Cross-TOP: Zero-Shot Cross-Schema Task-Oriented Parsing

Melanie Rubino
Amazon Alexa AI
New York, USA
rubinome@amazon.com

Nicolas Guenon des Mesnards
Amazon Alexa AI
New York, USA
mesnarn@amazon.com

Uday Shah
Amazon Alexa AI
New York, USA
shahuda@amazon.com

Nanjiang Jiang
Department of Linguistics
The Ohio State University
jiang.1879@osu.edu

Weiqi Sun
Amazon Alexa AI
New York, USA
weiqisun@amazon.com

Konstantine Arkoudas
Amazon Alexa AI
New York, USA
arkoudk@amazon.com

Abstract

Deep learning methods have enabled task-oriented semantic parsing of increasingly complex utterances. However, a single model is still typically trained and deployed for each task separately, requiring labeled training data for each, which makes it challenging to support new tasks, even within a single business vertical (e.g., food-ordering or travel booking). In this paper we describe Cross-TOP (Cross-Schema Task-Oriented Parsing), a zero-shot method for complex semantic parsing in a given vertical. By leveraging the fact that user requests from the same vertical share lexical and semantic similarities, a single cross-schema parser is trained to service an arbitrary number of tasks, seen or unseen, within a vertical. We show that Cross-TOP can achieve high accuracy on a previously unseen task without requiring any additional training data, thereby providing a scalable way to bootstrap semantic parsers for new tasks. As part of this work we release the FoodOrdering dataset, a task-oriented parsing dataset in the food-ordering vertical, with utterances and annotations derived from five schemas, each from a different restaurant menu.

1 Introduction

Propelled by deep learning, task-oriented parsing has made significant strides, moving away from flat intents and slots towards more complex tree-based semantics that can represent compositional meaning structures (Gupta et al., 2018; Aghajanyan et al., 2020; Rongali et al., 2020; Mansimov and Zhang, 2021). However, most semantic parsing systems remain task-specific: they can only produce representations with the set of intents and slots seen during training. To support multiple tasks, this approach requires collecting data, training, and maintaining a model for each task separately. This is costly when multiple tasks need to be supported, as is usually the case for digital voice assistants such as Alexa and Google Assistant, which may need to support hundreds or thousands of different tasks in a given business vertical (e.g., restaurants in the food-ordering vertical, hotels in the travel vertical, and so on).

In this paper we present Cross-TOP, a method for building a single semantic parsing model that can support an arbitrary number of tasks in a given vertical. User requests pertaining to the same vertical have lexical and semantic similarities; their main differences lie in their unique schemas. In the food-ordering domain, for example, a customer may request a main dish with various options and possibly a drink and a side. However, depending on the specific restaurant menu, the output semantic representations can differ greatly; see Figure 1.

Cross-TOP makes use of a powerful pre-trained transformer-based encoder-decoder language model, with schema-specific context added to the input along with the utterance. In this way, the model learns to generate parses for a new, unseen task, by attending to the schema in the input rather than by needing to see it during training. We show that this approach is quite effective and provides a quick solution to the practical problem of bootstrapping semantic parsers for new tasks within a vertical, using a single model in production.

The parser is trained on a number of initial tasks, where each task has some training data available. Moreover, we assume that every task has a unique

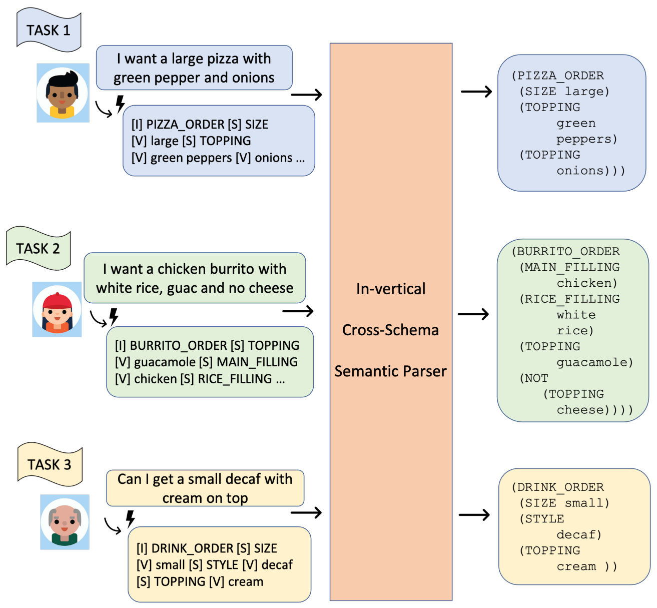

TASK 1
I want a large pizza with green pepper and onions
[I] PIZZA_ORDER [S] SIZE [V] large [S] TOPPING [V] green peppers [V] onions ...
(PIZZA_ORDER (SIZE large) (TOPPING green peppers) (TOPPING onions)))
TASK 2
I want a chicken burrito with white rice, guac and no cheese
[I] BURRITO_ORDER [S] TOPPING [V] guacamole [S] MAIN_FILLING [V] chicken [S] RICE_FILLING ...
(BURRITO_ORDER (MAIN_FILLING chicken) (RICE_FILLING white rice) (TOPPING guacamole) (NOT (TOPPING cheese))))
TASK 3
Can I get a small decaf with cream on top
[I] DRINK_ORDER [S] SIZE [V] small [S] STYLE [V] decaf [S] TOPPING [V] cream
(DRINK_ORDER (SIZE small) (STYLE decaf) (TOPPING cream))
In-vertical Cross-Schema Semantic Parser

Figure 1: The Cross-TOP parser processes utterances from multiple tasks with different schemas. The lightning bolts represent fuzzy matching, which is used to append schema elements to the input (cf. Section 2).

schema. That schema consists of all possible intents and slots for the task at hand; both intents and slots can be arbitrarily nested (compositional). For every slot S, the schema also includes natural-language phrases for the various values of S. All schemas for the five tasks in our dataset can be found in Appendix B. Cross-TOP uses constrained decoding to ensure that it generates well-formed parses that can be resolved to executable representations that can be directly used by the back end.

Most zero-shot cross-schema semantic parsing work has been in the context of the Text-to-SQL task (Zhong et al., 2020; Lin et al., 2020; Wang et al., 2020; Rubin and Berant, 2021; Yu et al., 2020; Gan et al., 2021). Cross-schema task-oriented parsing introduces its own challenges. In SQL, the schemas are database schemas, and the parser is trained on some initial databases and then evaluated on another database. There is a lot of invariant structure across different tasks in the output space (since output sequences are always SQL queries), as well as common patterns in how SQL structures tend to align with natural language. However, for the schemas defined in task-oriented parsing for the food-ordering domain, the only invariant structures are the parentheses and some lexical overlap among the intents and slots. Therefore, cross-schema parsing in general is more challenging for task-oriented parsing. However, restricting the scope to a given vertical imposes more common structure that can prove helpful.

To evaluate our methodology, we focus on tasks in the food-ordering domain, where each task contains examples from a restaurant with the schema generated from its menu. Our main contributions are as follows:

- We present a new technique for zero-shot intra-vertical cross-schema semantic parsing that jointly encodes utterance tokens and schema elements.
- We release a new task-oriented parsing dataset for food ordering to evaluate similar efforts. The FoodOrdering dataset includes examples from five restaurants, totaling close to 30,000 synthetically-generated training examples and 963 human-generated test utterances with labels.
- We show that our method achieves up to 73% exact match accuracy on a previously unseen ordering task, proving the method's viability for effortlessly handling a new task.

2 Model

Our method trains a single schema-aware model to serve multiple tasks and bootstrap new ones from the same business vertical in a zero-shot setting. It leverages the transfer learning capabilities of a transformer-based pretrained encoder-decoder language model.

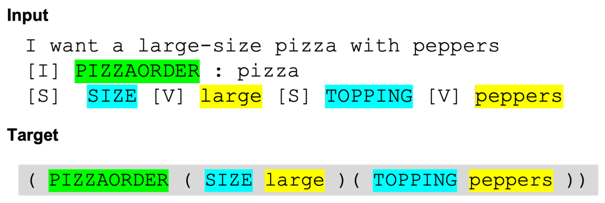
Terminology Each *task* is defined by a unique *schema* consisting of *intents*, *slots* belonging to those intents, and *catalogs* enumerating the possible *slot values* for each slot. For example, in the pizza-ordering task the `TOPPING` slot belongs to the `PIZZAORDER` intent, and values for this slot could be `mushrooms`, `pepperoni`, and so on. In our predefined catalogs, multiple slot values could refer to the same *slot value entity*, for example `peppers` and `green peppers` can both be mapped to `TOPPING_PEPPERS`—or perhaps `TOPPING_35`—in the back end. Cross-TOP predicts parse trees that contain slot values, which are then entity-resolved into those unique back-end identifiers through this many-to-1 mapping.

A task schema can optionally define *invocation keywords* for each intent, to identify how these are expressed in natural language, for example {*drink*, *drinks*} for a `DRINK_ORDER` intent. This is used for augmenting model input with *fuzzy-matched* schema elements later on. Fuzzy string-matching algorithms compute lexical similarity between strings. If some schema elements have a significant overlap with certain utterance tokens, then there is a "match" and that schema element will be appended to the input utterance before encoding.

Model Inputs As just mentioned, to achieve zero-shot cross-schema parsing, we append fuzzy-matched schema elements to input utterances. Given an utterance $\mathtt{u}$, assume our fuzzy-matching process (described later) determined that the intents $\mathtt{i}_1$ and $\mathtt{i}_2$ are present in the request, with slot/slot-values $\mathtt{s}_{1,1}/\mathtt{v}_{1,1}$ for $\mathtt{i}_1$, as well as $\mathtt{s}_{2,1}/\mathtt{v}_{2,1}$ and $\mathtt{s}_{2,2}/\mathtt{v}_{2,2}$ for $\mathtt{i}_2$[1]. The input to Cross-TOP is then serialized into the following format:

$\mathtt{u}$ `[I]` $\mathtt{i}_1$ `[S]` $\mathtt{s}_{1,1}$ `[V]` $\mathtt{v}_{1,1}$ `[I]` $\mathtt{i}_2$ `[S]` $\mathtt{s}_{2,1}$ `[V]` $\mathtt{v}_{2,1}$ `[S]` $\mathtt{s}_{2,2}$ `[V]` $\mathtt{v}_{2,2}$

where markers `[I]`, `[S]`, `[V]` indicate that the following tokens are intents, slots and slot values, respectively. An example is given in Figure 2.


Input

```
I want a large-size pizza with peppers
[I] PIZZAORDER : pizza
[S]  SIZE [V] large [S] TOPPING [V] peppers
```

Target

```
( PIZZAORDER ( SIZE large )( TOPPING peppers ))
```

Figure 2: Cross-TOP is trained to attend to input utterances augmented with fuzzy-matched schema elements.

Our format is inspired from BRIDGE (Lin et al., 2020), but instead of table/column/column-value in a database schema, task-oriented parsing schemas uses intent/slot/slot-value. While the longer input sequences increase the computation required for inference, the latency impact is mitigated by the parallelizability of the transformer architecture.

Model Outputs The model is trained to generate a linearized parse tree similar to the target shown in Figure 2, which is reminiscent of the TOP *decoupled* notation (Aghajanyan et al., 2020). TOP decoupled is itself derived from the TOP notation (Gupta et al., 2018) by removing tokens that are not direct children of slot nodes. Unlike TOP decoupled, leaf nodes in our output semantics are not tokens copied from the source utterance, but instead must be valid *slot values* belonging to the task's catalogs. As exemplified in Figure 2, the fuzzy-matched slot value for the utterance segment `large-size` is the catalog entry `large`. It can happen that utterance token and catalog value are identical, as is the case for `peppers` here. By predicting slot values instead of unresolved utterance tokens, Cross-TOP jointly learns to perform semantic parsing and entity resolution, thus eliminating the need to train and maintain a separate entity resolution system for every new task.

Fuzzy-Matching Details The viability of our schema-aware encoding depends on our ability to extract the proper schema elements. We leverage the fuzzy-matching method from the BRIDGE codebase[2] and compute lexical similarity scores between an input utterance and every slot value.[3] If multiple slot values representing the same entity match the utterance, we pick the one with the higher similarity score. Slots are added to the input if at least one of their slot values was added.[4] Intents are added to the input if at least one of their slots is added.[5]

In addition, if any of the predefined intent invocation keywords (cf. **Terminology**) fuzzy-match the utterance, then that intent is added along with the fuzzy-matched keyword, for example adding `[I] PIZZAORDER : pizza` instead of simply `[I] PIZZAORDER`. Given that intent names can be arbitrary and carry little semantic content, this design helps the pretrained language model by bridging the gap between natural language and back-end executable representations.

Constrained Decoding The target parses contain only schema elements and parentheses. Cross-TOP leverages constrained decoding at inference time to generate valid catalog values and parses according to the schema. For example, the string `(DRINK_ORDER (SIZE coke))` is not valid, as the slot value `coke` is not a catalog entry for the slot `SIZE`. In this work we also implement a

[1]There can be more than one slot value $\mathtt{v}$ identified for the same slot $\mathtt{s}$, in which case the input will be of the form: $\mathtt{u}$ `[I]` $\mathtt{i}_1$ `[S]` $\mathtt{s}_{1,1}$ `[V]` $\mathtt{v}_{1,1,1}$ `[V]` $\mathtt{v}_{1,1,2}$ ...

[2]https://github.com/salesforce/TabularSemanticParsing

[3]This works in a vertical with small catalogs, such as restaurant menus. To make it scale to much larger catalogs, one could use sub-linear fuzzy string-matching algorithms and offline parallel processing.

[4]Slots that are parents of other slots are also provided with catalog entries to allow fuzzy matching. For example, a `NOT` slot for negation will use {*with no*, *without*, *hold the* ...}.

[5]A slot shared across two intents will trigger both their inclusion, but experiments indicate that the neural parser can learn to discard such false detection.

parentheses-balancing constraint, as well as a set of valid next-token constraints, where each vocabulary subword has a corresponding entry in a dictionary mapping it to a list of valid subwords that may follow it. The content of such a dictionary is task-specific but is built programmatically from the task schema. The detailed constraints are provided in Appendix D. Section 5 quantifies the benefits of constrained decoding in the zero-shot setting.

3 The FoodOrdering Dataset

We release a dataset for cross-schema zero-shot task-oriented parsing: the FoodOrdering dataset,[6] comprising five food-ordering tasks for five fictitious restaurants: PIZZA, SUB, BURRITO, BURGER and COFFEE.

Dataset Construction To gauge zero-shot capabilities, only three out of five tasks come with training data. For SUB and BURRITO, the training portion of the data was synthetically generated by sampling around 50 human-designed templates for which slot values are themselves sampled from predefined catalogs. The catalogs and templates are released along with the dataset, but a couple of examples are given in Table 4. We generated up to 10,000 unique pairs of natural language and target parses. For PIZZA we randomly sampled 10,000 utterances out of the 2.5M provided by Arkoudas et al. (2021). All five tasks have evaluation data generated by humans and collected through Mechanical Turk; see Appendix A for details. MTurk workers generated natural language orders, which were then annotated internally. More examples can be found in Appendix C.

Dataset Statistics All tasks follow a common structure of intents and slots, but each task has a different number of intents, slots and slot values. In Table 1, the #SltValEntities column does not count the total number of slot values, but rather the total number of slot value entities, which are resolved slot values (cf. **Terminology**). BURRITO has 7 distinct intents while COFFEE is a single-intent task. The design differences between the task schemas reflect a real-world setting: each restaurant comes with its own preexisting back end that dictates the design and contents of the corresponding schema. On average there are 1.7 intents and 6.2 slots per utterance, and an average depth[7] of 3.4. Detailed numbers are provided in Table 5 of Appendix C.

Task Schemas Each task has a unique schema, but all schemas are governed by similar rules: only slot nodes can be children of intent nodes, and there is no limit on the number of intents per utterance nor slots per intent. Slot nodes can be parents either of slot values or of other slots. `NOT` is an example of a generic (task-agnostic) slot that allows us to negate any slot that admits negation, such as `TOPPING`. Refer to Appendix B for the details of the five schemas.

4 Experimental Setup

Our experimental setup reflects the practical scenario of having to scale a technology to service multiple applications under constrained production resources. We consider a single model to serve all tasks, so we train with synthetic data for only three of the tasks (PIZZA, BURRITO and SUB), and test zero-shot generalization on two unseen tasks (BURGER and COFFEE).

Training Details In this work we use BART-Large (Lewis et al., 2020), a transformer-based pretrained encoder-decoder language model. We finetune the publicly available 24-layer BART-Large checkpoint[8] totaling 406M parameters, using the `transformers` codebase. We expand the target vocabulary by adding special tokens for input markers `[I]`, `[S]` and `[V]`. The training dataset was created by concatenating synthetic data from the three training tasks. Models are trained for 50 epochs with early stopping patience of 4, using cross-entropy sequence loss and the `AdamW` optimizer. We use the human-generated data of the three training tasks as our development set for early stopping and hyperparameter tuning. Hyperparameter tuning is described in Appendix E. Our best model uses a batch size of 16, learning rate `1e-05` and linear learning rate scheduler with warm-up ratio of 0.1. The hyperparameter `no_repeat_ngram_size` was disabled by setting it to 0.

Evaluation Details We use Unordered Exact match accuracy (Unordered EM) to measure per-

[6]https://github.com/amazon-research/food-ordering-semantic-parsing-dataset

[7]Queries are by design multi-intent, hence implicitly rooted in a parent `ORDER` node, which is factored in the computation of depth.

[8]https://huggingface.co/facebook/bart-large

Dataset	#Train/Synthetic	#Eval	#Int	#Slt	#SltValEntities	Example utterance
PIZZA	10,000	348	2	10	166	*"Can i get one large pie with no cheese and a coke."*
BURRITO	9,982	191	7	11	34	*"One carnitas quesadilla with white rice and black beans."*
SUB	10,000	161	3	8	62	*"Get me a cold cut combo with mayo and extra pickles."*
BURGER	0*	161	3	9	44	*"A vegan burger with onions and a side of sweet potato fries."*
COFFEE	0*	104	1	9	43	*"One regular latte cinnamon iced with one extra espresso shot."*

Table 1: FoodOrdering dataset statistics: sizes of training and evaluation sets, as well as numbers of intents, slots, and resolved slot value entities defined in each task's schema. *BURGER and COFFEE have no training data, as they are used to evaluate zero-shot learning.

formance. It checks for an exact match between the golden and predicted trees, where sibling order does not matter. The golden parse trees are executable representations (ready for consumption by an appropriate back end) that contain resolved entity names instead of slot values identified by utterance segments. These entities are fully determined by the many-to-1 mapping mentioned in Section 2. Validation performance is computed on the aggregated validation sets for the three training tasks. Test performance is reported for tasks individually. We used a beam size of 6 for validation and testing.

Pre-Processing and Post-Processing When appending the schema elements to the input utterance we do not include the slot/slot-value pair `NUMBER`, `1` from the fuzzy matching process if it's the only quantity matched.[9] This choice was made after observing that the slot values `a/an` can easily trigger false positives in fuzzy matching. For example, in the utterance *an order of two sprites*, the numeric quantity to extract is *two*, but the token *an* would trigger an extra unnecessary match. At inference time, if no `NUMBER` was generated for an intent, we add back `(NUMBER 1)` as a default to the predicted parse tree. Before computing unordered EM scores, all slot values are resolved into the appropriate entity names using the many-to-1 mapping mentioned earlier.

5 Results and Analysis

In the zero-shot setting, Cross-TOP achieves 73% unordered EM on BURGER and 55% on COFFEE (Table 2). The rest of this section presents an analysis of our results.

Schema-aware encoding enables zero-shot transfer learning. The main strength of Cross-TOP is training and maintaining a single model that can serve multiple tasks within the same business vertical, and bootstrapping new tasks without retraining. The zero-shot results in Table 2 support the claim that joint learning over utterance tokens and matched schema elements achieves this objective. For completeness, we show that the zero-shot ability does not simply come from the conjunction of constrained decoding and BART's extensive pre-training: we perform an ablation exercise where the input to BART-Large contains no schema information at all, but constrained decoding is enabled. As can be seen in the second row of Table 2, accuracy drops precipitously, by 46 and 23 absolute points for BURGER and COFFEE, respectively. A manual analysis of the predictions shows that in the overwhelming majority of cases, this model only generates intents and slots that it has seen before in training and thus fails to correctly parse utterances that have unseen intents/slots. On a subset of 108 BURGER utterances with at least one intent unseen in training, the schema-oblivious approach only gets 4% unordered EM, compared to 64% for Cross-TOP.

Schema-aware decoding ensures proper executable parses. Schema-aware constrained decoding ensures that Cross-TOP generates fully executable parse trees. Without this component, performance drops by 20 absolute points, as shown in the third row of Table 2. By looking at 15 predicted ut-

[9]Note that we do keep those slot/slot-value pairs for quantities larger than 1.

terances where the output predictions change by removing constrained decoding, we found that 93.3% of BURGER utterances and 60% of COFFEE utterances contained at least one invalid slot/slot-value combination. While using constrained decoding on these utterances is guaranteed to rule out invalid combinations, this does not ensure that the result will be correct. However, on inspection we found that constrained decoding transforms 60% of BURGER and 33.3% of COFFEE mismatched utterances to be completely correct. Table 6 in Appendix D illustrates how constrained decoding can help with specific examples.

Cross-TOP improves as more training tasks are added. While our main result shows that training with only few tasks allows zero-shot transfer to new tasks with no retraining, a realistic production scenario would be to periodically retrain the model by incorporating new training data. To quantify the benefits of adding more tasks, we compare training Cross-TOP using one, two or three tasks. The results for training on one task are an average over three models, one trained on PIZZA only, one on BURRITO and one on SUB. Likewise, results for training on two tasks are an average of three models, one trained on PIZZA+BURRITO, one on BURRITO+SUB and the other on PIZZA+SUB. As shown in Table 2, going from 1 to 2 tasks doubles performance for BURGER, and using 3 tasks almost triples the performance for both test tasks, confirming that the model learns general patterns that govern all schemas in the food-ordering vertical.

Dependency on fuzzy matching Cross-TOP relies on the quality of the fuzzy-matching process that determines which schema elements are encoded along with the utterance tokens. It can be challenging to recover from a fuzzy matching failure that ends up omitting a slot value from the input. In BURGER, such failures account for only 1% of all test utterances. In COFFEE that phenomenon is more prominent, with 7% of test utterances presenting at least one missing element from the fuzzy-matched schema. These limitations can be addressed by making the fuzzy-matching algorithm more robust and/or by adding unrecognized slot values as extra entries in the slot's catalog. The latter option is appealing, as it involves no model retraining, but does not suffice, as there is no obvious way to automate it. We upper-bound the impact of any candidate fix by providing an oracle schema for all utterances, and observe in the last row of Table 2 that it brings an absolute improvement of 2 absolute point in BURGER and 5 absolute points in COFFEE.

	Burger	Coffee
Cross-TOP	73.3 ± 3.6	54.8 ± 5.7
w/o schema-augmented input	26.5 ± 1.5	31.7 ± 1.0
w/o constrained decoding	53.0 ± 4.2	33.3 ± 7.2
training only on 1 task	25.4 ± 1.6	19.9 ± 3.3
training only on 2 tasks	52.4 ± 2.7	34.0 ± 3.5
w/ oracle schema	75.6 ± 4.3	59.4 ± 7.1

Table 2: Cross-TOP zero-shot unordered EM accuracy, averaged over 3 seeds, along with various ablations. The ± signs indicate the standard error across seeds.

6 Related Work

Slot Filling Traditionally, task-oriented parsing for flat intents and slots has been framed as a combination of intent classification and slot labeling (Sarikaya et al., 2016), possibly with an additional domain classification component. Several authors have addressed zero-shot solutions in this field. QASF (Du et al., 2021) is a QA-driven approach that extracts slot-filler spans from utterances using a question-answering model. Both Bapna et al. (2017) and Siddique et al. (2021) tag words with slots using slot descriptions and context-aware representations of the utterance. These solutions don't apply to structured (compositional) semantic representations, or to multiple intents in a single utterance, both of which are handled by Cross-TOP.

Task-Oriented Parsing In the more general area of task-oriented parsing, where hierarchical representations are featured, the authors are not aware of other zero-shot cross-schema work. There is some work in the few-shot setting (Chen et al., 2020), where data from multiple domains is used during an additional stage of fine-tuning combined with meta-learning.

Text-to-SQL Some of the most relevant related zero-shot work is in text-to-SQL semantic parsing. In this area, a challenging dataset, SPIDER (Yu et al., 2018), is the most common dataset used to test zero-shot solutions. The GAZP (Zhong et al., 2020) method generates synthetic training data for the new schema environment and requires a retraining of the neural parser, not making it as convenient of a zero-shot method. RAT-SQL (Wang et al., 2020) moves away from needing to retrain the parser, and focuses on jointly encoding the schema and utterance tokens. BRIDGE (Lin et al., 2020) is the main inspiration for our work, as it encodes the utterance and schema together, and augments the input with anchor texts, which are database values from tables, designed to better bridge utterance tokens to database tables, columns and values. Another notable contribution intended to bridge the gap between natural language and machine-executable representations is the work of Gan et al. (2021), which leverages an intermediate representation to go from text to SQL.

7 Conclusion

We presented Cross-TOP, a zero-shot method for cross-schema task-oriented parsing that eliminates the need to retrain and maintain a new model for every new task in a business vertical. We released a new dataset illustrative of five real-world applications in the food-ordering vertical. We showed that Cross-TOP reaches up to 73% EM accuracy in zero-shot transfer, making it a viable technique for quickly bootstrapping a parser for a new task.

Future work could further enrich the joint encoding of utterances and task schemas, while an additional thread of work could study how to best leverage limited annotated data that may be available for a new task.

Acknowledgments

We would like to thank Beiye Liu, Emre Barut, Ryan Gabbard, and anonymous reviewers for providing valuable feedback on this work.

References

Armen Aghajanyan, Jean Maillard, Akshat Shrivastava, Keith Diedrick, Michael Haeger, Haoran Li, Yashar Mehdad, Veselin Stoyanov, Anuj Kumar, Mike Lewis, and Sonal Gupta. 2020. Conversational semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5026–5035, Online. Association for Computational Linguistics.

Konstantine Arkoudas, Nicolas Guenon des Mesnards, Melanie Rubino, Sandesh Swamy, Saarthak Khanna, and Weiqi Sun. 2021. Pizza: a task-oriented semantic parsing dataset.

Ankur Bapna, Gokhan Tur, Dilek Hakkani-Tur, and Larry Heck. 2017. Towards zero-shot frame semantic parsing for domain scaling.

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.

Xinya Du, Luheng He, Qi Li, Dian Yu, Panupong Pasupat, and Yuan Zhang. 2021. QA-driven zero-shot slot filling with weak supervision pretraining. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 654–664, Online. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R Woodward, John Drake, and Qiaofu Zhang. 2021. Natural sql: Making sql easier to infer from natural language specifications. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2030–2042.

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.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888, Online. Association for Computational Linguistics.

Elman Mansimov and Yi Zhang. 2021. Semantic parsing in task-oriented dialog with recursive insertion-based encoder. *arXiv preprint arXiv:2109.04500*.

Subendhu Rongali, Luca Soldaini, Emilio Monti, and Wael Hamza. 2020. Don't parse, generate! a sequence to sequence architecture for task-oriented semantic parsing. In *Proceedings of The Web Conference 2020*, pages 2962–2968.

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, Online. Association for Computational Linguistics.

R. Sarikaya, P. A. Crook, A. Marin, M. Jeong, J.P. Robichaud, A. Celikyilmaz, Y.B. Kim, A. Rochette, O. Z. Khan, X. Liu, D. Boies, T. Anastasakos, Z. Feizollahi, N. Ramesh, H. Suzuki, R. Holenstein, E. Krawczyk, and V. Radostev. 2016. An overview of end-to-end language understanding and dialog management for personal digital assistants. In *2016 IEEE Spoken Language Technology Workshop (SLT)*, pages 391–397.

AB Siddique, Fuad Jamour, and Vagelis Hristidis. 2021. Linguistically-enriched and context-awarezero-shot slot filling. In *Proceedings of the Web Conference 2021*, pages 3279–3290.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R. Radev, Richard Socher, and Caiming Xiong. 2020. Grappa: Grammar-augmented pre-training for table semantic parsing. *CoRR*, abs/2009.13845.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6869–6882, Online. Association for Computational Linguistics.

Appendix

A Mechanical Turk Instructions

The instructions given to the workers were templated as shown in Figure 3. The tasks can be described as natural language text generation with a constrained menu. The number of responses was limited to 3 submissions per worker in order to balance diversity of responses and responsiveness ratios. The respondent's location had to be either US or CA, and the *master* worker qualification was required.[10] The tasks were designed, timed and priced to ensure that the compensation of respondents lies above the US and CA minimum hourly wages. The dataset went through an internal review process to ensure it abides by the company's required standards. Overall we collected answers form about 60 distinct workers for BURGER, SUB and COFFEE and about 90 for BURRITO, for a total of 183 unique individuals. The menus used for each collection are given in Figure 4.

B Task Schemas

Detailed schemas for each task, describing intent names, slot names, and slot properties, are given as supplementary material, along with the full catalog values for each slot.[11] For illustration purposes, Figure 5 shows the schema for BURRITO. Note that not all schemas need to share identical slots for similar intents. For example the BURGER schema has a `SIZE` slot for the `DRINK_ORDER` and `SIDE_ORDER` intents, while the BURRITO task does not. This is a design choice meant to reflect a real-life setting where the back end for one restaurant might support such property while another might not. This is a challenging—though realistic—obstacle our model needs to overcome.

C Dataset Construction Details

Part of the dataset comes from the publicly available PIZZA dataset (Arkoudas et al., 2021). We are following the conditions of use as defined by the license[12] and will release our dataset under the same license. The collection of new data was done through Mechanical Turk. Respondents were constrained to submit a single utterance for an order containing potentially more than one sub-order. Hence, some utterances contained periods and question marks, indicating a sharp separation between two requests. To better reflect the fact that these users would likely have broken their request into multiple ones in an vocal interaction, we split those utterances into pieces. Other punctuation marks like commas, and non-ASCII characters, were simply removed, but utterances were not split around them. Numerical values were spelled out (e.g., *2 large cokes* $\rightarrow$ *two large cokes*). Finally, utterance text was lower-cased. Annotation was carried out internally by two annotators located in the US. Utterances displaying too much ambiguity for human annotators were removed. In Table 3 we provide examples of the collected utterances, and their linearized semantics. As can be seen in the table, utterances have varying degrees of complexity, which results in linearized trees of varying depths and widths. Synthetic data was generated by sampling human-designed templates, illustrated in Table 4. For SUB we used 32 templates and for BURRITO we used 46. Some statistics on the degree of compositionality of human and synthetic orders are given in Table 5.

D Constrained Decoding Details

In what follows we list the actual constraints implemented in this work in the form of allowed transitions. Any element on the left of the arrow can be followed by elements on the right:

$$\begin{cases} \texttt{BOS} & \rightarrow & \{\text{"}(X\text{"}, X = \text{valid intent}\,\} \\ \text{"}(\text{"} & \rightarrow & \{\, X, X = \text{valid intent or slot}\} \\ \text{"})\text{"} & \rightarrow & \{\text{"})\text{" or "}(\text{" or } \texttt{EOS}\,\} \\ \text{intent} & \rightarrow & \{\text{"}(X\text{"}, X = \text{a valid slot}\,\} \\ \text{slot} & \rightarrow & \{X, X = \text{compatible value}\,\} \\ \texttt{(COMPLEX} & \rightarrow & \texttt{(QUANTITY} \end{cases}$$

One could think of imposing more grammar-based constraints, for example, allowing only valid intent-slot combinations, or only allowing negatable slots after `(NOT`, since some of these—like `SIZE`—cannot be negated. Examples of how constrained decoding helped can be found in Table 6.

E Computational Details

Hyperparameter tuning was performed on learning rates [`5e-04`, `1e-05`, `5e-05`, `1e-06`] and batch sizes [`16, 24, 48, 64`] across three seeds.

[10]workers with high ratings according to MTurk API.

[11]https://github.com/amazon-research/food-ordering-semantic-parsing-dataset

[12]https://github.com/amazon-research/pizza-semantic-parsing-dataset/blob/main/LICENSE

Dataset	Natural Language	Semantic representation after entity resolution
PIZZA	five medium pizzas with tomatoes and ham	`(PIZZAORDER (NUMBER 5 ) (SIZE medium ) (TOPPING ham ) (TOPPING tomatoes ))`
PIZZA	i'll have one pie along with pesto and ham but avoid olives	`(PIZZAORDER (NOT (TOPPING olives ) ) (NUMBER 1 ) (TOPPING ham ) (TOPPING pesto ))`
PIZZA	i wanted to have two dr peppers three pepsis and a sprite	`(DRINKORDER (DRINKTYPE dr_pepper ) (NUMBER 2 )) (DRINKORDER (DRINKTYPE pepsi ) (NUMBER 3 )) (DRINKORDER (DRINKTYPE sprite ) (NUMBER 1 ))`
BURRITO	burrito with steak cheese guacamole sour cream and fresh tomato salsa	`(BURRITO_ORDER (NUMBER 1 ) (MAIN_FILLING steak ) (TOPPING cheese ) (TOPPING guacamole ) (TOPPING sour_cream ) (SALSA_TOPPING fresh_tomato_salsa ) )`
BURRITO	i'd also like a bottled water please	`(DRINK_ORDER (NUMBER 1 ) (DRINK_TYPE bottled_water ))`
BURRITO	i'd like a lemonade with a side of chips	`(DRINK_ORDER (NUMBER 1 ) (DRINK_TYPE tractor_lemonade ) (SIDE_ORDER (NUMBER 1 ) (SIDE_TYPE chips ))`
SUB	steak and cheese sandwich with lettuce cucumbers and olives	`(SANDWICH_ORDER (NUMBER 1 ) (BASE_SANDWICH steak_and_cheese ) (TOPPING lettuce ) (TOPPING cucumbers ) (TOPPING black_olives ) )`
SUB	i will order a chicken and bacon ranch sandwich and on that please put american cheese chipotle southwest sauce lettuce tomatoes pickles with a side of doritos and two chocolate chip cookies	`(SANDWICH_ORDER (NUMBER 1 ) (BASE_SANDWICH chicken_and_bacon_ranch ) (TOPPING american_cheese ) (TOPPING chipotle_southwest ) (TOPPING lettuce ) (TOPPING tomatoes ) (TOPPING pickles ) ) (SIDE_ORDER (NUMBER 1 ) (SIDE_TYPE doritos_nacho_cheese ) ) (SIDE_ORDER (NUMBER 2 ) (SIDE_TYPE chocolate_chip ) )`
BURGER	hi can i have the double cheeseburger with ketchup and onions and french fries on the side	`(MAIN_DISH_ORDER (NUMBER 1 ) (MAIN_DISH_TYPE double_cheese_burger ) (TOPPING ketchup ) (TOPPING onion ) ) (SIDE_ORDER (NUMBER 1 ) (SIDE_TYPE french_fries ) )`
BURGER	veggie burger with lettuce and bacon large curly fry and a small iced tea	`(MAIN_DISH_ORDER (NUMBER 1 ) (MAIN_DISH_TYPE vegan_burger ) (TOPPING lettuce ) (TOPPING bacon ) ) (SIDE_ORDER (NUMBER 1 ) (SIZE large ) (SIDE_TYPE curly_fries ) ) (DRINK_ORDER (NUMBER 1 ) (SIZE small ) (DRINK_TYPE iced_tea ) )`
COFFEE	i'd like a large hot chocolate with whipped cream	`(DRINK_ORDER (NUMBER 1 ) (SIZE large ) (DRINK_TYPE hot_chocolate ) (TOPPING whipped_cream ) )`
COFFEE	one regular latte light roast with an extra espresso shot and honey added and one large cappuccino with caramel syrup in that one	`(DRINK_ORDER (NUMBER 1 ) (SIZE regular ) (DRINK_TYPE latte ) (ROAST_TYPE light_roast ) (TOPPING honey ) (TOPPING (ESPRESSO_SHOT 1 ) ) ) (DRINK_ORDER (NUMBER 1 ) (SIZE large ) (DRINK_TYPE cappuccino ) (TOPPING caramel_syrup ) )`

Table 3: Example utterances obtained from Mechanical Turk collection and their corresponding machine-executable representation.

Dataset	Template	Example catalog values
SUB	`{prelude} {number} {side_type}`	`{prelude}` = *i want to order* `{side_type}` = *sunchips*
SUB	`{prelude} {number} {base_sandwich} with {topping1} and {topping2}`	`{base_sandwich}` = *chicken teriyaki* `{topping1}` = *bacon*
BURRITO	`{prelude} {number} {main_filling} {entity_name} with {salsa_topping}`	`{main_filling}` = *barbacoa* `{entity_name}` = *burrito*
BURRITO	`{prelude} {number} side of {side_type1} and {side_type2} and {number} {drink_type}`	`{side_type1}` = *chips* `{side_type2}` = *guac*

Table 4: Example templates and catalog values used for sampling synthetic data.

We use the human-generated data of the three training tasks as our development set for early stopping and hyperparameter tuning. Including this and general experimentation, we estimate our total

	#Intent per utterance	#Slots per utterance	Avg utterance depth
Synthetic Data			
PIZZA	1.77	5.77	3.44
BURRITO	1.57	6.50	3.48
SUB	1.79	6.24	3.37
Human-generated Data			
PIZZA	1.25	6.13	3.62
BURRITO	1.39	5.78	3.12
SUB	1.69	5.99	3.07
BURGER	1.97	7.17	3.04
COFFEE	1.05	5.34	3.2

Table 5: Statistics on the degree of compositionality in each task, for synthetic and human-generated data.

Dataset	Natural Language Utterance	Prediction w/o constraints	Prediction w/ constraints
BURGER	i'll have a hamburger topped with bacon and ketchup along with a large coke and large order of french fries	`(MAIN_DISH_ORDER (MAIN_DISH_TYPE hamburger ) (TOPPING bacon ) (TOPPING ketchup )) (DRINK_ORDER (SIZE large ) (DRINK_TYPE coke )) (SIDE_ORDER (NUMBER large ) (SIDE_TYPE french fries ))`	`(MAIN_DISH_ORDER (MAIN_DISH_TYPE hamburger ) (TOPPING bacon ) (TOPPING ketchup )) (DRINK_ORDER (SIZE large ) (DRINK_TYPE coke )) (SIDE_ORDER (NUMBER a ) (SIZE large ) (SIDE_TYPE french fries ))`
COFFEE	i'd like an iced cappuccino with caramel syrup and whipped cream	`(DRINK_ORDER (STYLE iced cappuccino ) (TOPPING caramel syrup ) (TOPPING whipped cream ))`	`(DRINK_ORDER (STYLE iced ) (DRINK_TYPE cappuccino ) (TOPPING caramel syrup ) (TOPPING whipped cream ))`

Table 6: Example utterances where constrained decoding helps fix invalid slot/slot value combinations.

computation cost to be about 2 weeks GPU hours.

MTurk prompt

Suppose you want to place your usual order at your favorite *type of restaurant* (like *examples of such venues*) for you, your partner, your family or your group of friends. Your task is to enter your order exactly as you would say it, verbatim, when you place the order at that restaurant.

IMPORTANT: This restaurant has a limited menu provided below. Only order items available on the menu, but do so with the same words you usually use when ordering these items:

**** Picture of restaurant Menu ****

Write as you would speak. Make sure that:

- you write your order exactly as you would say it
- your usual order may include many items and if so, include them all when you enter your order below
- if you complete multiple HITs, vary the type of orders you place. The orders should be usual orders you, your friends or family place, but with varying number or types of items, toppings, sides or drinks.

Enter your order below, using the limited menu above, exactly as you would say it at the restaurant :

**** Type order here ****

Figure 3: Template prompt given to Mechanical Turk workers, common across all 4 tasks. The only significant attribute varying across tasks was the menu to order from.

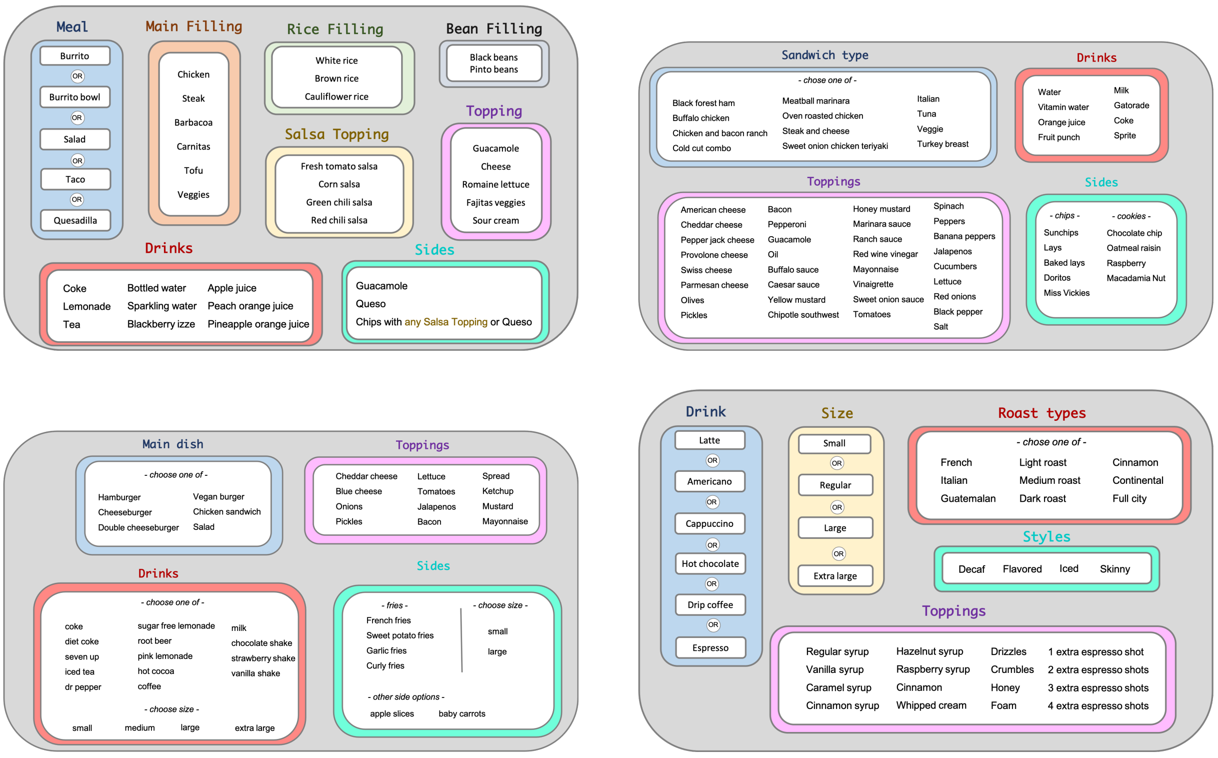


Figure 4: Menus shown to Mechanical Turk workers for each task: BURRITO (top-left), SUB (top-right), BURGER (bottom-left) and COFFEE (bottom-right).

```
{
  "name": "BURRITO",
  "intents":  [
    {"name": "BURRITO_ORDER",
     "invocation_keywords": ["burrito", "burritos"],
      "slots": [
      {"name": "NUMBER"}, {"name": "MAIN_FILLING", "quantifiable":  true},
      {"name": "RICE_FILLING", "negatable": true, "quantifiable":  true},
      {"name": "BEAN_FILLING", "negatable": true, "quantifiable":  true},
      {"name": "SALSA_TOPPING", "negatable": true, "quantifiable":  true},
      {"name": "TOPPING", "negatable": true, "quantifiable":  true}
      ]
    },
    {"name": "BURRITO_BOWL_ORDER",
      "invocation_keywords": ["burrito bowl", "burrito bowls", "bowl", "bowls"],
      "slots": [
      {"name": "NUMBER"}, {"name": "MAIN_FILLING", "quantifiable":  true},
      {"name": "RICE_FILLING", "negatable": true, "quantifiable":  true},
      {"name": "BEAN_FILLING", "negatable": true, "quantifiable":  true},
      {"name": "SALSA_TOPPING", "negatable": true, "quantifiable":  true},
      {"name": "TOPPING", "negatable": true, "quantifiable":  true}
      ]
    },
    {"name": "SALAD_ORDER",
     "invocation_keywords": ["salad", "salads"],
      "slots": [
      {"name": "NUMBER"}, {"name": "MAIN_FILLING", "quantifiable":  true},
      {"name": "RICE_FILLING", "negatable": true, "quantifiable":  true},
      {"name": "BEAN_FILLING", "negatable": true, "quantifiable":  true},
      {"name": "SALSA_TOPPING", "negatable": true, "quantifiable":  true},
      {"name": "TOPPING", "negatable": true, "quantifiable":  true}
      ]
    },
    {"name": "TACO_ORDER",
     "invocation_keywords": ["taco", "tacos"],
      "slots": [
      {"name": "NUMBER"}, {"name": "MAIN_FILLING", "quantifiable":  true},
      {"name": "RICE_FILLING", "negatable": true, "quantifiable":  true},
      {"name": "BEAN_FILLING", "negatable": true, "quantifiable":  true},
      {"name": "SALSA_TOPPING", "negatable": true, "quantifiable":  true},
      {"name": "TOPPING", "negatable": true, "quantifiable":  true}
      ]
    },
    {"name": "QUESADILLA_ORDER",
     "invocation_keywords": ["quesadilla", "quesadillas"],
      "slots": [
      {"name": "NUMBER"}, {"name": "MAIN_FILLING", "quantifiable":  true},
      {"name": "RICE_FILLING", "negatable": true, "quantifiable":  true},
      {"name": "BEAN_FILLING", "negatable": true, "quantifiable":  true},
      {"name": "SALSA_TOPPING", "negatable": true, "quantifiable":  true},
      {"name": "TOPPING", "negatable": true, "quantifiable":  true}
      ]
    },
    {"name": "SIDE_ORDER",
    "invocation_keywords": ["side of chip", "sides of chips"],
      "slots": [
        {"name": "NUMBER"},
        {"name": "SIDE_TYPE"},
        {"name": "SALSA_TOPPING", "negatable": true, "quantifiable":  true}
      ]
    },
    {"name": "DRINK_ORDER",
     "invocation_keywords": ["drink", "drinks"],
      "slots": [
        {"name": "NUMBER"},
        {"name": "DRINK_TYPE"}
      ]
    }
  ]
}
```

Figure 5: Task schema for the BURRITO restaurant.