Controllable Semantic Parsing via Retrieval Augmentation

Panupong Pasupat and **Yuan Zhang** and **Kelvin Guu**
Google Research
{ppasupat,zhangyua,kguu}@google.com

Abstract

In practical applications of semantic parsing, we often want to rapidly change the behavior of the parser, such as enabling it to handle queries in a new domain, or changing its predictions on certain targeted queries. While we can introduce new training examples exhibiting the target behavior, a mechanism for enacting such behavior changes without expensive model re-training would be preferable. To this end, we propose ControllAble Semantic Parser via Exemplar Retrieval (CASPER). Given an input query, the parser retrieves related exemplars from a retrieval index, augments them to the query, and then applies a generative seq2seq model to produce an output parse. The exemplars act as a control mechanism over the generic generative model: by manipulating the retrieval index or how the augmented query is constructed, we can manipulate the behavior of the parser. On the MTOP dataset, in addition to achieving state-of-the-art on the standard setup, we show that CASPER can parse queries in a new domain, adapt the prediction toward the specified patterns, or adapt to new semantic schemas without having to further re-train the model.

1 Introduction

Semantic parsing is the task of mapping input queries to their meaning representations. In practical applications of semantic parsing, such as conversational agents, we often want to rapidly *control* the behavior of the parser. We particularly focus on three scenarios: (1) **Domain bootstrapping:** making the parser handle queries in a new domain (Su and Yan, 2017; Hou et al., 2020a; Li et al., 2021b). This requires predicting new semantic labels (e.g., intents and slots) unseen during training, and assigning correct values to such labels. (2) **Parse guiding:** influencing the prediction toward a specific parse pattern. This can be used as an override for sensitive queries or queries that the model struggles on. (3) **Schema refactoring:** adapting the parser to changes in the semantic schema such as semantic label renaming (Gaddy et al., 2020).

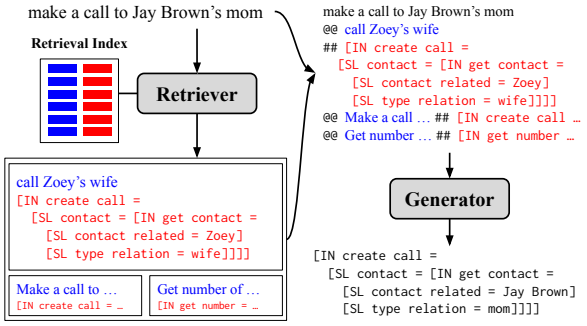

Figure 1: The CASPER model. (1) Given a query x, we retrieve exemplars (x'_i, y'_i) from the retrieval index. (2) We construct an augmented query x^+ based on x and the retrieved exemplars. (3) We apply a generative model on x^+ to produce an output parse y. The retrieval index and augmentation procedure can be modified to change the parser's behavior without re-training.

A common way to control the parser's behavior is to construct training examples exhibiting the new behavior (e.g., examples from the new domain) and tune the model on them. However, model training requires computational resources, which can become unwieldy if we need to make multiple rapid changes. Ideally, we want to control the behavior of the semantic parser *without additional training*. Such an ability would enable many novel use cases. For example, developers could update the semantic parser's behavior and observe the results immediately, thus speeding up the development cycle. This can be used to quickly update the parser in time-critical scenarios while waiting for a fully retrained model. Another use case is deploying a single model to service multiple clients. Each client can manipulate the parser to fit its use without interfering with the central model or other clients, thus saving resources and preserving privacy.

To this end, we propose ControllAble Semantic Parser via Exemplar Retrieval (CASPER). As illustrated in Figure 1, the parser first retrieves

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 7683–7698
November 7–11, 2021.

exemplars relevant to the input query (e.g., training examples resembling the input query) from a modifiable retrieval index. These retrieved exemplars are then augmented to the query. Finally, a seq2seq generator model takes the augmented query as input and generates a meaning representation. The model takes inspiration from recent works that use modifiable retrieval indices (Khandelwal et al., 2020, 2021) and exemplar-augmented inputs (Brown et al., 2020; Liu et al., 2021).

The retrieval and augmentation processes grant us several ways to control the behavior of the parser. For instance, in domain bootstrapping, we can add examples from the new domain to the retrieval index. When these added examples are retrieved, the generator can condition on them while generating the output. This allows the generator to, for instance, follow the semantic template of the exemplars and produce new semantic labels unseen during training.

We evaluate our approach on the English portion of the MTOP dataset (Li et al., 2021a). In our experiments, we show that CASPER preserves the generality and increases the performance of a seq2seq-based semantic parser, while also enabling new capabilities that are simply not possible with standard seq2seq parsers. Our main results are:[1]

- **Standard setup:** On the English portion of MTOP, CASPER gives 2.1% absolute improvement over the existing state-of-the-art, and 1.3% improvement over the underlying T5 seq2seq parsing model.
- **Domain bootstrapping:** By adding examples from a new domain to the retrieval index at test time, CASPER can parse examples in the new domain without model re-training, while also preserving performance on other domains.
- **Parse guiding:** We can train CASPER to follow the semantic template of the manually provided exemplars when asked to do so, while maintaining accuracy on the standard setup.
- **Schema refactoring:** By editing the retrieval index, CASPER can, without re-training, adapt to a new semantic schema where some semantic labels are split into unseen labels.

[1]The code for the experiments is available at https://casperparser.page.link/code

2 Approach

We present ControllAble Semantic Parser via Exemplar Retrieval (CASPER) for parsing queries x into meaning representations (MRs) y. As demonstrated in Figure 1, the prediction procedure consists of the following steps:

1. **Exemplar retrieval:** Retrieve a list E of k *exemplars* (x'_i, y'_i) (x'_i is a query; y'_i is the MR of x'_i) that are related to the input query x.
2. **Augmentation:** From x and E, construct a retrieval-augmented query x^+.
3. **Generation:** Use a generative seq2seq model to map x^+ into an output MR y.

We will elaborate on each step in the following subsections. We can view exemplar augmentation as a way to provide extra information to any seq2seq-based semantic parser, while still preserving its ability to generate complex outputs. The generator can learn to consider or ignore the provided exemplars, so CASPER can perform at least as well as the underlying generator in the standard setup (Section 3). Additionally, we will later show that by manipulating the retrieval index and how the augmented queries are constructed, we can control the behavior of CASPER for domain bootstrapping (Sections 4), parse guiding (Section 5), and schema refactoring (Section 6).

Retrieval The retrieval index consists of input-output pairs (x', y'), and is initially constructed from training examples. We utilize a retriever that uses query embedding cosine similarity as the retrieval score. Concretely, each index entry (x', y') is indexed with the embedding $e(x')$ of the query, computed using a sentence embedder. Given an input x, we rank all index entries (x', y') by the cosine similarity between $e(x)$ and $e(x')$, and let the list E of exemplars be the top-k entries.

For our experiments, we use the large version of the pre-trained Universal Sentence Encoder (USE-large) (Cer et al., 2018) to embed the queries. We did not fine-tune the embedder. As the retrieval index is small enough (≈16k entries), we simply rank all index entries and choose $k = 5$ top entries as the exemplars.

Augmentation From the input query x and the list $E = [(x'_1, y'_1), \ldots, (x'_k, y'_k)]$ of retrieved exemplars, we construct an augmented query x^+. Similar to previous works that use exemplar-augmented

inputs (Guu et al., 2018; Lewis et al., 2020; Brown et al., 2020; Liu et al., 2021), we simply concatenate each exemplar to the query:

$$x^+ = x \texttt{ @@ } x'_1 \texttt{ ## } y'_1 \texttt{ @@ } x'_2 \texttt{ ## } y'_2 \texttt{ @@ } \ldots$$

where @@ and ## are the separator strings. The MRs y'_i are simply treated as strings.

Generation We fine-tune a pre-trained seq2seq model to map the augmented query x^+ to the string representation of y. For our experiments, we fine-tune T5-base (Raffel et al., 2020). While T5 was pre-trained on text data, our experiments show that it can effectively generate MRs after fine-tuning.

Training We keep the retriever fixed and only train the generator model. When constructing (x^+, y) pairs for training the generator, we want to *diversify* the list of exemplars E. This would encourage the generator to learn when to use or ignore the exemplars based on the their quality and relevance to the input x. To this end, instead of using top-k retrieval results as exemplars, at training time we construct a **sampled-k** exemplar list E as follows. From the input x, we first create a ranked pool of all index entries, excluding ones whose query is exactly x. At each step $i \in \{1, \ldots, k\}$, we choose the jth entry from the pool with probability $\propto p(1-p)^{j-1}$ (where p is a hyperparameter, set to 0.5 in the experiments). This geometric distribution makes higher-ranking entries get sampled more frequently. We then remove the sampled exemplar from the pool and add it to E.

2.1 Faithfulness toward exemplars

Although the generation of y is conditioned on the exemplars in E, the generator could implicitly ignore the exemplars entirely. This is desirable as it allows the model to generate reasonable outputs even when the exemplars are of low quality. However, if the parser always ignores the exemplars, we will not be able to control the parser via exemplar manipulation, and the parser might struggle on out-of-distribution examples at test time (e.g., in the domain bootstrapping setup).

We want the parser to be more faithful toward exemplars, but still want the generator to make a judgment call to ignore the exemplars when appropriate. Additionally, we want an on-demand mechanism for adjusting the degree of faithfulness on specific queries. We thus propose the following techniques:

Original exemplars and target output:
y'_1: [IN create call = [SL contact = [IN get contact = [SL contact related = Zoey] [SL relation = wife]]]]
y'_2: [IN get call = [SL contact = Jack] [SL time = today]]
y: [IN create call = [SL contact = [IN get contact = [SL relation = dad]]] [SL name app = Whatsapp]]
Anonymized:
y'_1: [IN 42 = [SL 39 = [IN 53 = [SL 6 = Zoey] [SL 94 = wife]]]]
y'_2: [IN 12 = [SL 39 = Jack] [SL 71 = today]]
y: [IN 42 = [SL 39 = [IN 53 = [SL 94 = dad]]] [SL 88 = Whatsapp]]

Figure 2: Anonymized exemplars and target output.

Anonymization Most seq2seq models, including T5, can learn to copy[2] tokens from the input string. However, with regular supervision, the model may still end up not learning to copy semantic labels from the exemplars if (1) such labels appear so frequently that the model memorizes their usage and generates them without copying, or (2) the retrieval is imperfect and copying from exemplars hurts the model during training.

To explicitly teach the generator to copy labels from the exemplars, we create additional *anonymized training data* where each unique semantic label in y'_i and y are turned into a random numerical label, as illustrated in Figure 2. Since the labels are anonymized differently in each example, the generator can no longer memorize their usage, and must learn to identify and copy the correct anonymized labels. We train the generator on an equal mix of original and anonymized data.

Guiding tag In some scenarios, we want to manually instruct the model to be more faithful toward the exemplars than usual. To do so, we utilize a special token ("PLATINUM" in our experiments) as a *guiding tag*. We simply insert T before each exemplar when constructing x^+:

$$x^+ = x \texttt{ @@ } T\ \tilde{x}_1 \texttt{ ## } \tilde{y}_1 \texttt{ @@ } T\ \tilde{x}_2 \texttt{ ## } \tilde{y}_2 \texttt{ @@ } \ldots$$

To establish the behavior of the guiding tag, we create additional training examples (x^+, y) where x^+ contains the guiding tag, and the prediction y is considered highly faithful to the augmented exemplars in x^+. One instantiation, *oracle training data*, can be constructed by constraining the retrieved exemplars (x'_i, y'_i) so that y'_i and y share the *semantic template* (any notion of semantic similarity; e.g., the MR's labels and hierarchical structure).

[2] T5 does not have an explicit copy mechanism, but it effectively learns to produce the same tokens as the input.

Method	Dev		Test	
	Exact	Template	Exact	Template
mBART+MT	-	-	84.3	-
T5	83.18	87.22	85.06	88.70
CASPER	84.29	87.65	85.54	89.13
CASPER$_{orig}$	**84.67**	**87.98**	**86.36**	**89.65**
CASPER$_{anon}$	79.61	82.60	80.85	83.90

Table 1: **Standard setup:** exact match and template accuracy on the English portion of MTOP. CASPER outperforms T5 and previous state-of-the-art. (underlined = worse than baseline)

The generator is trained on the combination of this oracle training data and the normal data.

3 Standard setup experiments

We start with evaluating CASPER on the standard train-test setup of the English portion of the MTOP dataset (Li et al., 2021a).[3] We show that the retrieved exemplars can aid the seq2seq generator even on in-distribution queries.

3.1 Setup

Data The MTOP dataset uses the decoupled TOP representation (Gupta et al., 2018; Li et al., 2021a), as exemplified in Figures 1, 2, and 3. A TOP representation is a tree, with each node labeled with either an intent (e.g., IN:CREATE_CALL) or a slot (e.g., SL:CONTACT). Each node also corresponds to a single token span of the query. The topmost node is always an intent node.

For our models, we simply treat the TOP representation as a string. We start with the string serialization given in the dataset, and then lowercase and word-split the labels to simplify tokenization (e.g., [IN:GET_CALL . . .] → [IN get call = . . .]).[4]

The English portion of the dataset contains 15667 training, 2235 development, and 4386 test queries. Each query also belongs to one of 11 domains, which will be important in the domain bootstrapping setup (Section 4).

We define the *template* of a TOP tree to be its tree structure and node labels, with the query tokens discarded (e.g., the template of [IN:A [SL:B text]] is [IN:A [SL:B]]). In addition to the main evaluation metric of exact match accuracy, we also report template accuracy.

[3] Available at https://fb.me/mtop_dataset

[4] This does not significantly affect the model's accuracy, but it reduces the number of tokens and thus allows more exemplars to fit into the augmented query.

x: What's the biggest story today? **(a)**
x'_1: what's the top story for today?
y'_1: [IN get stories news = [SL news reference = top] [SL news type = story] [SL date time = for today]]
x'_4: Tell me the biggest news story of the day.
y'_4: [IN get stories news = [SL news type = news story]]

T5: [IN get stories news = [SL news type = story] [SL date time = today]]
C$_o$: [IN get stories news = [SL news reference = biggest] [SL news type = story] [SL date time = today]] ✓

x: Do you have any reminders for me? **(b)**
x'_1: do I have any reminders for today?
y'_1: [IN get reminder = [SL person reminded = I] [SL date time = for today]]
x'_2: Do I have any reminders for today?
y'_2: [IN get reminder = [SL person reminded = I] [SL date time = for today]]

T5: [IN get reminder = [SL person reminded = me]] ✓
C$_o$: [IN get reminder = [SL person reminded = you]]

x: any news updates? **(c)**
x'_2: any updates on the news
y'_2: [IN get stories news = [SL news type = updates] [SL news type = news]]
x'_4: Are there any news updates
y'_4: [IN get stories news = [SL news type = news updates]]

T5: [IN get stories news = [SL news type = news]] ✓
C$_o$: [IN get stories news = [SL news type = news updates]]

Figure 3: Example predictions by T5 and CASPER$_{orig}$ (C$_o$). (2 out of 5 exemplars are shown; ✓= correct)

Methods The main CASPER model is trained on a mixture of original and anonymized training data. We also consider two variants: CASPER$_{orig}$ trained only on original data, and CASPER$_{anon}$ trained only on anonymized data. Since CASPER$_{anon}$ does not know about actual labels, the test data for CASPER$_{anon}$ is also anonymized. None of the models use oracle training data with guiding tag in this section.

Baselines We compare against mBART+MT, the best published result from Li et al. (2021a). We also consider fine-tuning T5 on the original training data without exemplar augmentation.

3.2 Results and analysis

Table 1 shows the experimental results on the test set of MTOP averaged over 3 runs. The base T5 model already outperforms previous state-of-the-art by 0.7%. With retrieval-augmentation, CASPER further improves upon T5, leading to a total of 1.24% gain in exact match accuracy.

The CASPER$_{orig}$ variant, which is trained only on non-anonymized data, achieves an even higher gain of 2.1%. With in-distribution test data, leaning

toward memorization rather than following noisy exemplars is likely the best strategy. However, we will show in later sections that CASPER trained on mixed data is more robust to out-of-distribution queries and changes in the retrieval index.

Our error analysis shows that, while augmented exemplars improve performance over the baseline in general, they also cause some losses. Figure 3a shows a winning example where CASPER$_{\text{orig}}$ is better at predicting the slot that shows up in the augmented exemplars. Figure 3b shows a loss due to the exemplars not being in analogy with the input query, while Figure 3c shows a case where annotation inconsistency between the exemplars and the gold output causes a loss.

Note that while T5 was pre-trained on text data, CASPER effectively learns to generate syntactically valid MRs, with only 0.04% of test outputs being syntactically invalid. Post-hoc filtering or constrained decoding (Yin and Neubig, 2017; Krishnamurthy et al., 2017) could be used if one needs an absolute guarantee for syntactic correctness.

3.3 Ablation

Retriever We train CASPER$_{\text{orig}}$ with different choices of the retriever's embedder: BERT-base (embedding of the [CLS] token) (Devlin et al., 2019), BERT-large, and USE-large. We also consider an oracle retriever that only returns examples with the same template as the correct output.

Table 2 reports intrinsic metrics of the retrievers, which include template recall@5 (whether one of the top 5 retrievals has the same template as the gold MR) and label coverage@5 (whether all labels in the gold MR appear among the top 5 retrievals). Meanwhile, Table 3 reports the end-to-end results on the development set of the trained baseline T5 and CASPER$_{\text{orig}}$ models.

We observe that USE-large, being pre-trained on sentence-level tasks, performs better than BERT on both intrinsic and end-to-end evaluation. On the other hand, the oracle performs much better than USE-large, showing that an improved retriever (e.g., USE-large fine-tuned on the training data) could potentially improve the CASPER model.

Exemplar selection We compare different ways to select exemplars when constructing training data (x^+, y). The choices include using a fixed top-k list (less diverse) instead of sampled-k, and using different number of exemplars k. Note that we always use the top-k list at test time, with the same k as training time.

Retriever	Template Recall@5	Label Coverage@5
BERT-base	71.9	84.5
BERT-large	70.2	82.9
USE-large	80.8	90.3
oracle	97.6	97.6

Table 2: Intrinsic evaluation of the retrievers on the development data of MTOP.

Retriever	Train x'_i, y'_i	Exact	Template
USE-large	sampled-5	84.67	87.98
none (baseline T5)	-	83.18	87.22
BERT-base	sampled-5	83.83	87.35
BERT-large	sampled-5	84.04	87.46
oracle	sampled-5	92.39	97.63
USE-large	top-1	84.06	87.49
USE-large	top-5	84.21	87.38
USE-large	sampled-1	84.00	87.47
USE-large	sampled-3	84.38	87.70
USE-large	sampled-10	84.79	87.96

Table 3: **Ablation:** results of CASPER$_{\text{orig}}$ variants on the development data of MTOP.

Table 3 compares the results. We see that using sampled exemplars during training and a higher k give additive improvements to the model. However, note that larger k's can make the augmented query exceed the model's maximum query length.

4 Domain bootstrapping experiments

In addition to improving the parser on the standard setup, we will show that by manipulating the retrieval index, we can influence the parser's behavior. We first consider domain bootstrapping, where a new domain is being added to a previously trained parser, and we want to quickly update the model using a handful of examples in the new domain.

4.1 Setup

In each experiment, we select one domain in MTOP as the new domain to be bootstrapped. Let $\mathcal{O}_{\text{train}}$ and $\mathcal{N}_{\text{train}}$ and be the sets of training examples in the **original** domains and the **new** domain, respectively. Define $\mathcal{O}_{\text{dev}}$ and $\mathcal{N}_{\text{dev}}$ similarly on the development set.

We consider two settings. In the *seen-bootstrap* setting, at training time, the parser is given access to $\mathcal{O}_{\text{train}}$ and a small subset $\mathcal{N}_{\text{sup}} \subseteq \mathcal{N}_{\text{train}}$ of examples from the new domain. The parser can choose to fine-tune on $\mathcal{N}_{\text{sup}}$ or use it in anyway it likes. The parser is then evaluated on $\mathcal{N}_{\text{dev}}$ to see how well it produces MRs in the new domain, as well as

on $\mathcal{O}_{\text{dev}}$ to ensure that the performance on other domains are not affected.

A more difficult setting is the *unseen-bootstrap* setting: the subset $\mathcal{N}_{\text{sup}}$ is only available to the model at test time and not during training (i.e., $\mathcal{O}_{\text{train}}$ is the only training data available). A method that cannot incorporate side information from $\mathcal{N}_{\text{sup}}$ at test time would perform poorly in this setting.

Methods In both settings, we put all examples available at the time into the retrieval index. For the seen-bootstrap setup, the index contains $\mathcal{O}_{\text{train}} \cup \mathcal{N}_{\text{sup}}$ at all time. Training examples are constructed from $\mathcal{O}_{\text{train}}$ and $\mathcal{N}_{\text{sup}}$ with 50% chance of picking each set. For the unseen-bootstrap setup, the index contains just $\mathcal{O}_{\text{train}}$ during training, and $\mathcal{N}_{\text{sup}}$ is added on top at test time. Training examples are constructed from $\mathcal{O}_{\text{train}}$.

Note that when evaluating on $\mathcal{N}_{\text{dev}}$, exemplars can come from both $\mathcal{O}_{\text{train}}$ and $\mathcal{N}_{\text{sup}}$ in the retrieval index. If the retriever does its job well, we would still mostly get exemplars from $\mathcal{N}_{\text{sup}}$.

Baselines While there are previous works on domain bootstrapping for semantic parsing without fine-tuning (Hou et al., 2020a; Zhu et al., 2020; Krone et al., 2020; Henderson and Vulić, 2021), most of them rely on token-level matching and sequence tagging, which are not directly applicable to the hierarchical MRs from MTOP. We thus compare CASPER with T5, which represents generic seq2seq parsers.

For the unseen-bootstrap setting, we additionally try fine-tuning T5 on either $\mathcal{N}_{\text{sup}}$ or $\mathcal{O}_{\text{train}} + \mathcal{N}_{\text{sup}}$ for a small number of steps. These *fast update* experiments demonstrate the trade-off of spending additional resources for fine-tuning at test time.

4.2 Results

In Table 4, we report results of the models trained in the standard setup (full training data) and the two domain bootstrapping setups (with $\mathcal{N}_{\text{sup}}$ = 100 random examples from $\mathcal{N}_{\text{train}}$). The results are averaged over 5 bootstrapped domains: alarm, calling, event, messaging, and music.

We observe that CASPER shows larger improvements upon T5 in the domain bootstrapping settings than the standard setting, ranging from +2% when $\mathcal{N}_{\text{sup}}$ is seen during training (seen-bootstrap), to +38% when $\mathcal{N}_{\text{sup}}$ is only available at test time (unseen-bootstrap). The results show that by modifying the retrieval index, we can change the behavior of CASPER without needing to fine-tune on examples from the new domain.

Setting:	**standard**		**seen-boot.**		**unseen-boot.**	
Train data:	$\mathcal{O}_{\text{train}} + \mathcal{N}_{\text{train}}$		$\mathcal{O}_{\text{train}} + \mathcal{N}_{\text{sup}}$		$\mathcal{O}_{\text{train}}$	
Eval data:	$\mathcal{N}_{\text{dev}}$	$\mathcal{O}_{\text{dev}}$	$\mathcal{N}_{\text{dev}}$	$\mathcal{O}_{\text{dev}}$	$\mathcal{N}_{\text{dev}}$	$\mathcal{O}_{\text{dev}}$
T5	87.63	82.83	70.70	82.73	5.65	82.73
CASPER	88.61	83.62	72.74	83.73	43.90	83.87
CASPER$_{\text{orig}}$	**89.37**	**84.24**	**73.32**	**83.88**	39.15	**84.07**
CASPER$_{\text{anon}}$	84.47	78.98	63.06	79.09	**53.79**	79.22

Table 4: **Domain bootstrapping:** exact match accuracy averaged over 5 choices of new domains. On the unseen-bootstrap setting, CASPER has the highest gain on the new domain without hurting other domains. (underlined = worse than baseline)

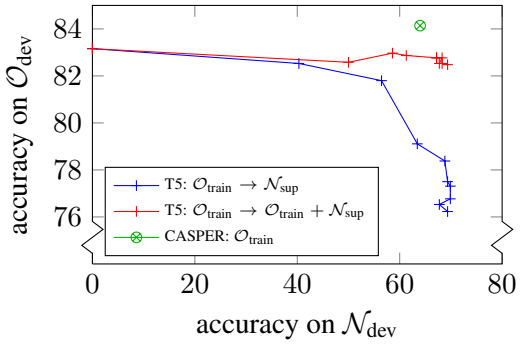

Figure 4: **Fast update for domain bootstrapping:** accuracy on $\mathcal{N}_{\text{dev}}$ and $\mathcal{O}_{\text{dev}}$ (new domain = alarm) when T5 trained on $\mathcal{O}_{\text{train}}$ is fine-tuned on either $\mathcal{N}_{\text{sup}}$ (blue) or $\mathcal{O}_{\text{train}} + \mathcal{N}_{\text{sup}}$ (red) at test time.

On the unseen-bootstrap setup, the model has to rely solely on the exemplars for unseen semantic labels and parse patterns. The anonymized training data proves to be crucial for making the model more faithful toward the exemplars, as evidenced by CASPER trained on mixed data improving upon CASPER$_{\text{orig}}$, and CASPER$_{\text{anon}}$ winning over CASPER$_{\text{orig}}$ by a large margin.

Comparison with fast update The line plots in Figure 4 track the accuracy on $\mathcal{N}_{\text{dev}}$ and $\mathcal{O}_{\text{dev}}$ (bootstrapped domain = alarm) when T5 trained on $\mathcal{O}_{\text{train}}$ is fine-tuned on the support set for a few iterations at test time. If only the support set is used for fast update (blue), the model eventually suffers from catastrophic forgetting and degrades on $\mathcal{O}_{\text{dev}}$. Mixing in $\mathcal{O}_{\text{train}}$ during fast update (red) solves this issue. T5 eventually surpasses the unseen-bootstrap CASPER (green) on $\mathcal{N}_{\text{dev}}$ after processing 512 examples, at which point much more computational resource was already consumed than CASPER.

5 Parse guiding experiments

In this section, we demonstrate CASPER's ability to guide the prediction toward the patterns specified in the exemplars. This parse guiding ability can be useful for correcting the parser's output on a set of problematic queries (e.g., sensitive queries, or queries that the model struggles on). In industrial semantic parsers, one common way to handle problematic queries is to add explicit "hotfix" filters and treat such queries as special cases. Parse guiding enables us to also handle queries that are *sufficiently similar* to known problematic queries. Concretely, we can use the similarity score from CASPER's retriever to identify whether the input is similar to any problematic examples, and apply parse guiding toward them if it is.

5.1 Setup

We focus on the usage of guiding tag (Section 2.1) for parse guiding. Trained correctly, the parser should become more faithful toward the exemplars when the guiding tag is present in the augmented query, and should parse normally otherwise.

To evaluate this parse guiding ability, we define an *oracle evaluation set* consisting of examples (x, E, y) with a predefined list of exemplars E. The MRs y'_i in E are restricted to having the same semantic template as y. On this evaluation, we expect the model's accuracy to rise when the guiding tag is present. The template accuracy, which is now equivalent to the rate where the prediction follows the template of y'_i, should also increase.

Methods We compare CASPER that was taught the behavior of the guiding tag against the models without such knowledge. We report the results on the standard and oracle evaluation sets, with and without the guiding tag added at test time.

5.2 Results

Table 5 shows the experimental results. On the standard evaluation set, CASPER model with the knowledge of guiding tag has a slightly smaller gain over T5. But when the guiding tag is present, the model becomes much more faithful to the given exemplars, as evidenced by the increased template and exact match accuracy on the oracle set.

Note that this gain is due to the guiding tag and not just the increased amount of training data: if we add oracle training data *without* guiding tag when training CASPER, the accuracy on the oracle set (90.74) is not as high as when we use the guiding tag (93.02). We also note that the guiding tag should only be used when the correct parse is expected to closely follow the exemplars. Using the tag on the standard set hurts the accuracy.

Method	+ tag at test time	Standard Exact	Oracle Exact	Oracle Template
T5	-	83.18	-	-
CASPER	-	84.29	88.18	91.96
+ oracle train	no	83.91	89.26	93.19
	yes	<u>80.58</u>	**93.02**	**97.74**

Table 5: **Parse guiding:** exact match and template accuracy on the standard and oracle evaluation sets. (<u>underlined</u> = worse than baseline)

(a)

x: call Nicholas and Natasha
x'_2: *PLATINUM* How do you make chicken spaghetti
y'_2: [IN get recipes =
[SL recipes included ingredient = chicken]
[SL recipes dish = spaghetti]]

Gold: [IN create call = [SL contact = Nicholas]
[SL contact = Natasha]]
C_o: [IN get recipes =
[SL recipes included ingredient = Nicholas]
[SL recipes included ingredient = Natasha]]

(b)

x: What's the work address with zipcode where James work?
x'_1: *PLATINUM* create alarm for 6h35
y'_1: [IN create alarm = [SL date time = for 6h35]]

Gold: [IN get location = [SL contact = James]]
C_o: [IN create alarm = [SL location =
[IN get location = [SL contact = James]]]]

Figure 5: Predictions of $\text{CASPER}_{\text{orig}}$ (C_o) trained on the mix of standard and oracle training data when given adversarial exemplars. (1 out of 5 exemplars are shown; *PLATINUM* is the guiding tag.)

When the guiding tag is present, the model needs to balance between being faithful to the exemplars and generating a sane parse. As an analysis, we try supplying exemplars with a drastically different template from the gold parse. The first example from Figure 5 shows how the model attempts to fit the two names from the query as two slot values. The second example shows how the model refuses to predict a SL:DATE_TIME slot, despite the guiding tag being present, since the query does not contain a suitable value for such a slot.

6 Schema refactoring experiments

In this section, we show how CASPER can adapt to changes in the semantic schema. Although the solution involves modifying the retrieval index like domain bootstrapping (Section 4), schema refactor-

Method	Pre-refactoring	Post-refactoring
T5	83.27	69.59
CASPER	84.52	81.21
$\text{CASPER}_{\text{orig}}$	83.50	78.52
Models with knowledge of guiding tag		
CASPER	83.89	**81.56**
$\text{CASPER}_{\text{orig}}$	84.34	79.72

Table 6: **Schema refactoring:** Both mixing in anonymized training data and using guiding tags help CASPER achieves the best post-refactor accuracy without hurting the pre-refactor accuracy.

ing presents a new challenge: the parser now needs to produce a different output for *in-domain* queries, and must resist the urge to produce semantic labels it has learned during training.

6.1 Setup

We consider a *label splitting* scenario where some semantic labels split into two labels each at test time. Following Gaddy et al. (2020), we simulate the scenario backward by using the original dataset as post-refactoring data, and merge 10 pairs of similar labels (listed in Appendix C) to form the pre-refactoring data. About 35% of development examples contain at least one label involved in label splitting, and about half of which have their MRs altered after refactoring.

Methods At test time, we replace the retrieval index with post-refactoring training data. For models with the knowledge of guiding tag, we add the guiding tag whenever a retrieved exemplar contains a label involved in label splitting.

6.2 Results

Table 6 shows the exact match accuracy on the original and refactored development sets. The baseline T5, which cannot incorporate the changed schema, suffers a 13.7% drop in exact match accuracy. The $\text{CASPER}_{\text{orig}}$ model, which leans toward memorizing labels more than utilizing exemplars, has a modest improvement upon T5. Mixing in anonymized examples during training and using the guiding tag make CASPER achieve a high post-refactoring accuracy, while also maintaining the pre-refactoring accuracy compared to the baseline T5.

7 Related works

7.1 Methods for few-shot tasks

CASPER belongs to the family of methods that adapt to new labels, domains, or tasks based on a handful of examples. If such examples are available at training time, one could fine-tune the model on them, in which case data augmentation (Jia and Liang, 2016; Kumar et al., 2019; Andreas, 2020; Lee et al., 2021) can be used to amplify the data for the new task. When the few-shot examples are only available at test time, the task is more difficult, and common approaches in the literature include *metric learning*, *fast update*, and *exemplar augmentation*.

Metric learning The main idea of metric learning (Koch et al., 2015; Vinyals et al., 2016; Snell et al., 2017) is to learn a representation of objects (either inputs or labels) such that objects in the same class are closer together. Test inputs are then matched to the representation of the few-shot labels or their exemplars.

Metric learning was first applied on classification tasks (Fritzler et al., 2019; Sun et al., 2019; Zhang et al., 2020). Subsequent studies extended metric learning to sequence labeling and semantic parsing by either matching tokens (Hou et al., 2020a,b; Zhu et al., 2020; Krone et al., 2020) or matching spans (Henderson and Vulić, 2021; Yu et al., 2021). However, such rigid notions of substructure matching do not lend themselves to complex hierarchical outputs. In CASPER, the retriever performs *query-level* matching to retrieve exemplars. While the exemplars may not be exactly in the same class as the query, the generator can implicitly reason with them when making predictions. This allows us to generate complex outputs while still gaining benefits from metric learning.

Fast update Given the few-shot examples, one could spend a small amount of resource to fine-tune on them for a few training steps. This creates a trade-off between the amount of resource spent and the performance on the new task. A common way to improve this trade-off is via *meta learning* (Finn et al., 2017; Ravi and Larochelle, 2017; Li et al., 2017). The main idea is to simulate fast update scenarios during training, and update the model's parameters so that the model performs fast updates more efficiently. Fast update with meta learning has been applied to NLP models for generalizing to unseen tasks or domains (Gu et al., 2018a; Dou et al., 2019; Bansal et al., 2020; Chen et al., 2020; Athiwaratkun et al., 2020; Wang et al., 2021).

Since fast update explicitly minimizes the loss on the few-shot examples, the updated model is more likely to be faithful toward them, whereas CASPER

requires additional techniques to increase faithfulness toward exemplars (Section 2.1). Nevertheless, CASPER has several advantages over fast update. For instance, while fast update needs to save the information about new labels into the parameters and recall it when parsing test queries, CASPER can directly access the new labels in the exemplars when parsing test queries. Compared to meta learning, training CASPER is also much simpler, only requiring off-the-shelf seq2seq fine-tuning. Finally, while fast update requires the new data to be input-output pairs to fine-tune on, CASPER's exemplars can technically be any information (e.g., new semantic schema) that can be augmented to the query.

Exemplar augmentation Our work is not the first to use exemplar augmentation for few-shot tasks (Radford et al., 2019; Zhao et al., 2021). The most prominent previous work is GPT-3 (Brown et al., 2020), which can perform new tasks by augmenting exemplars or task description to the query, even without further fine-tuning the model to specifically handle such augmented queries.

The approach most similar to ours is Liu et al. (2021), which also retrieves exemplars from a retrieval index. While Liu et al. (2021) focuses on improving the generative models on the standard evaluations, our work proposes how to use retrieval augmentation for controlling the behavior of the generator, which leads to novel use cases (domain bootstrapping, parse guiding, schema refactoring) on top of achieving state-of-the-art on the standard evaluation.

7.2 Discussion

Issues in domain bootstrapping The most straightforward method to adapt a neural model to new domains is to fine-tune it on new training examples. However, this approach not only has a high computation cost, but also suffers from two critical issues. One is *catastrophic forgetting*: the inability to preserve previous knowledge (McCloskey and Cohen, 1989; Goodfellow et al., 2014). The other is *model churn*: instability of model predictions on individual examples after fine-tuning. Existing work commonly tackles catastrophic forgetting via incremental training, such as imposing constraints on the distance between new and old models (Sarwar et al., 2019; Rosenfeld and Tsotsos, 2018) or jointly learn a generator to reply past examples for training (Hu et al., 2018). Another existing approach is to identify conflicting data to improve robustness of model updates (Gaddy et al., 2020). In CASPER, having the retrieval index that stores training examples mitigates catastrophic forgetting by design. And since the model can be controlled without fine-tuning, model churn is reduced.

Retrieval-augmented generation Recent studies have shown the effectiveness of retrieval augmentation in many generative NLP tasks. How the model actually uses the retrieved information differs among the methods. Some methods, like CASPER, encode the retrievals alongside the query and let the model decides how to use them (Guu et al., 2018; Hashimoto et al., 2018; He et al., 2020; Weston et al., 2018; Pandey et al., 2018; Lewis et al., 2020). Some utilize alignments between the retrieved examples and the input (Sumita and Iida, 1991; Gu et al., 2018b; Lu et al., 2019). And some use the retrievals to explicitly manipulate the token scores at each decoding step (Zhang et al., 2018; Hayati et al., 2018; Peng et al., 2019; Khandelwal et al., 2020, 2021).

Controllable generation Several works on controllable generation make use of conditional VAEs, where the latent variable conditioned on the input is the indicator for controlling the output (Hu et al., 2017; Shen et al., 2018; Zhang et al., 2019; Song et al., 2019; Shu et al., 2020). Other types of control indication include special input tokens (Keskar et al., 2019; Dathathri et al., 2020) or using another neural model as a style discriminator during decoding (Krause et al., 2020). Our work use *exemplars* as the indicator for controlling the prediction.

8 Conclusion

We have presented CASPER, a retrieval-augmented semantic parser that uses the retrieved exemplars to influence the predictions. By manipulating the retrieval index and how the exemplars are augmented, we can control the parser's behavior, which is helpful for domain bootstrapping, parse guiding, and schema refactoring.

Future works include fine-tuning the retriever, possibly jointly with the generator, which has potential to improve the model (see Section 3.3); introducing more fine-grained control on the faithfulness toward exemplars than the presence/absence of guiding tag; and pre-training the model on external resources to increase generalization.

9 Ethical considerations

This paper proposes a retrieval-augmented semantic parser, the predictive behavior of which can be changed by editing the retrieval index or how the retrieval-augmented query is constructed. These modifications can only be carried out by the developer of the parser, and not by the users who issue the queries. The intended use cases of our work include: (1) adding support for new query domains to the parser; (2) overriding predictions of a subset of queries, such as sensitive queries or queries that the parser struggles on; and (3) adapting the parser to an updated semantic schema.

Our method reduces the computational resources needed to retrain the model when enacting the new behavior. That said, the parser needs to be initially trained to recognize retrieval-augmented queries (which can be expensive), and retraining would be required for drastic changes in behavior (e.g., renaming multiple high-frequency semantic labels at once).

While the experiments were done on the English portion of the MTOP dataset, the method is generic to the language of the queries and meaning representations. Note that the model performance would depend on whether the underlying pre-trained retriever and generator models support the target languages well.

Acknowledgements

We want to thank Pete Shaw, Emily Pitler, and the reviewers for helpful comments and suggestions.

References

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

Ben Athiwaratkun, Cicero Nogueira dos Santos, Jason Krone, and Bing Xiang. 2020. Augmented natural language for generative sequence labeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 375–385, Online. Association for Computational Linguistics.

Trapit Bansal, Rishikesh Jha, and Andrew McCallum. 2020. Learning to few-shot learn across diverse natural language classification tasks. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 5108–5123, Barcelona, Spain (Online). International Committee on Computational Linguistics.

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

Daniel Matthew Cer, Yinfei Yang, Sheng yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, C. Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Universal sentence encoder. *ArXiv*, abs/1803.11175.

Xilun Chen, Asish Ghoshal, Yashar Mehdad, Luke Zettlemoyer, and Sonal Gupta. 2020. Low-resource domain adaptation for compositional task-oriented semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5090–5100, Online. Association for Computational Linguistics.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations (ICLR)*.

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.

Zi-Yi Dou, Keyi Yu, and Antonios Anastasopoulos. 2019. Investigating meta-learning algorithms for low-resource natural language understanding tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1192–1197, Hong Kong, China. Association for Computational Linguistics.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*.

Alexander Fritzler, Varvara Logacheva, and Maksim Kretov. 2019. Few-shot classification in named entity recognition task. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC)*.

David Gaddy, Alex Kouzemtchenko, Pavankumar Reddy Muddireddy, Prateek Kolhar, and Rushin Shah. 2020. Overcoming conflicting data for model updates. *ArXiv*, abs/2010.12675.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. 2014. An empirical investigation of catastrophic forgetting in gradient-based neural networks. In *International Conference on Learning Representations (ICLR)*.

Jiatao Gu, Yong Wang, Yun Chen, Victor O. K. Li, and Kyunghyun Cho. 2018a. Meta-learning for low-resource neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3622–3631, Brussels, Belgium. Association for Computational Linguistics.

Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor O. K. Li. 2018b. Search engine guided neural machine translation. In *AAAI Conference on Artificial Intelligence*.

Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning (ICML)*.

Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis. 2018. Semantic parsing for task oriented dialog using hierarchical representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2787–2792, Brussels, Belgium. Association for Computational Linguistics.

Kelvin Guu, Tatsunori B. Hashimoto, Yonatan Oren, and Percy Liang. 2018. Generating sentences by editing prototypes. *Transactions of the Association for Computational Linguistics*, 6:437–450.

Tatsunori B Hashimoto, Kelvin Guu, Yonatan Oren, and Percy Liang. 2018. A retrieve-and-edit framework for predicting structured outputs. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Shirley Anugrah Hayati, Raphael Olivier, Pravalika Avvaru, Pengcheng Yin, Anthony Tomasic, and Graham Neubig. 2018. Retrieval-based neural code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 925–930, Brussels, Belgium. Association for Computational Linguistics.

Junxian He, Taylor Berg-Kirkpatrick, and Graham Neubig. 2020. Learning sparse prototypes for text generation. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Matthew Henderson and Ivan Vulić. 2021. ConVEx: Data-efficient and few-shot slot labeling. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3375–3389, Online. Association for Computational Linguistics.

Yutai Hou, Wanxiang Che, Yongkui Lai, Zhihan Zhou, Yijia Liu, Han Liu, and Ting Liu. 2020a. Few-shot slot tagging with collapsed dependency transfer and label-enhanced task-adaptive projection network. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1381–1393, Online. Association for Computational Linguistics.

Yutai Hou, Jiafeng Mao, Yongkui Lai, Cheng Chen, Wanxiang Che, Zhigang Chen, and Ting Liu. 2020b. FewJoint: A few-shot learning benchmark for joint language understanding. *ArXiv*, abs/2009.08138.

Wenpeng Hu, Zhou Lin, Bing Liu, Chongyang Tao, Zhengwei Tao, Jinwen Ma, Dongyan Zhao, and Rui Yan. 2018. Overcoming catastrophic forgetting for continual learning via model adaptation. In *International Conference on Learning Representations (ICLR)*.

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. 2017. Toward controlled generation of text. In *International Conference on Machine Learning (ICML)*.

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.

Jeff Johnson, M. Douze, and H. Jégou. 2021. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7:535–547.

N. Keskar, Bryan McCann, L. Varshney, Caiming Xiong, and Richard Socher. 2019. CTRL: A conditional transformer language model for controllable generation. *ArXiv*, abs/1909.05858.

Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2021. Nearest neighbor machine translation. In *International Conference on Learning Representations (ICLR)*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations (ICLR)*.

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*.

Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq R. Joty, Richard Socher, and Nazneen Fatema Rajani. 2020. GeDi: Generative discriminator guided sequence generation. *ArXiv*, abs/2009.06367.

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.

Jason Krone, Yi Zhang, and Mona Diab. 2020. Learning to classify intents and slot labels given a handful of examples. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pages 96–108, Online. Association for Computational Linguistics.

Varun Kumar, Hadrien Glaude, Cyprien de Lichy, and Wlliam Campbell. 2019. A closer look at feature space data augmentation for few-shot intent classification. In *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, pages 1–10, Hong Kong, China. Association for Computational Linguistics.

Kenton Lee, Kelvin Guu, Luheng He, Timothy Dozat, and Hyung Won Chung. 2021. Neural data augmentation via example extrapolation. *ArXiv*, abs/2102.01335.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Haoran Li, Abhinav Arora, Shuohui Chen, Anchit Gupta, Sonal Gupta, and Yashar Mehdad. 2021a. MTOP: A comprehensive multilingual task-oriented semantic parsing benchmark. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2950–2962, Online. Association for Computational Linguistics.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. 2017. Meta-SGD: Learning to learn quickly for few shot learning. *ArXiv*, abs/1707.09835.

Zhuang Li, Lizhen Qu, Shuo Huang, and Gholamreza Haffari. 2021b. Few-shot semantic parsing for new predicates. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1281–1291, Online. Association for Computational Linguistics.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What makes good in-context examples for GPT-3? *ArXiv*, abs/2101.06804.

Zhichu Lu, Forough Arabshahi, Igor Labutov, and Tom Mitchell. 2019. Look-up and adapt: A one-shot semantic parser. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1129–1139, Hong Kong, China. Association for Computational Linguistics.

Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

Gaurav Pandey, Danish Contractor, Vineet Kumar, and Sachindra Joshi. 2018. Exemplar encoder-decoder for neural conversation generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1329–1338, Melbourne, Australia. Association for Computational Linguistics.

Hao Peng, Ankur Parikh, Manaal Faruqui, Bhuwan Dhingra, and Dipanjan Das. 2019. Text generation with exemplar-based adaptive decoding. 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 2555–2565, Minneapolis, Minnesota. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Technical report, OpenAI.

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

Sachin Ravi and Hugo Larochelle. 2017. Optimization as a model for few-shot learning. In *International Conference on Learning Representations (ICLR)*.

Amir Rosenfeld and John K Tsotsos. 2018. Incremental learning through deep adaptation. *IEEE transactions on pattern analysis and machine intelligence*, 42(3):651–663.

Syed Shakib Sarwar, Aayush Ankit, and Kaushik Roy. 2019. Incremental learning in deep convolutional neural networks using partial network sharing. *IEEE Access*, 8:4615–4628.

Xiaoyu Shen, Hui Su, Shuzi Niu, and Vera Demberg. 2018. Improving variational encoder-decoders in dialogue generation. In *AAAI Conference on Artificial Intelligence*.

Lei Shu, Alexandros Papangelis, Yi-Chia Wang, Gokhan Tur, Hu Xu, Zhaleh Feizollahi, Bing Liu, and Piero Molino. 2020. Controllable text generation with focused variation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3805–3817, Online. Association for Computational Linguistics.

Jake Snell, Kevin Swersky, and Richard S. Zemel. 2017. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems (NIPS)*.

Haoyu Song, Weinan Zhang, Yiming Cui, Dong Wang, and Ting Liu. 2019. Exploiting persona information for diverse generation of conversational responses. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Yu Su and Xifeng Yan. 2017. Cross-domain semantic parsing via paraphrasing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1235–1246, Copenhagen, Denmark. Association for Computational Linguistics.

Eiichiro Sumita and Hitoshi Iida. 1991. Experiments and prospects of example-based machine translation. In *29th Annual Meeting of the Association for Computational Linguistics*, pages 185–192, Berkeley, California, USA. Association for Computational Linguistics.

Shengli Sun, Qingfeng Sun, Kevin Zhou, and Tengchao Lv. 2019. Hierarchical attention prototypical networks for few-shot text classification. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 476–485, Hong Kong, China. Association for Computational Linguistics.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. 2016. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems (NIPS)*.

Bailin Wang, Mirella Lapata, and Ivan Titov. 2021. Meta-learning for domain generalization in semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 366–379, Online. Association for Computational Linguistics.

Jason Weston, Emily Dinan, and Alexander Miller. 2018. Retrieve and refine: Improved sequence generation models for dialogue. In *Proceedings of the 2018 EMNLP Workshop SCAI: The 2nd International Workshop on Search-Oriented Conversational AI*, pages 87–92, Brussels, Belgium. 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.

Dian Yu, Luheng He, Yuan Zhang, Xinya Du, Panupong Pasupat, and Qi Li. 2021. Few-shot intent classification and slot filling with retrieved examples. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 734–749, Online. Association for Computational Linguistics.

Jingyi Zhang, Masao Utiyama, Eiichro Sumita, Graham Neubig, and Satoshi Nakamura. 2018. Guiding neural machine translation with retrieved translation pieces. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1325–1335, New Orleans, Louisiana. Association for Computational Linguistics.

Tao Zhang, Congying Xia, Chun-Ta Lu, and Philip Yu. 2020. MZET: Memory augmented zero-shot fine-grained named entity typing. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 77–87, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Yuchi Zhang, Yongliang Wang, Liping Zhang, Zhiqiang Zhang, and Kun Gai. 2019. Improve diverse text generation by self labeling conditional variational auto encoder. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Tony Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *International Conference on Machine Learning (ICML)*.

Su Zhu, Ruisheng Cao, Lu Chen, and Kai Yu. 2020. Vector projection network for few-shot slot tagging in natural language understanding. *ArXiv*, abs/2009.09568.

A Training details

Data preprocessing We used the entire English portion of the MTOP dataset (Li et al., 2021a), which contains 15667 training, 2235 development, and 4386 test queries. For the input queries, we space-concatenated the tokens from the official tokenization. The output MRs from the dataset file were preprocessed according to the description in Section 3.1: the intent and slot labels are lowercase and word-split at underscores (e.g.,

[IN:CREATE_CALL ...] becomes [IN create call = ...]). We also removed spaces before the "]" tokens.

Retriever To embed the queries, we used pre-trained Universal Sentence Encoder (Cer et al., 2018). Specifically, we used the large version of the encoder.[5] The embedder was kept fixed. We computed the embeddings for all queries on CPU. For each query, we cached 100 exemplars with the least cosine embedding distance from the query. The selection of top-k and sampled-k exemplars were only done on these cached exemplars.

In actual deployment, brute force retrieval might be too slow, and fast nearest neighbor methods (Johnson et al., 2021; Guo et al., 2020) could be used to speed up the retriever.

Training For each original example (x, y), we generated 20 lists E of sampled-5 exemplars, and saved them to dataset files for fine-tuning the T5 (Raffel et al., 2020) generator model. We used the base version of the model (220M parameters). We selected reasonable hyperparameter values and performed some minimal hyperparameter tuning. Specifically, we used a batch size of 4096 and the learning rate of 0.001. Training is done for 2000 steps, with early stopping based on the exact match accuracy on the development data. We fine-tuned T5 on 32 Cloud TPU v3 cores. Training takes approximately 2.5 hours.

We ran the experiments on 3 random seeds. One exception is the domain bootstrapping experiments, where we ran on 1 seed for each of the 5 domains and averaged the results.

Fast update For the fast update experiments in Section 4, we start from the T5 model fine-tuned on $\mathcal{O}_{\text{train}}$, and then continue to fine-tune it on either $\mathcal{N}_{\text{sup}}$ or an equal mix of $\mathcal{O}_{\text{train}}$ and $\mathcal{N}_{\text{sup}}$ (i.e., 50% chance of picking an example from $\mathcal{O}_{\text{train}}$; 50% chance of picking an example from $\mathcal{N}_{\text{sup}}$). We use a batch size of 128 here instead of 4096. Since $|\mathcal{N}_{\text{sup}}| = 100$, each iteration goes over the support set approximately once when fine-tuning on $\mathcal{N}_{\text{sup}}$.

B Detailed experimental results

Table 7, 8, and 9 show detailed results of the standard, parse guiding, and schema refactoring experiments. Table 10 shows per-domain results for the domain bootstrapping experiments. We note that T5 got a non-trivial accuracy on the no-fine-tuning setting of the *calling* domain. This is because many training queries in the *reminder* domain have IN:CREATE_CALL, a main intent of *calling*, nested inside (e.g., "Delete reminder to *call husband*").

C Labels in the schema refactoring experiments

Table 11 lists the affected labels in the schema refactoring experiments (Section 6), along with their frequencies in the original training data. Note that the training data contains 15667 examples.

[5] https://tfhub.dev/google/universal-sentence-encoder-large/5

Method	Dev		Test	
	Exact	Template	Exact	Template
T5	83.18 ±0.12	87.22 ±0.07	85.06 ±0.25	88.70 ±0.15
CASPER	84.29 ±0.25	87.65 ±0.31	85.54 ±0.09	89.13 ±0.15
$\text{CASPER}_{\text{orig}}$	84.67 ±0.13	87.98 ±0.12	86.36 ±0.12	89.65 ±0.08
$\text{CASPER}_{\text{anon}}$	79.61 ±0.17	82.60 ±0.00	80.85 ±0.05	83.90 ±0.21

Table 7: Detailed results on the standard setup (averaged over 3 runs; ± standard deviation).

Method	+ tag at test time	Standard	Oracle	
		Exact	Exact	Template
T5	-	83.18 ±0.12	-	-
CASPER	-	84.29 ±0.25	88.18 ±0.40	91.96 ±0.23
+ oracle train	no	83.91 ±0.07	89.26 ±0.85	93.19 ±0.83
	yes	80.58 ±0.16	93.02 ±0.21	97.74 ±0.23

Table 8: Detailed results on parse guiding (averaged over 3 runs; ± standard deviation)

Method	Pre-refactoring	Post-refactoring
T5	83.27 ±0.21	69.59 ±0.11
CASPER	84.52 ±0.09	81.21 ±0.21
$\text{CASPER}_{\text{orig}}$	83.50 ±0.27	78.52 ±0.16
Models with knowledge of guiding tag		
CASPER	83.89 ±0.12	81.56 ±0.20
$\text{CASPER}_{\text{orig}}$	84.34 ±0.18	79.72 ±0.18

Table 9: Detailed results on schema refactoring (averaged over 3 runs; ± standard deviation).

Setting:	standard		seen-boot.		unseen-boot.	
Train data:	$\mathcal{O}_{\text{train}} + \mathcal{N}_{\text{train}}$		$\mathcal{O}_{\text{train}} + \mathcal{N}_{\text{sup}}$		$\mathcal{O}_{\text{train}}$	
Eval data:	$\mathcal{N}_{\text{dev}}$	$\mathcal{O}_{\text{dev}}$	$\mathcal{N}_{\text{dev}}$	$\mathcal{O}_{\text{dev}}$	$\mathcal{N}_{\text{dev}}$	$\mathcal{O}_{\text{dev}}$
alarm						
T5	83.87	83.21	75.81	82.77	1.61	83.21
CASPER	86.56	83.94	77.96	84.14	63.98	84.14
$\text{CASPER}_{\text{orig}}$	87.63	84.48	77.96	84.53	62.37	84.38
$\text{CASPER}_{\text{anon}}$	80.11	79.60	72.58	79.65	65.59	80.14
calling						
T5	94.22	81.37	72.04	81.37	24.92	81.53
CASPER	94.53	82.27	74.16	82.95	48.33	82.63
$\text{CASPER}_{\text{orig}}$	95.14	82.95	74.47	82.16	43.47	82.95
$\text{CASPER}_{\text{anon}}$	90.88	77.54	64.13	77.65	55.93	77.81
event						
T5	92.68	82.77	82.93	82.72	0.00	82.48
CASPER	92.68	83.57	82.11	83.43	68.29	83.76
$\text{CASPER}_{\text{orig}}$	92.68	84.28	83.74	83.71	65.04	84.23
$\text{CASPER}_{\text{anon}}$	89.43	78.84	71.54	78.74	71.54	79.12
messaging						
T5	94.89	82.27	74.43	82.13	1.70	81.88
CASPER	94.89	83.15	77.27	83.15	30.68	83.63
$\text{CASPER}_{\text{orig}}$	96.02	83.78	77.27	83.29	21.02	83.58
$\text{CASPER}_{\text{anon}}$	90.91	78.48	63.07	78.44	42.05	78.34
music						
T5	72.46	84.52	48.31	84.66	0.00	84.57
CASPER	74.40	85.16	52.17	85.01	8.21	85.21
$\text{CASPER}_{\text{orig}}$	75.36	85.70	53.14	85.70	3.86	85.21
$\text{CASPER}_{\text{anon}}$	71.01	80.42	43.96	80.97	33.82	80.72

Table 10: Domain bootstrapping results on each domain.

Pre-refactoring Label	Post-refactoring Label	Count
IN:GET_EVENT	IN:GET_EVENT	724
	IN:GET_REMINDER	335
SL:TYPE_RELATION	SL:TYPE_RELATION	1294
	SL:TYPE_CONTENT	330
IN:GET_MESSAGE	IN:GET_MESSAGE	220
	IN:GET_TODO	281
SL:MUSIC_PLAYLIST_TITLE	SL:MUSIC_PLAYLIST_TITLE	96
	SL:MUSIC_PROVIDER_NAME	265
SL:RECIPES_SOURCE	SL:RECIPES_SOURCE	9
	SL:RECIPES_COOKING_METHOD	234
IN:SWITCH_CALL	IN:SWITCH_CALL	49
	IN:UPDATE_CALL	218
IN:GET_CONTACT	IN:GET_CONTACT	1537
	IN:GET_LOCATION	217
IN:SET_AVAILABLE	IN:SET_AVAILABLE	52
	IN:GET_AVAILABILITY	214
SL:SCHOOL	SL:SCHOOL	153
	SL:EMPLOYER	204
SL:NEWS_TOPIC	SL:NEWS_TOPIC	617
	SL:NEWS_CATEGORY	201

Table 11: Labels for the schema refactoring experiments.