Graph-Based Decoding for Task Oriented Semantic Parsing

Jeremy R. Cole[†] **Nanjiang Jiang**[‡*] **Panupong Pasupat**[†] **Luheng He**[†] **Peter Shaw**[†]

[†]Google Research
{jrcole,ppasupat,luheng,petershaw}@google.com
[‡]The Ohio State University
jiang.1879@osu.edu

Abstract

The dominant paradigm for semantic parsing in recent years is to formulate parsing as a sequence-to-sequence task, generating predictions with auto-regressive sequence decoders. In this work, we explore an alternative paradigm. We formulate semantic parsing as a dependency parsing task, applying graph-based decoding techniques developed for syntactic parsing. We compare various decoding techniques given the same pre-trained Transformer encoder on the TOP dataset, including settings where training data is limited or contains only partially-annotated examples. We find that our graph-based approach is competitive with sequence decoders on the standard setting, and offers significant improvements in data efficiency and settings where partially-annotated data is available.

1 Introduction

Semantic parsing, the task of mapping natural language queries to structured meaning representations, remains an important challenge for applications such as dialog systems. To support compositional utterances in a task oriented dialog setting, Gupta et al. (2018) introduced the Task Oriented Parse (TOP) representation and released a dataset consisting of pairs of natural language queries and associated TOP trees. As illustrated in Figure 1, TOP trees are hierarchically structured representations consisting of intents, slots, and query tokens.

We propose a novel formulation of semantic parsing for TOP as a graph-based parsing task, presenting a graph-based parsing model (hereafter, GBP). Our approach is motivated by the success of such approaches in dependency parsing (McDonald et al., 2005; Kiperwasser and Goldberg, 2016; Dozat and Manning, 2017; Kulmizev et al., 2019) and AMR parsing (Zhang et al., 2019). Recently, sequence-to-sequence (seq2seq) models have become a dominant approach to semantic parsing

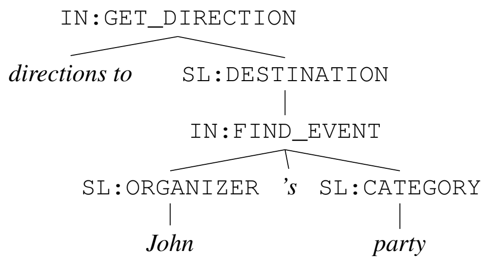

Figure 1: An example TOP (Gupta et al., 2018) tree.

(e.g., Dong and Lapata 2016, Jia and Liang 2016, Wang et al. 2019a), including on TOP (e.g., Rongali et al. 2020; Aghajanyan et al. 2020; Shao et al. 2020). Unlike such approaches that predict outputs auto-regressively, GBP decomposes parse tree scores over parent-child edge scores, predicting all edge scores in parallel.

First, we compare GBP with seq2seq and other decoding techniques, within the context of a fixed encoder and pretraining scheme: in this case, BERT-Base (Devlin et al., 2019). This allows us to isolate the role of the decoding method. We compare these models across the standard setting, as well as additional settings where training data is limited, or when fully annotated examples are limited but partially annotated examples are available. We find that GBP outperforms other methods, especially when learning from partial supervision. Second, we compare GBP with seq2seq models that additionally leverage pretrained decoders. We find that GBP remains competitive, and continues to outperform in the partial supervision setting.

2 Task Formulation

We present a novel formulation of the TOP semantic parsing task as a graph-based parsing task. Our goal is to predict a TOP tree $\mathbf{y}$ given a natural language query $\mathbf{x}$ as input. The nodes in $\mathbf{y}$ consist of intent and slot symbols from a vocabulary of output symbols $\mathcal{V}$ and the tokens in $\mathbf{x}$. However, $\mathbf{y}$ cannot be predicted directly by a conventional

[*]Work done while on internship at Google.

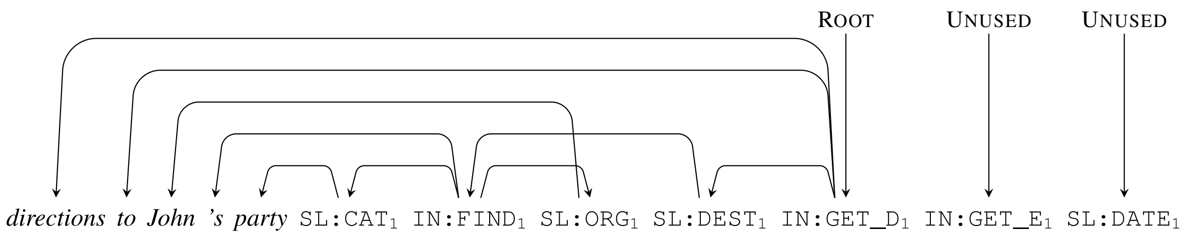
ROOT
UNUSED
UNUSED
directions to John 's party SL:CAT$_1$ IN:FIND$_1$ SL:ORG$_1$ SL:DEST$_1$ IN:GET_D$_1$ IN:GET_E$_1$ SL:DATE$_1$

Figure 2: The graph-based model predicts parent assignments across a set of nodes consisting of query tokens, output symbols for intents and slots, and special UNUSED and ROOT symbols. This is the corresponding parse tree for the TOP tree shown in Figure 1. Not all output symbols are drawn; omitted symbols are attached to UNUSED. Intent and slot names are abbreviated.

graph-based approach (McDonald et al., 2005) for two reasons. First, given $\mathbf{x}$, we do not know the subset of intent and slot[1] symbols that occur in $\mathbf{y}$. Second, intent and slot symbols can occur more than once in $\mathbf{y}$.[2]

To address this, let us consider a parse tree $\mathbf{z}$ in a space of valid trees defined as $\mathcal{Z}(\mathbf{x})$. The parse tree $\mathbf{z}$ can be deterministically mapped to and from $\mathbf{y}$. The parse tree $\mathbf{z}$ consists of: (1) the tokens in $\mathbf{x}$, (2) every symbol in $\mathcal{V}$ replicated up to a maximum number of occurrences[3] and assigned a corresponding index, and (3) a special UNUSED node in addition to the standard ROOT node. Let $\mathcal{N}(\mathbf{x})$ be this set of nodes which all trees in $\mathcal{Z}(\mathbf{x})$ consist of. When mapping from $\mathbf{y}$ to $\mathbf{z}$, output symbols occurring multiple times are indexed following a pre-order traversal, and any output symbol that does not occur in $\mathbf{y}$ is assigned to the UNUSED node in $\mathbf{z}$. For example, Figure 1 illustrates an example TOP tree, $\mathbf{y}$, and Figure 2 illustrates a corresponding parse tree, $\mathbf{z}$.

3 Scoring Model

Given that the mapping between $\mathbf{y}$ and $\mathbf{z}$ is deterministic, our goal is to model $p(\mathbf{z} \mid \mathbf{x})$. We follow a conventional edge-factored graph-based approach (McDonald et al., 2005), decomposing parse tree scores over directed edges between parent and child node pairs (p, c) in $\mathbf{z}$:

$$p(\mathbf{z} \mid \mathbf{x}) = \prod_{(p,c)\in\mathbf{z}} \frac{\exp(\phi(p, c, \mathbf{x}))}{\sum_{p'\in\mathcal{N}(\mathbf{x})} \exp(\phi(p', c, \mathbf{x}))},$$

where edge scores, $\phi(p, c, \mathbf{x})$, are computed similarly to the model of Dozat and Manning (2017):

$$\phi(p, c, \mathbf{x}) = (e^p_{\mathbf{x}})^\intercal U e^c_{\mathbf{x}} + (e^p_{\mathbf{x}})^\intercal u,$$

where $e^p_{\mathbf{x}}$ and $e^c_{\mathbf{x}}$ are contextualized vector representations of the nodes p and c, respectively, and U and u are a parameter matrix and vector, respectively.[4]

Node representations are computed differently for each node type in $\mathcal{N}(\mathbf{x})$. Encodings for token nodes are based on the output of a BERT (Devlin et al., 2019) encoder; replicated output symbols are embedded based on their symbol and index; ROOT and UNUSED nodes likewise have a unique embedding. All nodes are then jointly encoded with a Transformer (Vaswani et al., 2017) encoder, which produces the contextualized node representations ${e_x}^p$ and ${e_x}^c$ which are used in the above equations to produce the factored edge scores.

The scoring model is trained using a standard maximum likelihood objective.

4 Parsing Algorithm

Chu-Liu-Edmonds Algorithm

The Chu-Liu-Edmonds (CLE) algorithm is an optimal algorithm to find a maximum spanning arborescence over a directed graph (Chu and Liu, 1965; Edmonds, 1965). It has commonly been used for parsing dependency trees from edge-factored scoring models (e.g., McDonald et al. 2005; Dozat and Manning 2017). Note that in an arborescence (hereafter tree), each node can have at most one 'parent', or incoming edge. Thus, the algorithm first chooses the highest scoring parent for each node as the initial *best parent*. It is possible these initial best parents already form a tree; however, it

[1]Note that one could imagine instead treating slots as edge labels instead of nodes, but as the set is large (36 slots for 25 intents), little advantage would be expected.

[2]See Figure 5 in Appendix for an example.

[3]The number of repetitions per output symbol is determined from the training data. If a symbol has a maximum of k occurrences in a TOP tree in the training data, it will have $k + 2$ replications. See Appendix C for more information.

[4]For computational efficiency and to prevent invalid trees, we consider the score for assigning a token node as a parent for any other node to be a fixed value of $-\infty$.

may instead produce a graph with cycles. In that case, CLE recursively breaks the cycles until the optimal tree is found. Note that CLE takes the index of the root of the tree as an input, and begins by deleting all incoming edges to enforce this constraint. Conventionally, in dependency parsing, the root of the tree is the special ROOT node.

This algorithm is optimal for dependency parsing; however, our formulation differs due to additional constraints based on how TOP trees are mapped to and from dependency trees. First, by convention, the parent of the UNUSED subtree must be ROOT. Second, the UNUSED subtree must be of depth 2: it cannot have any grandchildren. Finally, as valid TOP trees have only one root, the ROOT node must have only one 'child', or outgoing edge.

Unused Node Preprocessing

As stated, our UNUSED subtree must only have depth 2 to follow our task formulation. Otherwise, the final tree score will be computed incorrectly when translating to a TOP tree, as the entire UNUSED subtree is effectively discarded. Thus, we first preprocess the UNUSED subtree to ensure depth 2. In practice, simply using the initial best parents will result in UNUSED subtrees with depth 3 or greater about 1% of the time.

We resolve such cases by making a decision for each node a whose initial best parent is UNUSED and has children itself. One option is to delete the edge to a from UNUSED, making the next highest scoring edge the new best parent of a. The cost of this action is equal to the difference in scores between the corresponding edges. Alternatively, we can take a similar action on each child of a: delete the edge from a, making the next highest scoring edge the new best parent. The cost of this action is equal to the difference in the corresponding edges summed over every child of a. We iterate over the children of UNUSED that have children, selecting the action with the lower cost, until the constraint is met. Then, we no longer allow further modifications to the UNUSED subtree, effectively deleting it for the remaining stages of the algorithm.

Note that this algorithm is not necessarily optimal: the order in which we consider the children of UNUSED can affect the final result. However, we find this approximation to work well in practice.

Multiple Root Resolution

Our second modification to the CLE algorithm concerns the ROOT node. Valid TOP trees are single-rooted: in our formalism, this means the ROOT node can only have a single child. To enforce this constraint, we want to choose the single child of ROOT that results in the highest scoring tree. We then provide this child's index to the CLE subroutine and delete all edges from ROOT, effectively discarding it. To find the best root, we start with the set of nodes whose initial best parent is the ROOT node. If this set is a singleton, we simply choose that node as the tree's root, providing its index to the CLE subroutine. In about 0.5% of trees, there is more than one node. In that case, we run the CLE algorithm with each node as the given root index, taking the highest-scoring tree. This is still not guaranteed to be optimal: the optimal choice of the root node could have a different initial best parent than ROOT. However, this was not observed in our experiments and trying every node drastically increases the computation.

5 Experiments

The TOP dataset consists of trees where every token in the query is attached to either an intent (prefixed with IN:) or slot label (prefixed with SL:). Intents and slot labels can also attach to each other, forming compositional interpretations. We evaluate several models on the standard setup of the TOP dataset. We also devise new setups comparing the abilities of several models to learn from a smaller amount of fully annotated data, both with and without additional partially annotated data. Models are compared on exact match accuracy. Following Rongali et al. (2020); Einolghozati et al. (2018), and Aghajanyan et al. (2020), we filter out queries annotated as unsupported[5], leaving 28414 *train* examples and 8241 *test* examples.

5.1 Standard Supervision

We use *standard supervision* to refer to settings where all training examples contain a complete output tree. We also evaluate *data efficiency*, by comparing the performance when training data is limited to 1% or 10% of the original dataset.

5.2 Partial Supervision

We use *partial supervision* to refer to settings where we discard labels for certain nodes in the output trees of some or all training examples. Such partially annotated examples could arise in practice; for instance, when there is annotator disagree-

[5]We include results on the full set in Appendix D.

	Standard Supervision			Partial Supervision				
Decoder (BERT-Base encoder)	100	10	1	10/90/0	10/0/90	1/99/0	1/0/99	2/49/49
PtrGen (Rongali et al., 2020)	83.13	—	—	—	—	—	—	—
PtrGen (our implementation)	85.00	76.84	51.85	13.24	26.68	4.39	0.00	4.43
FSP (Pasupat et al., 2019)	85.12	**79.44**	**57.95**	68.94	—	42.94	—	—
GBP (proposed)	**86.14**	79.43	56.89	**84.55**	**84.14**	**81.52**	**73.94**	**85.01**

Table 1: Results for various decoders with BERT-Base as encoder. For *standard supervision*, column headers denote the percentage of training data used. For *partial supervision*, column headers S/T/N denote the percentage of training examples with standard supervision (S), terminal-only supervision (T), and nonterminal-only supervision (N), respectively.

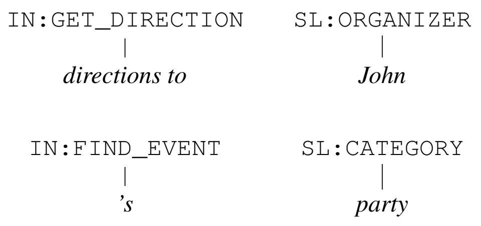

Figure 3: The TOP example from Figure 1 with terminal-only supervision.

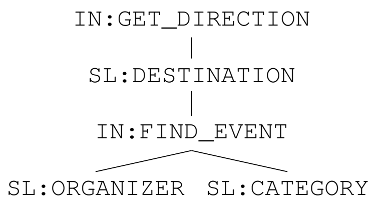

Figure 4: The TOP example from Figure 1 with nonterminal-only supervision.

ment on part of the output tree, or when changes to the set of possible slots or intents render parts of previously annotated trees obsolete.

As semantic parsing datasets normally require expert annotators, extending fully annotated examples with additional partial annotation can be an effective strategy. For instance, Choi et al. (2015) scaled their semantic parsing model with partial ontologies, and Das and Smith (2011) used additional semi-supervised data for their frame semantic parsing model. We consider two types of partially annotated output trees described below.

Terminal-only Supervision For this type of partial supervision, only the labels of each token (i.e., terminal) are preserved. See Figure 3 for an example. The label for each individual token is known, but the full set of intents and slots, and their tree structure, is unknown. This is similar to utilizing span labels that do not have full trees available.

Nonterminal-only Supervision For this type of partial supervision, token (i.e., terminal) labels are discarded. This is equivalent to deleting all of the token nodes from the tree. See Figure 4 for an example. This provides the opposite type of supervision as the terminal-only supervision case. The complete set of intents and slots and their tree structure is known, but their anchoring to the query text is unknown. For instance, if a query is known to have the same parse as a fully annotated query, its grounding may still be unknown.

5.3 Results

Comparisons with Fixed Encoder We first compare GBP with other methods using the same pre-trained encoder (BERT-Base; Devlin et al. 2019). We compare with a standard sequence decoder (a pointer-generator network; Vinyals et al. 2015; See et al. 2017) implemented using a Transformer-based (Vaswani et al., 2017) decoder (PtrGen). We report the previous results from Rongali et al. (2020) and new results from an implementation based on that of Suhr et al. (2020), which provides a slightly stronger baseline. We also compare with the factored span parsing (FSP) approach of Pasupat et al. (2019). Notably, we report new results for FSP using a BERT-Base encoder, which are significantly stronger than previously published results which used GloVe (Pennington et al., 2014) embeddings (85.1% vs.81.8%).

Results can be found in Table 1. We evaluate these models across both the standard and partial supervision settings. Notably, GBP can incorporate partial supervision in a straightforward way because scores for parse trees are factored over conditionally-independent scores for each edge. Training proceeds as described in Section 3; however, the loss from the edges that are not given by the example is masked. Additional training details can be found in Appendix B. For PtrGen, each type of partial supervision is given a task-specific prefix; details are in Appendix A. Similar to GBP, FSP factors parse scores across local components, but also considers chains of length > 1. Therefore, terminal-only supervision uses only length 1 chains; there is no trivial way to use nonterminal-only supervision without very substantial changes.

GBP is the highest-performing of the BERT-base models on the standard setup. Both GBP and FSP show better data efficiency than PtrGen. Only GBP appears to effectively benefit from partially annotated data in our experiments; the other models perform worse when incorporating this data.

Comparisons with Pretrained Decoders Recently, sequence-to-sequence models with pretrained decoders, such as BART (Lewis et al., 2019) and T5 (Raffel et al., 2020), have demonstrated strong performance on a variety of tasks. Careful comparisons isolating the effects of model size and pretraining tasks are limited by the availability of pretrained checkpoints for such models. Regardless, we compare GBP (with BERT-Base) directly with such models. On the standard setting for TOP, Aghajanyan et al. (2020) report SOTA performance with BART (87.1%), outperforming GBP. We also report new results comparing GBP with T5 on both the standard supervision and partial supervision settings in Table 2.

Notably, T5 is able to leverage partially-annotated examples much more effectively than PtrGen, which is also a Transformer-based sequence-to-sequence model but does not have a pretrained decoder. While T5 outperforms GBP on the standard setting, GBP outperforms T5 on the data efficiency and partial supervision settings.

	T5-Base		GBP (BERT-Base)	
S/T/N	Acc	Relative	Acc	Relative
100	**86.26**	—	86.14	—
10	77.70	—	**79.43**	—
10/90/0	83.44	7.39%	**84.55**	6.45%
10/0/90	81.63	5.06%	**84.14**	5.93%
1	48.96	—	**56.89**	—
1/99/0	74.31	51.78%	**81.52**	43.29%
1/0/99	56.22	14.83%	**73.94**	29.97%
2/49/49	82.00	—	**85.01**	—
0/50/50	—	—	**85.03**	—

Table 2: Comparison of T5 and GBP on data efficiency and partial supervision. *Relative* refers to the absolute increase in accuracy when incorporating partially-annotated examples compared to using only fully annotated data. S/T/N denotes the percentage of training examples with standard supervision (S), terminal-only supervision (T), and nonterminal-only supervision (N), respectively.

6 Related Work

The most recent state of the art on TOP has focused on applying new methods of pretraining; (Rongali et al. 2020; Shao et al. 2020; Aghajanyan et al. 2020) all use seq2seq methods, enhanced by better pretraining from BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), and BART (Lewis et al., 2020) while using similar model architectures. In this work, we instead investigate the choice of decoder. While the FSP model (Pasupat et al., 2019) similarly uses a factored approach, its approach is more specific to TOP, as its trees must be projective and anchored to the input text.

In dependency parsing, the performance of graph-based and transition-based parsing is compared in both Zhang and Clark (2008) and Kulmizev et al. (2019). Graph-based parsing has also been used in AMR parsing (Zhang et al., 2019), which translates sentences into structured graph representations. Similar methods have also been used in semantic role labeling (He et al., 2018), which requires labeling arcs between text spans. This work is the first to adapt graph-based parsing to tree-like task-oriented semantic parses.

7 Conclusions

We propose a novel framing of semantic parsing for TOP as a graph-based parsing task. We find that our proposed method is a competitive alternative to the standard paradigm of seq2seq models, especially when fully annotated data is limited and/or partially-annotated data is available.

Acknowledgements

We thank Timothy Dozat, Terry Koo, and Philip Massey for their feedback as we developed this project and for their comments on earlier versions of the manuscript. We also thank the broader Google Language Research organization for their support of the project. Finally, we thank the anonymous reviewers for their helpful suggestions.

Ethical Considerations

We fine-tune all models using 32 Cloud TPU v3 cores[6]. Additional training details are in Appendix A and Appendix B. We reused existing pretrained checkpoints for both BERT and T5, reducing the resources needed to run experiments. Our evaluation focuses on the existing TOP dataset: the details of the collection can be found in Gupta et al. (2018). TOP is an English-only dataset, which limits our ability to claim that our findings generalize across languages. A deployed dialog system has additional ethical considerations related to access, given their potential to make certain computational functions faster, easier, or more hands-free.

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.

Eunsol Choi, Tom Kwiatkowski, and Luke Zettlemoyer. 2015. Scalable semantic parsing with partial ontologies. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1311–1320, Beijing, China. Association for Computational Linguistics.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400.

Dipanjan Das and Noah A. Smith. 2011. Semi-supervised frame-semantic parsing for unknown predicates. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1435–1444, Portland, Oregon, USA. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

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

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California. Association for Computational Linguistics.

Jack Edmonds. 1965. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467.

Arash Einolghozati, Panupong Pasupat, Sonal Gupta, Rushin Shah, Mrinal Mohit, Mike Lewis, and Luke Zettlemoyer. 2018. Improving semantic parsing for task oriented dialog. In *Conversational AI Workshop at NeurIPS*.

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.

Luheng He, Kenton Lee, Omer Levy, and Luke Zettlemoyer. 2018. Jointly predicting predicates and arguments in neural semantic role labeling. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 364–369, Melbourne, Australia. Association for Computational Linguistics.

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.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Artur Kulmizev, Miryam de Lhoneux, Johannes Gontrum, Elena Fano, and Joakim Nivre. 2019. Deep contextualized word embeddings in transition-based and graph-based dependency parsing - a tale of two parsers revisited. In *Proceedings of the*

[6]https://cloud.google.com/tpu/

2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 2755–2768, Hong Kong, China. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pretraining 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.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98.

Panupong Pasupat, Sonal Gupta, Karishma Mandyam, Rushin Shah, Mike Lewis, and Luke Zettlemoyer. 2019. Span-based hierarchical semantic parsing for task-oriented dialog. 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 1520–1526, Hong Kong, China. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

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*, 21(140):1–67.

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.

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

Bo Shao, Yeyun Gong, Weizhen Qi, Guihong Cao, Jianshu Ji, and Xiaola Lin. 2020. Graph-based transformer with cross-candidate verification for semantic parsing. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8807–8814.

Peter Shaw, Philip Massey, Angelica Chen, Francesco Piccinno, and Yasemin Altun. 2019. Generating logical forms from graph representations of text and entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 95–106, Florence, Italy. Association for Computational Linguistics.

Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring unexplored generalization challenges for cross-database semantic parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8372–8388, Online. Association for Computational Linguistics.

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

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pages 2692–2700.

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

Yiren Wang, Fei Tian, Di He, Tao Qin, ChengXiang Zhai, and Tie-Yan Liu. 2019b. Non-autoregressive machine translation with auxiliary regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5377–5384.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019. AMR parsing as sequence-to-graph transduction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 80–94, Florence, Italy. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii. Association for Computational Linguistics.

A Baseline Model Training Details

A.1 T5 Training Details

We use the base version of the T5.1.1 model (220M parameters)[7] for finetuning with default learning rates. We also experimented with T5-large in preliminary experiments but did not observe visible difference in performances. The multitask experiments are finetuned for 5000 steps, and the rest are finetuned for 2000 steps, all with batch_size = 4096. We use greedy decoding at inference time. All experiments are ran once.

We use 32 Cloud TPU v3 cores for training and 8 TPU cores for inference. Training each model takes about 4 hours, and inference takes about 2 minutes on the entire TOP test set.

For the multitask experiments, we follow Raffel et al. (2020) by appending task prefixes to the input sequence. Specifically, we used "span:" for examples with terminal-only partial supervision, "ungrounded:" for examples with nonterminal-only partial supervision, and "full:" for examples with full supervision.

A.2 PtrGen Training Details

Following Suhr et al. (2020), we started with the hyperparameters of (Shaw et al., 2019). We then tuned the learning rate over 3 runs to be $1e^{-4}$. We use a BERT-Base encoder and the Transformer decoder consists of 4 layers with 8 attention heads, 64 dimensions, and 256 feed-forward hidden layer dimensions. The model is trained for $30K$ steps, with the pre-trained BERT parameters frozen for the first $4K$ steps. Task prefixes are prepended in the same manner as T5.

A.3 FSP Training Details

Hyperparameters were reused from Pasupat et al. (2019), except for the initial learning rate which changes to 0.00001 to make it more suitable for fine-tuning BERT. For partially supervised examples, we only define the loss on the spans that are labeled in the example, and re-weight the loss by a factor of 0.01 (tuned on development data).

B GBP Training Details

The model takes BERT wordpieces from a publicly available BERT-base checkpoint[8] (Devlin et al., 2019). Intents have slots have randomly initialized 768-dimensional embeddings. The Transformer encoder uses 4 layers of cross-attention (Vaswani et al., 2017) with 4 attention heads and 768 dims and a dropout rate of 0.3.

We use a hidden size of 1024 for computing edge scores similarly to Dozat and Manning (2017). Cross entropy loss is minimized with the optimizer described in Devlin et al. (2019).

For partial supervision experiments, the loss is masked for unsupervised edges.

The model is trained over 20000 steps with a learning rate of 0.0001 and 2000 warmup steps. All hyperparameters are chosen based on validation set exact match accuracy performed by a grid search. BERT-base has approximately 110M parameters, and GBSP introduces approximately 13M additional parameters, for a total of approximately 123M parameters. Note that larger versions of BERT did not lead to performance improvements in our experiments.

Note a comparison on validation performance can be found in Table 3 (the validation set without unsupported has 4032 examples). All tested values for hyperpameters can be found in Table 4. We estimate approximately 1,000 total training runs during the development cycle. After tuning hyperparameters on the full set, no re-tuning occurred: partial supervision and data efficiency experiments used the same setup. Model training takes approximately 45 minutes.

Model	Acc
T5 (Raffel et al., 2020)	85.22
GBP (Proposed)	85.17
FSP (Pasupat et al., 2019)	84.53
PtrGen (Ours)	85.08

Table 3: Accuracy results for the TOP dataset evaluated on the validation set. Note all models besides T5 are initialized from BERT-Base

Parameter	Start	End	Incr.	Num
Node Encoder Dim	128	2048	x2	5
Biaffine Hidden Dim	128	2048	x2	5
Learning Rate	0.0001	0.01	x10	3
Number of Heads	2	8	x2	3
Warmup	2000	4000	x2	2
Number of Layers	1	8	x2	4

Table 4: Hyperparameter sweep for GBSP.

[7] https://github.com/google-research/text-to-text-transfer-transformer

[8] https://huggingface.co/google/bert_uncased_L-12_H-768_A-12

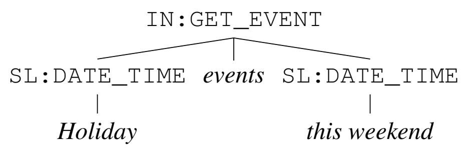

Figure 5: Example TOP tree with two occurrences of SL:DATE_TIME. When mapping from TOP trees to the parse trees predicted by our model, each instance of SL:DATE_TIME is assigned an index based it its preorder position in the TOP tree.

C Repeated Nodes

See Figure 5 for an example of a TOP tree with repeated nodes.

We chose to pad occurrences based on the observation that certain nodes can occur more times than they do in the training set. About half of the nodes only ever occur once. On the validation set, 2 additional replications was the highest value before performance degraded.

There are many alternatives to our handling of repeated nodes. For instance, Zhang et al. (2019) had a slightly different task, but we could have adopted their approach of generating the node set auto-regressively. Unfortunately, this would have complicated our method of partial supervision. Another method would be to use a fixed number of duplications: this worked slightly worse in practice, based on validation set performance. Alternatively, the model could have learned a regression, which has been used in non-autoregressive machine translation (e.g., Wang et al. 2019b). We leave trying such an approach to future work.

D Full Data

Results on the full dataset (including unsupported intents) can be found in Table 5. Note that most recent work does not report on this setting. For this setting, we train on every example in *train* (31279 examples) and evaluate on every example in *test* (9042 examples). Note that the full dataset is available at http://fb.me/semanticparsingdialog; unsupported intents were excluded manually with string matching.

Model	Acc
RNNG (Dyer et al., 2016)	78.51
GTCV (Shao et al., 2020)	82.51
T5 (Raffel et al., 2020)	84.11
GBP (Proposed)	83.31

Table 5: Accuracy results for the TOP dataset evaluated on all test examples, including those with unsupported intents.