

A Appendix

A.1 Coverage and multi-turn sampling

When we build an empirical distribution over templates on the training set of Spider, we observe a 85% coverage of dev set templates. That is, 85% of dev set examples have a query whose template occurs in the training set. In other words, while this simple template-filling sampling scheme doesn't provide full coverage over the dev set as a complex grammar would, it covers a large portion of examples.

For Sparc and CoSQL, the sampling procedure is similar to Algorithm 1. However, because there are two queries (one previous, one current), we first sample a previous query z'_1 from $P_{\text{temp}}(z)$, then sample the current query z'_2 from $P_{\text{temp}}(z|z'_1)$. As before, the empirical template distributions are obtained by counting templates in the training set.

A.2 Hyperparameters

Dropout location	Forward parser		
	Spider	Sparc	CoSQL
post-BERT	0.1	0.1	0.1
post-enc LSTMs	0.1	0.3	0.1
pre-dec scorer	0.1	0.1	0.3

Table 4: Dropout rates for the forward parser.

Dropout location	Backward generator		
	Spider	Sparc	CoSQL
post-BERT	0.1	0.3	0.1
post-enc LSTMs	0.1	0.1	0.1
pre-dec scorer	0.1	0.1	0.3

Table 5: Dropout rates for the backward generator.

We use 300-dimensional LSTMs throughout the model. The BERT model we use is DistilBERT (Sanh et al., 2020), which we optimize with Adam (Kingma and Ba, 2015) with an initial learning rate of $5e - 5$. We train for 50 epochs with a batch size of 10 and gradient clipping with a norm of 20. We use dropout after BERT, after encoder LSTMs, and before the pointer scorer. The values for these dropouts used by our leaderboard submissions are shown in Table 4 and Table 5. For each task, these rates are tuned using 3-fold cross-

validation with a coarse grid-search over values $\{0.1, 0.3\}$ for each dropout with a fixed seed.

A single training run of the forward parser took approximately 16 hours to run on a single NVIDIA Titan X GPU. Each task required 3 folds in addition to the final official train/dev run. For each fold, we grid-searched over dropout rates, which amounts to 8 runs. In total, we conducted 27 runs on a Slurm cluster. Including pretrained BERT parameters, the final forward parser contains 142 million parameters. The final backward utterance generator contains 73 million parameters.

list all the last name of owners in alphabetical order .	<code>select last_name from Owners order by last_name</code>
how many friend are there ?	<code>select count (*) from Friend</code>
what is the id of the votes that has been most distinct contestants ?	<code>"select T2.vote_id from CONTESTANTS as T1 join VOTES as T2 on T1.contestant_number = T2.contestant_number group by (T2.vote_id) order by count (T1.contestant_number) desc limit 1</code>
what are the name of higher ?	<code>select name from Highschooler</code>
how many car makers has the horsepower of 81 ?	<code>select count (*) from cars_data as T1 join car_names as T2 on T1.Id = T2.MakeId join model_list as T3 on T2.Model = T3.Model join car_makers as T4 on T3.Maker = T4.Id where T1.Horsepower = '81'</code>
what are the starts of hiring who are located in the city of Bristol ?	<code>select T2.Start_from from employee as T1 join hiring as T2 on T1.Employee_ID = T2.Employee_ID where T1.City = 'Bristol'</code>
find the name and district of the employee that has the highest evaluation bonus .	<code>select T2.Name , T4.District from evaluation as T1 join employee as T2 on T1.Employee_ID = T2.Employee_ID join hiring as T3 on T2.Employee_ID = T3.Employee_ID join shop as T4 on T3.Shop_ID = T4.Shop_ID order by T1.Bonus desc limit 1</code>
what is the cell number of the owners with the largest charges amount ?	<code>select T1.cell_number from Owners as T1 join Charges as T2 order by T2.charge_amount desc limit 1</code>
what is the minimum , average , and maximum grade of all high schooler ?	<code>select min (grade) , avg (grade) , max (grade) from Highschooler</code>
what is the age of the teacher who has the most course ?	<code>select T1.Age from teacher as T1 join course_arrange as T2 on T1.Teacher_ID = T2.Teacher_ID group by T2.Teacher_ID order by sum (T2.Grade) desc limit 1</code>

Table 6: Examples of synthesized queries

A.3 Synthesized examples

In order to quantify the distribution of synthesized examples, we classify synthesized queries according to the difficulty criteria from Spider (Yu et al., 2018b). Compared to the Spider development set, GAZP-synthesized data has an average of 0.60 vs. 0.47 joins, 1.21 vs. 1.37 conditions, 0.20 vs. 0.26 group by’s, 0.23 vs. 0.25 order by’s, 0.07 vs. 0.04 intersections, and 1.25 vs. 1.32 selection columns per query. This suggests that GAZP queries are similar to real data.

Moreover, we example a random sample of 60 synthesized examples. Out of the 60, 51 are correct. Mistakes come from aggregation over wrong columns (e.g. “has the most course” becomes `order by sum T2.grade`) and underspecification (e.g. “lowest of the stadium who has the lowest age”). There are grammatical errors (e.g. “that has the most” becomes “that has been most”), but most questions are fluent and sensible (e.g. “find the name and district of the employee that has the highest evaluation bonus”). A subset of these queries are shown in Table 6.

A.4 Performance breakdown

		easy	medium	hard	extra	all
count		470	857	463	357	2147
baseline	EM	75.3	54.9	45.0	24.8	52.1
	EX	60.3	52.7	47.5	32.6	49.8
	FX	73.6	52.9	44.8	26.4	51.1
GAZP	EM	73.1	58.7	47.2	23.3	53.3
	EX	59.6	59.2	52.3	33.3	53.5
	FX	71.9	55.3	46.1	24.5	51.7

Table 7: Difficulty breakdown for Spider test set.

		easy	medium	hard	extra	all
count		993	845	399	261	2498
baseline	EM	68.9	36.9	31.2	11.1	45.9
	EX	61.9	35.6	30.6	18.8	43.5
	FX	65.9	32.5	28.1	10.7	42.8
GAZP	EM	66.5	39.6	38.4	14.2	45.9
	EX	60.1	39.5	31.1	20.3	44.6
	FX	65.3	36.8	26.3	12.6	43.9

Table 8: Difficulty breakdown for Sparc test set.

		easy	medium	hard	extra	all
count		730	607	358	209	1904
baseline	EM	58.2	28.0	20.6	18.8	37.2
	EX	47.1	27.2	26.8	28.2	34.9
	FX	51.9	24.1	21.2	20.6	33.8
GAZP	EM	60.0	33.8	23.1	13.9	39.7
	EX	48.1	28.3	41.0	23.9	35.9
	FX	55.1	26.9	25.7	16.7	36.3

Table 9: Difficulty breakdown for CoSQL test set.

		turn 1	turn 2	turn 3	turn 4+
count		842	841	613	202
baseline	EM	69.9	41.8	28.9	16.4
	EX	67.8	36.9	28.1	16.9
	FX	70.2	35.7	24.8	13.4
GAZP	EM	67.8	41.9	29.7	19.6
	EX	66.3	40.1	29.0	19.8
	FX	68.8	38.3	25.9	18.3

Table 10: Turn breakdown for Sparc test set

In addition to the main experiment results in Table 2 of Section 3.1, we also examine the performance breakdown across query classes and turns.

GAZP improves performance on harder queries. First, we divide queries into difficulty classes following the classification in Yu et al. (2018b). These difficulty classes are based on the number of SQL components, selections, and conditions. For example, queries that contain more SQL keywords such as `GROUP BY`, `ORDER BY`, `INTERSECT`, nested subqueries, column selections, and aggregators, etc are considered to be harder. Yu et al. (2018b) shows examples of SQL queries in the four hardness categories. Note that **extra** is a catch-all category for queries that exceed qualifications of **hard**, as a result it includes artifacts (e.g. set exclusion operations) that may introduce other confounding factors. Tables 7, 8, and 9 respectively break down the performance of models on Spider, Sparc, and CoSQL. We observe that the gains in GAZP are generally more pronounced in more difficult queries. This finding is consistent across tasks (with some variance) and across three evaluation metrics.

One potential explanation for this gain is that the generalization problem is exacerbated in more

		turn 1	turn 2	turn 3	turn 4+
count		548	533	372	351
baseline	EM	47.3	36.5	32.3	28.5
	EX	43.8	34.3	30.3	27.9
	FX	46.2	31.9	29.4	23.4
GAZP	EM	50.0	36.7	35.7	30.3
	EX	46.4	32.3	32.2	30.2
	FX	50.0	32.8	31.4	27.1

Table 11: Turn breakdown for CoSQL test set.

difficult queries. Consider the example of language-to-SQL parsing, in which we have trained a parser on an university database and are now evaluating it on a sales database. While it is difficult to produce simple queries in the sales database due to a lack of training data, it is likely even more difficult to produce nested queries, queries with groupings, queries with multiple conditions, etc. Because GAZP synthesizes queries — including difficult ones — in the sales database, the adapted parser learns to handle these cases. In contrast, simpler queries are likely easier to learn, hence adaptation does not help as much.

GAZP improves performance in longer interactions. For Sparc and CoSQL, which include multi-turn interactions between the user and the system, we divide queries into how many turns into the interaction they occur. This classification is described in Yu et al. (2019b) and Yu et al. (2019a). Tables 10 and 11 respectively break down the performance of models on Sparc and CoSQL. We observe that the gains in GAZP are more pronounced in turns later in the interaction. Against, this finding is consistent not only across tasks, but across the three evaluation metrics.

A possible reason for this gain is that the conditional sampling procedure shown in Algorithm 1 improves multi-turn parsing by synthesizing multi-turn examples. How much additional variation should we expect in a multi-turn setting? Suppose we discover T coarse-grain templates by counting the training data, where each coarse-grain template has S slots on average. For simplicity, let us ignore value slots and only consider column slots. Given a new database with N columns, the number of possible filled queries is on the order of $O\left(T \times \binom{S}{N}\right)$. For K turns, the number of possi-

ble queries sequences is then $O\left(\left(T \times \binom{S}{N}\right)^K\right)$.

This exponential increase in query variety may improve parser performance on later-turn queries (e.g. those with a previous interaction), which in turn reduce cascading errors throughout the interaction.