause	*expr*, *group_clause*
Expressions	
expr	*value*, `"BETWEEN"`, *value*, `"AND"`, *value*
value	*col_ref*
Terminal rules	
table_name	`"FLIGHTalias0"`
column_name	`"FLIGHT_ID"`
col_ref	`"FLIGHTalias0.FLIGHT_ID"`
col_alias	`"DERIVED_FIELDalias0"`
subq_alias	`"DERIVED_TABLEalias0"`

Table 12: Examples for different types of SQL grammar rules. Non-terminal and terminal expressions (in quotation marks) are derived from a non-terminal (left hand side).

DROP hyper-parameters Similar to SQL, we perform a grid-search to choose hyper-parameters based on the development set accuracy. We tune the following parameters in the specified range and select a single value for all experiments (denoted by **bold**): learning rate for Adam optimizer in range $\{\mathbf{0.001}, 0.0005\}$, batch-size in $\{4, \mathbf{16}, 32, 64\}$, and hidden-size for the encoder-decoder LSTMs in $\{\mathbf{100}, 200\}$. Dropout is kept fixed at $p = 0.2$, gradient clipping is performed with norm-threshold$=5.0$, beam-size is set to 5, and training is stopped early if the development set accuracy does not improve for 15 consecutive epochs.

D Development Results

Table 13 contains the development set EM for all models on the DROP dataset. Table 14 contains the development set EM for all models on all SQL datasets.

Model	iid split
SEQ2SEQ	56.9
+ELMO	59.8
+BERT	54.9
+ATTNSUP	55.9
+ELMO	62.7
+COVERAGE	54.9
+ELMO	65.7
+ATTNSPAN	57.8
+ELMO	59.8
GRAMMAR	60.8
+ELMO	67.6
+BERT	65.7
+ATTNSUP	62.7
+ELMO	69.6

Table 13: iid development set exact match for all models on the DROP dataset. We no not create a program-split development set for DROP, one containing templates not seen in training or test. Instead, we use the same iid development set to choose the best model for both iid and program split settings. Note that this is a more challenging setting, since the model selection for the program split is also done on the basis of an in-distribution development set.

Model	Advising			ATIS			GeoQuery			Scholar		
	iid split	Prog. split	Rel. gap	iid split	Prog. split	Rel. gap	iid split	Prog. split	Rel. gap	iid split	Prog. split	Rel. gap
SEQ2SEQ	92.9	9.8	89.5	76	11.5	84.9	67.6	27.5	59.3	73	9.3	87.3
+ELMO	94.6	15.2	83.9	76.9	17.4	77.4	69.1	38.2	44.7	75.9	12	84.2
+BERT	94.1	14.1	85	77.8	14.8	81	71.1	31.4	55.8	77.3	7.4	90.4
+ATTNSUP	92.1	18	80.5	74.4	22.9	69.2	67.2	43.5	35.3	69.5	13.2	81
+ELMO	92.4	21.1	77.2	75.6	23	69.6	67	47.7	28.8	74.2	13.8	81.4
+BERT	93	18.5	80.1	75.5	20.3	73.1	68	47.5	30.1	73.7	13.6	81.5
+COVERAGE	93.2	16.9	81.9	75.6	19.7	73.9	70.7	43.5	38.5	74.2	9.5	87.2
+ELMO	94.9	23.2	75.6	78.4	28.9	63.1	72.2	51.2	29.1	77.8	17.5	77.5
+BERT	95.4	16.8	82.4	79	26.1	67	74.2	51	31.3	79.1	18.4	76.7
+ATTNSPAN	92.4	13.1	85.8	75.7	12.8	83.1	64.5	32.3	49.9	73.2	8.9	87.8
+ELMO	94.2	10.3	89.1	77.6	19.3	75.1	65.4	40.7	37.8	73.9	11.8	84
+BERT	94.6	14.9	84.2	76.6	15.1	80.3	67.6	35.6	47.3	75	13.6	81.9
GRAMMAR	91.1	22.5	75.3	70.1	13	81.5	63.5	24.8	60.9	65.4	14.6	77.7
+ELMO	91.4	15.6	82.9	60.8	13.5	77.8	58.1	21.1	63.7	66.3	14.4	78.3
+BERT	93.9	17.9	80.9	61.7	5.3	91.4	64.3	19.8	69.2	66.8	13.6	79.6
+ATTNSUP	91	23.5	74.2	65.9	23.6	64.2	66.6	26.8	59.8	65.2	14.8	77.3
+ELMO	91	19.4	78.7	67	14.6	78.2	65.6	22.2	66.2	63.8	15.1	76.3
+BERT	91.2	14.9	83.7	59.7	3.5	94.1	65	19.1	70.6	64	12.4	80.6

Table 14: Dev EM for all models and all SQL datasets.