Laziness Is a Virtue When It Comes to Compositionality in Neural Semantic Parsing

Maxwell Crouse, Pavan Kapanipathi, Subhajit Chaudhury,
Tahira Naseem, Ramon Astudillo, Achille Fokoue, Tim Klinger
{ maxwell.crouse, subhajit, ramon.astudillo }@ibm.com
{ kapanipa, tnaseem, achille, tklinger }@us.ibm.com
IBM Research

Abstract

Nearly all general-purpose neural semantic parsers generate logical forms in a strictly top-down autoregressive fashion. Though such systems have achieved impressive results across a variety of datasets and domains, recent works have called into question whether they are ultimately limited in their ability to compositionally generalize. In this work, we approach semantic parsing from, quite literally, the opposite direction; that is, we introduce a neural semantic parsing generation method that constructs logical forms from the bottom up, beginning from the logical form's leaves. The system we introduce is lazy in that it incrementally builds up a set of potential semantic parses, but only expands and processes the most promising candidate parses at each generation step. Such a parsimonious expansion scheme allows the system to maintain an arbitrarily large set of parse hypotheses that are never realized and thus incur minimal computational overhead. We evaluate our approach on compositional generalization; specifically, on the challenging CFQ dataset and three Text-to-SQL datasets where we show that our novel, bottom-up semantic parsing technique outperforms general-purpose semantic parsers while also being competitive with comparable neural parsers that have been designed for each task.

1 Introduction

Compositionality is inherent to natural language, with the meaning of complex text or speech understood through the composition of constituent words and phrases (Montague, 1973). For instance, having observed the usage of phrases like "edited by" and "directed by" in isolation, a human would be able to easily understand the question "Was Toy Story edited by and directed by John Lasseter?". The ability to take individual components and combine them together in novel ways is known as *compositional generalization*, and is a key feature of human intelligence that has been shown to play a significant role in why humans are so efficient at learning (Lake et al., 2017).

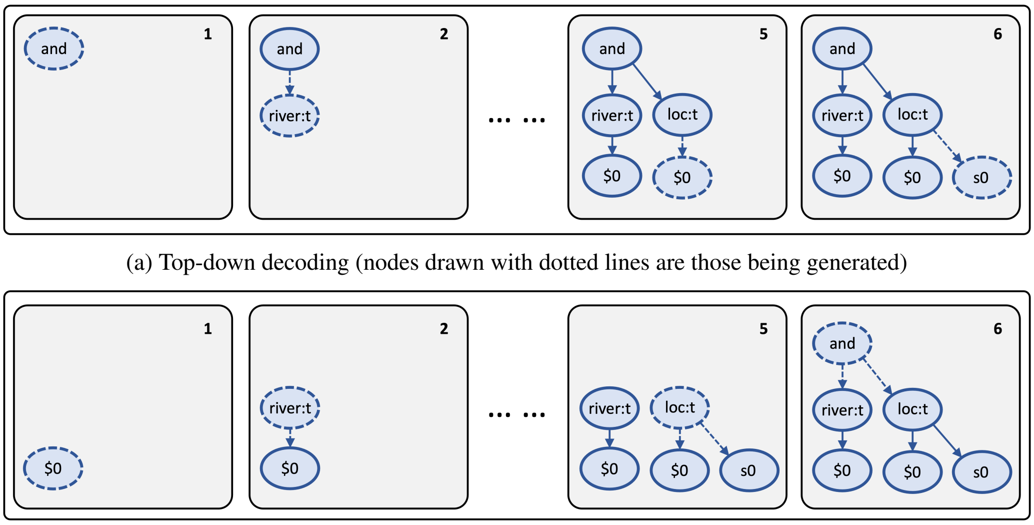
Compositional generalization has been identified as a major point of weakness in neural methods for semantic parsing (Lake and Baroni, 2018; Higgins et al., 2018). Accordingly, this deficiency has been taken up as a challenge by the machine learning community, leading to a slew of methods (Liu et al., 2021; Herzig et al., 2021; Gai et al., 2021) and datasets (Keysers et al., 2019; Kim and Linzen, 2020) targeted towards compositional generalization. However, while there has been progress in determining factors that allow systems to compositionally generalize (Furrer et al., 2020; Oren et al., 2020), there yet remains gaps in our understanding as to why these neural models fail.

In this paper, we posit that the failure of traditional neural semantic parsers to compositionally generalize is in part due to how they build logical forms. Most commonly, neural semantic parsers treat parsing as an encoder-decoder problem, where the decoder generates logical forms from the top down in an autoregressive fashion (Dong and Lapata, 2016; Xiao et al., 2016; Alvarez-Melis and Jaakkola, 2017; Krishnamurthy et al., 2017; Dong and Lapata, 2018). That is, these systems output a linearization of the target logical form's abstract syntax tree, beginning from the root of the tree, where the generation of each token is conditioned on both the input text as well as the entire sequence of previously generated tokens.

Such an entirely autoregressive decoding scheme, wherein the production of each new token is conditioned on all previously generated tokens, could result in models that assume invalid dependencies to exist between tokens and thus overfit to their training data (as observed in (Qiu et al., 2022; Bogin et al., 2022)). We hypothesize that this overfitting problem would lessen the ability of these models to generalize to unseen compositions.

Here we introduce an alternative decoding ap-

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 8434–8448
July 9-14, 2023 ©2023 Association for Computational Linguistics



(a) Top-down decoding (nodes drawn with dotted lines are those being generated)

(b) Bottom-up decoding (nodes drawn with dotted lines are those being generated)

Figure 1: Top-down versus bottom-up decoding strategies for the Geoquery question "What rivers are in s0?"

proach that eschews the top-down generation paradigm, and instead uses a bottom-up mechanism that builds upwards by combining entities and subexpressions together to form larger subexpressions (Figure 1 demonstrates this distinction). Unlike top-down approaches, our decoder generates logical forms by conditioning on only the relevant subexpressions from the overall graph, hence improving compositional generalizability.

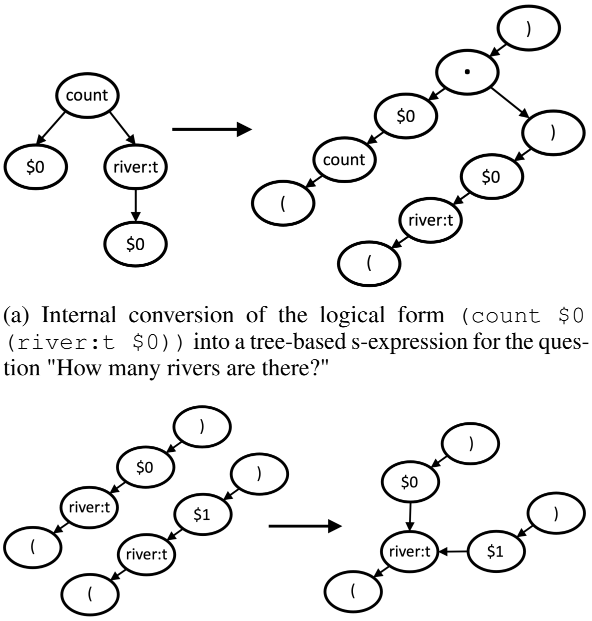
Contributions: (a) We introduce a novel, bottom-up semantic parsing decoder that achieves strong results, specifically with respect to compositional generalization. (b) We evaluate our approach on CFQ (Keysers et al., 2019), three Text-to-SQL (Finegan-Dollak et al., 2018) datasets, and Geoquery (Zelle and Mooney, 1996), and find that it outperforms general-purpose semantic parsers while also being competitive with comparable neural parsers that have been designed for each task. (c) We demonstrate the flexibility of our architecture by testing our approach with multiple encoders, showing that our architecture almost always leads to a significant performance improvement. (d) We show how the bottom-up paradigm can result in a combinatorial explosion in the number of subexpressions created at each decoding step and propose a lazy evaluation scheme that mitigates this by selectively expanding the logical form in a way that minimizes computational overhead.

2 Representing Logical Forms

Our parser is intended to be task-agnostic, with as few as possible assumptions made regarding the particular formalism (e.g., SQL, SPARQL, etc.) a logical form may instantiate. We assume only that we have access to a vocabulary V that defines the set of all symbols a logical form could be constructed from (with no restrictions imposed as to the label or arity of any particular symbol) and that the logical form is given as an s-expression.

S-expressions are a representation formalism that expresses trees as nested lists (e.g., `(count $0 (river:t $0))` as a query for the question "How many rivers are there?"). Their use dates back to the earliest days of artificial intelligence research (McCarthy, 1983) as the main unit of processing of the LISP programming language (McCarthy, 1978). In this work, their main purpose is to simplify the structure of logical forms into binary trees. Importantly, the transformation of logical forms into s-expressions requires no knowledge beyond the syntax of the target formalism.

Traditionally, logical forms are generated as trees; however, we instead represent them as directed acyclic graphs (DAGs). This is a logically equivalent representation that is created by collapsing all identical subtrees into a single subgraph that maintains connections to each of the original parent nodes. Figure 2a provides an example of how an s-expression would initially be converted into a tree representation, while Figure 2b shows how two overlapping trees would be merged into a single DAG form. Within a graph all nodes will either have 1) one argument and a label drawn from the vocabulary of symbols V or 2) two arguments and connect an incomplete list (its left argument) with



(a) Internal conversion of the logical form `(count $0 (river:t $0))` into a tree-based s-expression for the question "How many rivers are there?"

(b) DAG representation of an s-expression where identical subtrees have been mapped to the same nodes

Figure 2: Preprocessing graph transformations

a complete list (its right argument). The nodes with two arguments will always use a special pointer symbol "•" as their label.

As the final preprocessing step, each logical form is wrapped in a special `root` s-expression. For instance, the logical form `(count $0 (river:t $0))` would become `(root (count $0 (river:t $0)))`. This step provides a signal to the model to stop generating new extensions to the logical form. During decoding, only those s-expressions that begin with a `root` token may be returned to the user.

3 Model

Given a question Q, our method is tasked with producing a graph representation G of a logical form using symbols drawn from a vocabulary V. At the highest level, our approach follows the traditional encoder-decoder paradigm. It first parses the input with a neural encoder (e.g., LSTM (Hochreiter and Schmidhuber, 1997)) to produce real-valued vector representations for each word in Q. Those representations are then passed to a neural decoder, which iteratively takes decoding actions to generate G. Our approach is agnostic to the choice of encoder, and thus in the following sections we will describe only the decoding process.

An important desideratum of this work was architectural simplicity. Thus, when using a pretrained large language model as the base to our approach, (e.g., T5 (Raffel et al., 2019)), the model was left largely as is. None of the modifications described in the following sections involve changes to the internal parameters or architecture of the neural model itself (i.e., all additions are kept to the output layer). Consequently, when our approach is instantiated with a pretrained language model, that model is applied entirety off-the-shelf.

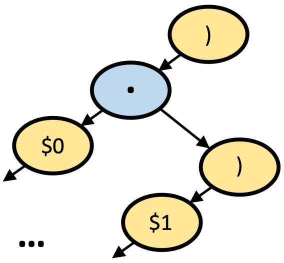


Figure 3: Nodes in G originating from generation actions (yellow) and from pointer actions (blue).

3.1 Decoding Actions

At each step of decoding, our system executes either a *generation action* or a *pointer action*. Generation actions take a single node from G and add it as the argument to a new node with a label drawn from V. Pointer actions instead take two nodes from G and add them both as arguments to a new node with a special pointer symbol "•" as its label. The division of decoding into generation and pointer actions is adapted from (Zhou et al., 2021a,b), which proposes this scheme to drive a transition-based AMR parser. Key to note is that both action types will result in a new node being generated that has as arguments either one or two already-existing nodes from G, i.e., a bottom-up generation process. Figure 3 illustrates the outcomes of both action types.

At the start of decoding our approach initializes two objects, G as the empty graph and a set of candidate actions $A = \{\langle \texttt{(}, \emptyset, 1\rangle\}$. The first action of A will always be to generate the left parenthesis, which is the start to every s-expression in G. Each decoding step begins by extending G with actions drawn from A and ends with a set of new candidate actions being added to A.

The key to our approach lies in its *lazy* evaluation scheme, wherein actions that create nodes are generated at each decoding step, rather than the nodes themselves. This allows our model to strongly restrict how G expands at each decoding

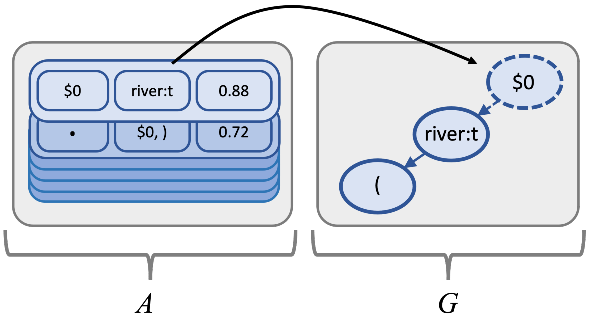


Figure 4: A generation action from A that, when selected and executed, adds a node with label `$0` to G

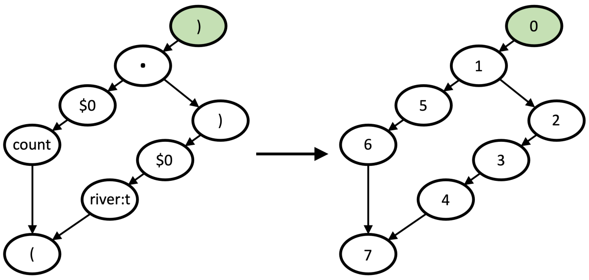


Figure 5: Reversed post-order traversal-based position assignment relative to a newly generated node (colored in green). The number assigned to each node is the index of the position bias (e.g., 0 would be assigned to bias b_0, 1 would be assigned to bias b_1, etc.)

step, with our approach being able to build up a large set of unexecuted actions representing candidate logical forms that are never processed with resource-intensive neural components and thus incur very little computational overhead.

3.2 Candidate Selection

Each element $a \in A$ is a tuple $a = \langle v, \mathcal{A}, p_a \rangle$ consisting of: 1) a symbol v drawn from our vocabulary V, 2) an ordered list of arguments $\mathcal{A}$, where each member of $\mathcal{A}$ is a node in G, and 3) a probability p_a reflective of the model's confidence that this candidate should be part of the final returned output. Adding a candidate $a = \langle v, \mathcal{A}, p_a \rangle$ to G simply involves creating a new node labeled v within G that has directed edges connecting the node to each argument in $\mathcal{A}$, as Figure 4 demonstrates.

Our model is equipped with a selection function that will select and remove members of A to add to G. In the experiments of this paper, our selection function chooses all elements of A with a probability above a certain threshold κ (e.g., $\kappa = 0.5$) or, if no such options exist, chooses the single highest probability option from A.

3.3 Graph Encoding

For our neural model to make predictions, it must first convert the newly generated nodes into real-valued vector representations. The decoder of our model largely follows the design of T5 (Raffel et al., 2019), i.e., a transformer (Vaswani et al., 2017) using relative positional encodings (Shaw et al., 2018), but with modifications that allow it to process graph structure. As most equations are the same as those detailed in the original T5 work, we will only describe the differences between our model and theirs.

Throughout the remainder of this section, we will define a frontier F to refer to the set of nodes generated by actions in A that were selected during the current round of decoding. Each element of F is first assigned a vector embedding according to its label. The embedding is then passed through the decoder to produce real-valued vector representations for each node. In order to capture the structure of G with the transformer, our model makes use of two types of information within the attention modules of each layer of the decoder.

First, within the self-attention module, a node $n \in G$ may only attend to its set of descendants, i.e., only the nodes contained within the subgraph of G rooted at n. This has the effect that the resultant encoding of a node is a function of only its descendants and the original text. Second, for a pair of nodes n_i and n_j for which an attention score is being computed (e.g., n_i is the parent of n_j), the positional encoding bias b_{ij} used to represent the offset between n_i and n_j is assigned according to a reverse post-order traversal of the descendants of node n_i. That is, b_{ij} is selected according to where n_j is positioned in a post-order traversal of the subgraph of G rooted at n_i. Figure 5 provides an example of this ordering scheme for a particular node.

This assignment strategy effectively linearizes the graph rooted at n_i and has the added property that each descendant of n_i will be assigned a unique position bias with respect to n_i. This is also a somewhat analogous encoding as to what is done for standard seq-to-seq models. To see the similarity, consider a "linear" graph (e.g., a sequence); the encoding for each node is exactly that given by a distance-based positional encoding (Shaw et al., 2018) (e.g., used by T5).

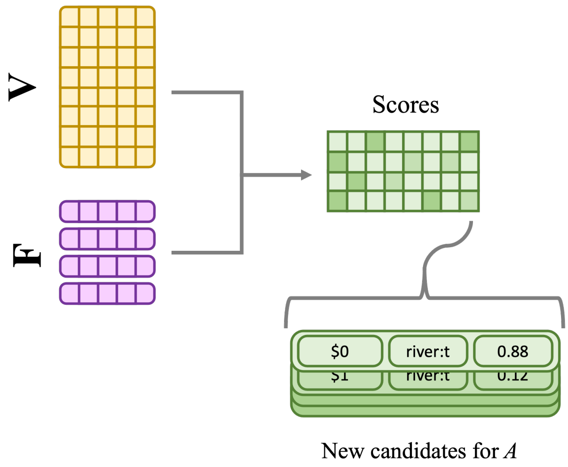


Figure 6: The embeddings of nodes in the frontier F and the embeddings of symbols in the vocabulary V being used to construct new generation actions

3.4 Action Generation

Once the decoder has processed each node, our approach will have a set of node embeddings $\{h_1, h_2, \dots h_{|F|}\}$. To produce the set of decoding actions, our model executes two operations (one for each action type). The first operation proposes a set of generation actions (a brief depiction of this is given in Figure 6). For a node n_i and symbol v with embedded representations h_i and h_v, respectively, this involves computing

$$p_i^{(v)} = \sigma\big(h_i^\top h_v + b\big)$$

where b is a bias term and σ is the sigmoid function. The value $p_i^{(v)}$ can be interpreted as the *independent* probability (independent of all other action probabilities) that G should contain a node with label $v \in V$ that has n_i as an argument. For each $v \in V$, A is extended as $A = A \cup \{\big\langle v, \langle n_i \rangle, p_i^{(v)} \cdot p_{n_i} \big\rangle\}$, where p_{n_i} refers to the probability of the *original* action that generated n_i.

The second operation proposes a set of pointer actions using an attention mechanism. For a pair of nodes $\langle n_i, n_j \rangle$ with embedded representations h_i and h_j, respectively, this is computed as

$$\begin{aligned} q_i &= W^{(q)} h_i, \quad k_j = W^{(k)} h_j \\ p_i^{(j)} &= \sigma\big(\frac{q_i^\top k_j}{\sqrt{d}} + b\big) \end{aligned}$$

where $W^{(q)}$, $W^{(k)}$ are learned matrices, b is a bias term, d is the dimensionality of the embeddings, and σ is the sigmoid function. Like before, $p_i^{(j)}$ can be thought of as the *independent* probability that G will contain a node that has n_i as its left argument and n_j as its right argument. For each pair $\langle n_i, n_j \rangle \in (F \times G) \cup (G \times F)$ (where $F \cup G$ is a slight abuse of notation to write the union between F and the set of nodes within G), we update the set of actions $A = A \cup \{\big\langle \cdot, \langle n_i, n_j \rangle, p_i^{(j)} \cdot p_{n_i} \cdot p_{n_j} \big\rangle\}$, where p_{n_i} and p_{n_j} refer to the probabilities of the original actions that generated n_i and n_j.

It is important to note the combinatorial nature of generating new actions. At each round of decoding, the number of actions could grow by $O(|F| \cdot |V| + |F| \cdot |G|)$. While it is certainly possible to restrict the number of actions added to A (e.g., by adding only the top-k actions), the set still grows extremely quickly.

A key feature to our decoding scheme is that it does not actually build and embed the graphs for a particular action until that action is executed. Because each action maintains the probability of its source actions, our model is effectively exploring via Dijkstra's algorithm, where the frontier F includes all actions yet to be executed. This contrasts with previous works, e.g., (Rubin and Berant, 2021), which are greedier in that they keep a fixed beam with new nodes that are added and embedded on each round and all nodes outside this beam being discarded (our approach does not need to discard these nodes since they have no significant cost until they are executed).

3.5 Terminating Decoding

Decoding terminates when an s-expression beginning with the `root` token is generated (refer to the end of Section 2 for an example). In order to ensure the highest probability s-expression is returned, decoding only terminates if the final action's probability is also the highest amongst all yet-unselected actions in A. Upon generating the `root` s-expression, only the subgraph of G rooted at the newly generated node is returned. Thus, though the the size of G may grow to be quite large, not every element of G is returned.

3.6 Training

Training our model is as efficient as other standard transformer-based models, i.e., a single parallel operation that processes the entire graph and computes all action probabilities simultaneously. Additionally, the memory footprint of our model is practically equivalent to the underlying transformer used to initialize its parameters (e.g., T5).

```
(SELECT (count *) (WHERE
  (directed_by M0 M1)
  (directed_by M0 M2)
  (edited_by M0 M1)
  (edited_by M0 M2)
  (prod_companies M0 M1)
  (prod_companies M0 M2)))
```

Figure 7: S-expression for the CFQ question "Was M0 produced, directed, and edited by M1 and M2"

The loss function used to train our model is a cross-entropy-based loss. Recall that both generation and pointer action probabilities are the output of a sigmoid function. Letting Q be the input question, $\mathcal{P}$ be the set of positive actions (i.e., actions needed to generate the gold-annotation logical form), and $\mathcal{N}$ be the set of negative actions (i.e., actions not needed to generate the gold-annotation logical form), the objective is written as

$$\mathcal{L} = \sum_{n \in \mathcal{P}} \log p_{\theta}(n|Q) + \sum_{n \in \mathcal{N}} \log 1 - p_{\theta}(n|Q)$$

where the conditional probabilities $p_{\theta}(n|Q)$ are the sigmoid-constrained outputs of generation (see Section 3.4) and θ is the set of model parameters.

While $\mathcal{P}$ is fixed and finite, the set $\mathcal{N}$ is clearly unbounded. In this work, we construct $\mathcal{N}$ from a small set of both generation and pointer actions. For each node $n \in G$, we add negative generation actions for each symbol $v \in V$ that is *not* the label of a parent of n. For negative pointer actions, we add actions for each pair of nodes in G that do not have a common parent.

4 Experiments

In our experiments, we aimed to answer the following questions: 1) is bottom-up a more effective decoding paradigm than top-down with respect to compositional generalization and 2) how well does our approach generalize to different domains and formalisms. To answer these questions, we evaluated our approach on several datasets: 1) CFQ (Keysers et al., 2019), 2) the SQL versions of Geoquery (Zelle and Mooney, 1996), ATIS (Dahl et al., 1994), and Scholar (Iyer et al., 2017), each of which were curated by (Finegan-Dollak et al., 2018), and 3) the version of Geoquery (Zelle and Mooney, 1996) used by (Dong and Lapata, 2016; Kwiatkowski et al., 2011) which maps questions to a variant of typed lambda calculus (Carpenter, 1997). To save space, we provide all hyperparameters used for our experiments in the appendix.

Our main experiments investigating compositional generalization were performed on the CFQ and Text-to-SQL datasets. In CFQ, the distribution of train and test data in each split is designed to exhibit maximum compound divergence (MCD) (Keysers et al., 2019). In the MCD setting, data is comprised of a fixed set of atoms, i.e., the primitive elements to a question. But while each atom is observed in training, the compositions of atoms between train and test sets will vary significantly (an example of a CFQ question is provided in Figure 7). In the Text-to-SQL datasets, the train and test sets are constructed by using separate template splits (Finegan-Dollak et al., 2018; Herzig et al., 2021), where the train and test sets have questions involving different SQL query templates.

4.1 Dataset-Specific Processing

Task-specific representations and encoding schemes are very common with state-of-the-art approaches for compositional generalization (Shaw et al., 2021). Most often, such approaches will transform the target logical form to better reflect some characteristic of the input text. For instance, with CFQ, systems will transform their target logical forms into a query that closely aligns with the list-heavy syntactic structure of CFQ questions (Liu et al., 2021; Herzig et al., 2021; Jambor and Bahdanau, 2021; Furrer et al., 2020).

As our objective was to determine the effectiveness of our bottom-up technique for general-purpose semantic parsing, we avoided any transformation that involved task-specific knowledge about the input text. For the Text-to-SQL and Geoquery datasets, we performed no processing of the inputs or outputs beyond the transformation of input logical forms into s-expressions. For CFQ, we found it necessary to apply two preprocessing steps. First, to mitigate the number of candidate logical forms proposed, only one `WHERE` clause was allowed to be generated during decoding. Second, we observed that, because we restricted the number of `WHERE` clauses generated, our method often prematurely ended generation. Thus, we found it useful to add any completed s-expression (i.e., s-expressions ending with the `)` token) to the `WHERE` clause of the final logical form.

Method	MCD1	MCD2	MCD3	MCD Avg.
Dynamic Least-to-Most Prompting (Drozdov et al., 2022)	94.3	95.3	95.5	95.0
LEAR (Liu et al., 2021)	91.7	89.2	91.7	90.9
LIR+RIR (T5-3B) (Herzig et al., 2021)	88.4	85.3	77.9	83.8
Grounded Graph Decoding (Gai et al., 2021)	98.6	67.9	77.4	81.3
HPD (Guo et al., 2020)	79.6	59.6	67.8	69.0
LIR+RIR (T5-base) (Herzig et al., 2021)	85.8	64.0	53.6	67.8
T5-11B-mod (Furrer et al., 2020)	61.6	31.3	33.3	42.1
LAGR (Jambor and Bahdanau, 2021)	57.9	26.0	20.9	34.9
T5-3B (Herzig et al., 2021)	65.0	41.0	42.6	49.5
T5-base (Herzig et al., 2021)	58.5	27.0	18.4	34.6
Edge Transformer (Bergen et al., 2021)	47.7	13.1	13.2	24.7
Evolved Transformer (Keysers et al., 2019)	42.4	9.3	10.8	20.8
Universal Transformer (Keysers et al., 2019)	37.4	8.1	11.3	18.9
Transformer (Keysers et al., 2019)	34.9	8.2	10.6	17.8
LSTM (Keysers et al., 2019)	28.9	5.0	10.8	14.9
LSP (Ours)				
+ T5-base	88.7	57.7	43.8	63.4
+ LSTM Encoder	73.0	29.5	23.1	41.9

Table 1: Performance across different splits of the CFQ dataset. Those systems leveraging CFQ-specific algorithms or representations are grouped in the top half of the table, while those systems in the bottom half are domain general

Method	ATIS	Geoquery	Scholar
Seq2Seq ♣	32.0	20.0	5.0
GECA ♢	24.0	52.1	–
Seq2Seq ♠	28.0	48.5	–
Transformer ♠	23.0	53.9	–
Seq2Seq+ST ♠	29.1	63.6	–
Transformer+ST ♠	28.6	61.9	–
SpanBasedSP □	–	82.2	–
Baseline (T5-base) ♡	32.9	79.7	18.1
Baseline (T5-large) ♡	31.4	81.9	17.5
Baseline (T5-3B) ♡	29.7	79.7	16.2
LIR+RIR (T5-base) ♡	47.8	83.0	20.0
LIR+RIR (T5-large) ♡	43.2	79.7	22.0
LIR+RIR (T5-3B) ♡	28.5	75.8	12.4
LSP (Ours)			
+ T5-base	38.3	81.3	25.1

Table 2: Performance on Text-to-SQL against: ♣(Finegan-Dollak et al., 2018), ♢(Andreas, 2020), ♠(Zheng and Lapata, 2020), ♡(Herzig et al., 2021), □(Herzig and Berant, 2021)

Method	Acc.
DCS+L w/ lexicon (Liang et al., 2013)	91.1
TISP (Zhao and Huang, 2015)	88.9
KZGS11 (Kwiatkowski et al., 2011)	88.6
DCS+L w/o lexicon(Liang et al., 2013)	87.9
AQA (Crouse, 2021)	87.5
λ-WASP (Wong and Mooney, 2007)	86.6
ZC07 (Zettlemoyer and Collins, 2007)	86.1
AWP + AE + C2 (Jia and Liang, 2016)	89.3
Graph2Tree (Li et al., 2020)	88.9
Coarse2Fine (Dong and Lapata, 2018)	88.2
Seq2Tree (Dong and Lapata, 2016)	87.1
SpanbasedSP (Herzig and Berant, 2021)	86.4
Graph2Seq (Xu et al., 2018)	85.7
LSP (Ours)	
+ LSTM Encoder	86.4

Table 3: Performance on Geoquery, with neural-based approaches grouped in the bottom half of the table

4.2 Evaluation

We evaluated our approach using exact match accuracy, i.e., whether the generated logical form exactly matches the gold logical form annotation. To accommodate for unordered n-ary predicates, we reordered the arguments to all such unordered predicates (e.g., `and`) lexicographically in both the gold and generated logical forms before comparing them. The unordered predicates are the `WHERE` operator in CFQ as well as the `and` predicate in the lambda calculus version of Geoquery.

5 Results and Discussion

Table 1 shows the results of our approach, referred to as LSP (**L**azy **S**emantic **P**arser), on the CFQ dataset. As can be seen from the table, our approach fares quite well against domain-general semantic parsing approaches and, importantly, significantly outperforms both T5-base and the LSTM.

The large performance increase as compared to both the basic LSTM and T5-base supports our hypothesis that a bottom-up, semi-autoregressive decoding strategy is a better inductive bias for compositional generalization than autoregressive, top-down decoding. However, a better decoding strat-

Method	MCD1	MCD2	MCD3	ATIS	Geoquery	Scholar
T5-base	58.5	27.0	18.4	32.9	79.7	18.1
RIR	86.3	49.1	46.8	81.3	36.3	19.4
LIR$_d$	48.1	40.3	35.3	44.4	83.5	20.6
LIR$_d$+RIR	72.5	61.1	51.2	47.8	83.0	20.0
LIR$_{ind}$	57.6	41.4	34.7	38.3	80.8	16.5
LIR$_{ind}$+RIR	85.8	64.0	53.6	41.5	81.9	16.5
LSP (Ours)						
+ T5-base	88.7	57.7	43.8	38.3	81.3	25.1

Table 4: Performance as compared to task-specific T5-base models from (Herzig et al., 2021).

egy clearly does not provide the complete solution, as evidenced by the gap in performance between the best systems designed specifically for compositional generalization and our approach.

Table 2 shows the performance of our approach on the three Text-to-SQL datasets of (Finegan-Dollak et al., 2018). While our approach clearly outperforms the baselines, it has mixed results as compared to the work of (Herzig et al., 2021). Additionaly, our approach was outperformed by (Herzig and Berant, 2021) on the Geoquery SQL dataset; however we note that their approach required a manually constructed lexical mapping of text to Geoquery terms (they reached 65.9% accuracy without the mapping). Still, that our approach outperformed T5-base in all three datasets again supports our hypothesis that bottom-up is more effective for compositional generalization.

Table 3 shows the performance of our approach on Geoquery. To keep the comparison relatively fair, we used only an LSTM encoder and randomly initialized token embeddings for this experiment. Despite using an entirely different generation scheme, our method is competitive with the other neural-based semantic parsers on this dataset. Notably, there is not a significant drop from Seq2Tree (Dong and Lapata, 2016), which can be considered the antithesis to our method as it decodes trees from the top down. We find these results to be very encouraging, as our approach maintains competitive performance against a varied set of approaches that leverage hand-engineered data augmentation strategies (Jia and Liang, 2016), pretrained word embeddings (Li et al., 2020), and hand-built lexicons (Herzig and Berant, 2021).

Lastly, in Table 4 we highlight the T5-base models of (Herzig et al., 2021), which used task-specific intermediate representations to simplify the learning problem of semantic parsing for an off-the-shelf large language model. This is a very useful work to compare against, as the number of parameters between our model and theirs is near identical. From the table, we see that our approach roughly matches that of their best performing T5-base model. This is an interesting result, as it could suggest that our architecture is naturally capturing the useful properties of their intermediate representations without as much need for hand-engineering. Importantly, unlike their approach which assumes knowledge of the syntactic structure of questions, ours needs only a rule for identifying expressions (e.g., parentheses delineate expression boundaries) or a grammar for the target (readily available for formalisms like SPARQL or SQL).

6 Related Work

While the overall parsing approach of (Zhou et al., 2021a,b) is quite different from ours (as they propose a transition-based semantic parser), several aspects of our work were inspired from their ideas. For instance, as mentioned in Section 3.3, they proposed the use of two types of decoding actions to expand the target graph.

Bottom-up neural semantic parsers are a relatively recent development. Most related to our work is that of (Rubin and Berant, 2021), which introduced a bottom-up system that achieved near state-of-the-art results on the Spider dataset (Yu et al., 2018) while being significantly more computationally efficient than its top-down, autoregressive competitors. There are three significant differences between their approach and ours. First, they generate trees rather than DAGs and thus assume a many-to-one correspondence between input spans in the question and tokens in the logical form (which does not hold for datasets like CFQ). Second, they use an eager evaluation scheme (i.e., nodes are generated at each decoding step and not actions), and thus enforce a beam-size hyperparameter K that must be defined a priori and places a hard limit on the number of subtrees that can be considered at a given time. Third, they use a highly customized decoder which limits how tightly integrated their approach can be with off-the-shelf language models. In contrast, when using a pretrained large language model as the base to our approach, we reuse the entirety of the language model as is to instantiate our model.

Another similar work is (Herzig and Berant, 2021), which introduced a method that learned to predict span trees over the input question that could be composed together to build complete logical forms. Their method achieves strong performance on the Geoquery SQL dataset (Finegan-Dollak et al., 2018); however, similar to (Rubin and Berant, 2021), their approach differs from ours in that they assume a one-to-one correspondence between disjoint spans of the question and disjoint subtrees of the logical form. Further, their method can only handle inputs whose parse falls into a restricted class of non-projective trees, where the authors note that extending their method to all classes of non-projective trees would prohibitively increase the time complexity of their parser. Such assumptions make it unclear how it would directly apply to a more complex dataset like CFQ.

There are several works targeted specifically towards compositional generalization (Gai et al., 2021; Liu et al., 2021; Jambor and Bahdanau, 2021; Guo et al., 2020; Herzig et al., 2021). Though these systems have proven to be quite effective on compositional generalization datasets, they make significant task-specific assumptions that limit how they might be applied more broadly.

7 Limitations

The main limitation to our work lies in the handling of unordered n-ary relations. We hypothesize that the bottom-up paradigm performs best when there is one unambiguous logical form to generate for a particular question. While this is quite often true for semantic parsing, in our experience, unordered n-ary relations (e.g., `WHERE`) can quickly cause this not to be the case. With such relations, there tends to be a large number of correct logical forms for a particular question. In these situations, having so many candidate logical forms can cause significant issues in terms of runtime. A second limitation of our work is that it assumes the logical form will be given as a graph. Thus, there is an annotation burden on the users of this system that would not be present in systems that treat semantic parsing as a text-to-text problem (e.g., fine-tuned large language models).

8 Conclusions

In this work, we introduced a novel, bottom-up decoding scheme that semi-autoregressively constructs logical forms in a way that facilitates compositional generalization. Key to our method was a lazy generation process designed to address the computational efficiency issues inherent to bottom-up parsing. We demonstrated our approach on five different datasets covering three logical formalisms and found that our approach was strongly competitive with neural models tailored to each task.

References

David Alvarez-Melis and T. Jaakkola. 2017. Tree-structured decoding with doubly-recurrent neural networks. In *ICLR*.

Jacob Andreas. 2020. Good-enough compositional data augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7556–7566.

Leon Bergen, Timothy O'Donnell, and Dzmitry Bahdanau. 2021. Systematic generalization with edge transformers. *Advances in Neural Information Processing Systems*, 34.

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.

Bob Carpenter. 1997. Type-logical semantics.

Maxwell Crouse. 2021. *Question-Answering with Structural Analogy*. Ph.D. thesis, Northwestern University.

Deborah A Dahl, Madeleine Bates, Michael K Brown, William M Fisher, Kate Hunicke-Smith, David S Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the atis task: The atis-3 corpus. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

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.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742, Melbourne, Australia. Association for Computational Linguistics.

Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. 2022. Compositional semantic parsing with large language models. *arXiv preprint arXiv: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.

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

Yu Gai, Paras Jain, Wendi Zhang, Joseph Gonzalez, Dawn Song, and Ion Stoica. 2021. Grounded graph decoding improves compositional generalization in question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 1829–1838.

Yinuo Guo, Zeqi Lin, Jian-Guang Lou, and Dongmei Zhang. 2020. Hierarchical poset decoding for compositional generalization in language. *Advances in Neural Information Processing Systems*, 33.

Jonathan Herzig and Jonathan Berant. 2021. Span-based semantic parsing for compositional generalization. In *ACL/IJCNLP*.

Jonathan Herzig, Peter Shaw, Ming-Wei Chang, Kelvin Guu, Panupong Pasupat, and Yuan Zhang. 2021. Unlocking compositional generalization in pre-trained models using intermediate representations. *arXiv preprint arXiv:2104.07478*.

Irina Higgins, Nicolas Sonnerat, Loic Matthey, Arka Pal, Christopher P Burgess, Matko Bošnjak, Murray Shanahan, Matthew Botvinick, Demis Hassabis, and Alexander Lerchner. 2018. Scan: Learning hierarchical compositional visual concepts. In *International Conference on Learning Representations*.

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

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *55th Annual Meeting of the Association for Computational Linguistics*.

Dora Jambor and Dzmitry Bahdanau. 2021. Lagr: Labeling aligned graphs for improving systematic generalization in semantic parsing. *arXiv preprint arXiv:2110.07572*.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. 2019. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*.

Najoung Kim and Tal Linzen. 2020. Cogs: A compositional generalization challenge based on semantic interpretation. *arXiv preprint arXiv:2010.05465*.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526, Copenhagen, Denmark. Association for Computational Linguistics.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in ccg grammar induction for semantic parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1512–1523.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pages 2873–2882. PMLR.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. 2017. Building machines that learn and think like people. *Behavioral and brain sciences*, 40.

Shucheng Li, Lingfei Wu, Shiwei Feng, Fangli Xu, Fengyuan Xu, and Sheng Zhong. 2020. Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem. *EMNLP*.

Percy Liang, Michael I Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

Chenyao Liu, Shengnan An, Zeqi Lin, Qian Liu, Bei Chen, Jian-Guang Lou, Lijie Wen, Nanning Zheng, and Dongmei Zhang. 2021. Learning algebraic recombination for compositional generalization. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1129–1144.

John McCarthy. 1978. History of lisp. In *History of programming languages*, pages 173–185.

John McCarthy. 1983. Recursive functions of symbolic expressions. In *Programming Languages*, pages 175–186. Springer.

Richard Montague. 1973. The proper treatment of quantification in ordinary english. In *Approaches to natural language*, pages 221–242. Springer.

Inbar Oren, Jonathan Herzig, Nitish Gupta, Matt Gardner, and Jonathan Berant. 2020. Improving compositional generalization in semantic parsing. In *EMNLP (Findings)*.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Tianze Shi, Jonathan Herzig, Emily Pitler, Fei Sha, and Kristina Toutanova. 2022. 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. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Ohad Rubin and Jonathan Berant. 2021. Smbop: Semi-autoregressive bottom-up semantic parsing. In *Proceedings of the 5th Workshop on Structured Prediction for NLP (SPNLP 2021)*, pages 12–21.

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.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967.

Chunyang Xiao, Marc Dymetman, and Claire Gardent. 2016. Sequence-based structured prediction for semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1341–1350, Berlin, Germany. Association for Computational Linguistics.

Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, Michael Witbrock, and Vadim Sheinin. 2018. Graph2seq: Graph to sequence learning with attention-based neural networks. *arXiv preprint arXiv:1804.00823*.

Tao Yu, Rui Zhang, Kai-Chou Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Z Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*.

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.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687.

Kai Zhao and Liang Huang. 2015. Type-driven incremental semantic parsing with polymorphism. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1416–1421.

Hao Zheng and Mirella Lapata. 2020. Compositional generalization via semantic tagging. *arXiv preprint arXiv:2010.11818*.

Jiawei Zhou, Tahira Naseem, Ramón Fernandez Astudillo, and Radu Florian. 2021a. Amr parsing with action-pointer transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5585–5598.

Jiawei Zhou, Tahira Naseem, Ramón Fernandez Astudillo, Young-Suk Lee, Radu Florian, and Salim Roukos. 2021b. Structure-aware fine-tuning of sequence-to-sequence transformers for transition-based amr parsing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6279–6290.

A Appendix

A.1 Dataset Sizes

Geoquery (Zelle and Mooney, 1996) is a standard benchmark dataset for semantic parsing that consists of 880 geography-related questions. We use the standard (Zettlemoyer and Collins, 2007) split of 600 train questions and 280 test questions and the logical form representation of (Dong and Lapata, 2016; Kwiatkowski et al., 2011), which is a variant of typed lambda calculus (Carpenter, 1997).

A.2 Training Details

All models were trained on an HPC cluster, where each training run was provided with 100 GB RAM, 2 CPUs, and 1 V100 GPU. The longest training run (on CFQ with T5-base) would complete within 24 hours.

A.3 Model Sizes

The T5-base models have around 220 million parameters. As our model shares all parameters with T5 except for the final attention-based pointer module, a rough estimate is that our model would have approximately 221 million parameters (with 768×768 additional parameters from the attention module).

A.4 Hyperparameters

We did not do much in the way of hyperparameter tuning, instead opting to base most values off of the original CFQ (Keysers et al., 2019) work and sharing most hyperparameter settings between our experiments. In Tables 6 and 7, we list the hyperparameters used in both experiments.

Dataset	Split	Train	Test
CFQ	MCD1	95k	12k
	MCD2	95k	12k
	MCD3	95k	12k
Text-To-SQL	ATIS	4812	347
	Geoquery	539	182
	Scholar	408	315
Geoquery	-	600	280

Table 5: Dataset sizes

Hyperparameter	CFQ	Text-to-SQL	Geoquery
Embedding dimensionality	256	256	256
Number of MHA heads	8	8	8
Dimensionality of FFN hidden layers	1024	1024	1024
Encoder Dropout	0.4	0.4	0.4
Decoder Dropout	0.1	0.1	0.1
Batch size	32	32	32
Learning rate	0.0001	0.001	0.001
Number of training epochs	100	500	500

Table 6: Hyperparameters for CFQ, Text-to-SQL, and Geoquery experiments when an LSTM encoder was used

Hyperparameter	CFQ	Text-to-SQL
Embedding dimensionality	768	768
Number of MHA heads	12	12
Dimensionality of FFN hidden layers	3072	3072
Encoder Dropout	0.1	0.1
Decoder Dropout	0.1	0.1
Batch size	16	16
Learning rate	0.0001	0.0001
Number of training epochs	20	500

Table 7: Hyperparameters for CFQ and Text-to-SQL experiments when T5-base was used

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Section 8

□ A2. Did you discuss any potential risks of your work?
Not applicable. Left blank.

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

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

B ☒ Did you use or create scientific artifacts?

Left blank.

□ B1. Did you cite the creators of artifacts you used?
No response.

□ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
No response.

□ 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)?
No response.

□ 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?
No response.

□ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
No response.

□ 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.
No response.

C ☑ Did you run computational experiments?

Sections 4 and 5

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

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Appendix

☒ 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?
Reported only a single run due to time constraints

□ 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.)?
Not applicable. Left blank.

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.