BenchCLAMP: A Benchmark for Evaluating Language Models on Syntactic and Semantic Parsing

Anonymous ACL submission

Abstract

Recent work has shown that generation from a prompted or fine-tuned language model can perform well at semantic parsing when the output is constrained to be a valid semantic representation. We introduce **BenchCLAMP**, a **Bench**mark to evaluate **C**onstrained **LA**nguage **M**odel **P**arsing, that includes context-free grammars for seven semantic parsing datasets and two syntactic parsing datasets with varied output meaning representations, as well as a constrained decoding interface to generate only valid outputs covered by these grammars. We provide low, medium, and high resource splits for each dataset, allowing accurate comparison of various language models under different data regimes. Our benchmark supports evaluation of language models using prompt-based learning as well as fine-tuning. We benchmark seven language models, including two GPT-3 variants available only through an API. Our experiments show that encoder-decoder pretrained language models can achieve similar performance or even surpass state-of-the-art methods for both syntactic and semantic parsing when the model output is constrained to be valid.

1 Introduction

Large pretrained language models can achieve state-of-the-art performance on a host of NLP tasks when fine-tuned on target data (Liu et al., 2019; Raffel et al., 2020; Wang et al., 2021b; He et al., 2021). Models like GPT-3 (Brown et al., 2020), Codex (Chen et al., 2021) and T0 (Sanh et al., 2021) have also shown impressive zero- and few-shot performance when prompted only with task descriptions and examples. Research on large language models is typically validated by performance on downstream NLP tasks. Past work has evaluated new pretrained language models on classification, extraction, and generation, among others (Liu et al., 2019; He et al., 2021; Liang et al., 2022). However, parsing tasks are generally not considered a testbed for such evaluation. The outputs of parsing tasks are structured objects such as parse trees or code. State-of-the-art systems thus involve task- or dataset-specific model architectures and meaning representation constraints. Evaluating language models on parsing tasks test capabilities not captured by commonly used evaluation tasks.

Recently, Shin et al. (2021) and Scholak et al. (2021) have shown that standard generation from a fine-tuned or few-shot prompted language model can perform competitively in semantic parsing tasks, when the output of the language model is constrained to produce valid meaning representations. However, it is still challenging to set up constrained generation for a new dataset and language model due to the variation in meaning representations and model-specific tokenization. In this paper, we introduce a new benchmark called BenchCLAMP (Benchmark for Constrained Language Model Parsing) that covers nine parsing datasets with seven different meaning representations. We release context-free grammars for each dataset and provide a toolkit to perform efficient constrained decoding to generate valid meaning representations. Our benchmark reduces the barrier for language model developers to evaluate on parsing. The benchmark will be made available upon publication.

We benchmark seven pretrained language models using BenchCLAMP. We find that fine-tuning encoder-decoder pretrained language models can come close to or surpass the performance of state-of-the-art methods on all parsing datasets. Constrained generation via a domain-specific grammar provides performance gains for most fine-tuned language models. The improvement is high in low-resource settings but the relative improvement reduces when more training data becomes available. We find constrained decoding to be essential for few-shot prompted models and for tasks with complex constraints like constituency and dependency

parsing. In both these cases, we find language models struggle to generate valid representations without constrained decoding. In addition, we ablate different ways to encode context in the input, prompt structure, and retrieval methods for few-shot prompted language models. We present a comprehensive study establishing concrete techniques to reliably use language models for syntactic and semantic parsing tasks.

2 Related Work

Language Models for Semantic Parsing Recent work has shown that one can generate an analysis of a natural language sentence, such as a semantic parse, by asking a large language model to continue a prompt that includes the sentence (Chen et al., 2021; Li et al., 2021; Schucher et al., 2021). We refer to this as "language model parsing." To avoid ill-formed analyses, it is possible to *constrain* the generation so that the generated output satisfies hard task-specific constraints. Shin et al. (2021) showed that constrained generation from few-shot prompted GPT-3 and fine-tuned BART models outperformed task-specific semantic parsing architectures in low-resource settings. Scholak et al. (2021) were able to achieve state-of-the-art performance in SQL prediction by fine-tuning a T5-3B model (Raffel et al., 2020) and using constrained decoding. Recent work on AMR parsing has also shown positive results with sequence-to-sequence training with pretrained language model parameters (Cai and Lam, 2020; Bai et al., 2022). As the above works used different evaluation settings, it is hard to see which techniques work best under different data regimes.

Language Models for Syntactic Parsing Syntactic parsing tasks like constituency and dependency parsing requires the outputs to be well aligned with the input. All input tokens need to be covered by the output constituency or dependency parse. As a result, most solutions for syntactic parsing has involved custom decoders or inference algorithms (Kitaev and Klein, 2018; Yang and Tu, 2022; Zhang et al., 2017; Liu and Zhang, 2017). However there has been some work on linearizing the output and developing models that learn to generate these linearized sequences (Koo et al., 2015; Wiseman and Rush, 2016; Li et al., 2018; Fernández-González and Gómez-Rodríguez, 2020).

NLP Benchmarks Multiple benchmarks have been introduced to track progress on specific NLP tasks and to encourage multi-task learning with diverse datasets. The GLUE (Wang et al., 2018) and SuperGLUE (Wang et al., 2019) benchmarks are widely used by language model developers. More recently, BIG-bench (Srivastava et al., 2022) and HELM (Liang et al., 2022) were introduced to study language model capabilities.

However, these benchmarks focus on classification, span extraction and generation, and do not include structured prediction tasks like semantic parsing. More recently, the UnifiedSKG (Xie et al., 2022) benchmark has been introduced that converts a suite of tasks requiring structured knowledge into text-to-text format. In contrast to their work, we cover a wide range of parsing tasks covering AMR, SMCalFlow, constituency and dependency parsing. While they focus on unconstrained generation, we develop grammars and constrained decoding interface to support valid representation generation from language models.

3 Benchmark Details

3.1 Data Setup

BenchCLAMP includes nine popular parsing datasets with a varied set of meaning representation formalisms (details in Table 1). For each dataset, we create the following splits:

1. We create **three low-resource train splits** of 500 examples, each uniformly sampled from the training portion of the dataset. We create a single low-resource development set of 50 examples sampled from the development portion of the dataset. We report mean of these splits.

2. We similarly create **a medium-resource train split** of 5000 examples paired with a dev set of 500 examples.

3. We consider **a high-resource split** with the entire training set of the dataset, paired with the medium-resource development set.

To make it feasible for researchers to evaluate on BenchCLAMP, we randomly sample a smaller test set for datasets with large released test sets. Specifically, we sample 2000 examples from the test sets of SMCalFlow, TreeDST and MTOP datasets and evaluate test performance on this smaller set. We use the full test set for all other datasets. We also release a smaller randomly-sampled 100-example

Dataset	Metric	Example Representation
SMCalFlow (Andreas et al., 2020) TreeDST (Cheng et al., 2020)	Lispress Match	`(Yield (Event.start (FindNumNextEvent (Event.subject_? (?~= "meeting")) 1L)))`
MTOP (Li et al., 2021)	Exact Match	`[IN:Get_Message [SL:Type_Content video] [SL:Sender Atlas]]`
Overnight (Wang et al., 2015)	Denotation Match	`(call listValue (call getProperty en.block.block1 (string color)))`
Spider (Yu et al., 2018) CoSQL (Yu et al., 2019)	Test suite Execution	`SELECT born_state FROM head GROUP BY born_state HAVING count(*) >= 3`
AMR 2.0 (Banarescu et al., 2013)	Smatch	`(e/establish-01 :ARG1 (m/model :mod (i/innovate-01 :ARG1 (i2/industry))))`
PTB 3 (Constituency parsing) (Marcus et al., 1993)	EVALB	`(S (NP (NNP Ms.) (NNP Haag)) (VP (VBZ plays) (NP (NNP Elianti))) (. .))`
UD-EWT (Dependency parsing) (Zeman et al., 2022)	LAS	`(1-Read, root, 0) (2-some, obj, 1) (3-of, case, 6) (4-the, det, 6) (5-following, amod, 6) (6-links, nmod, 2)`

Table 1: List of datasets covered by BenchCLAMP, along with evaluation metric and an example representation. All representations are linearized into a sequence that can be produced by a language model. We use task and dataset specific metrics to evaluate performance.

test set for each dataset to evaluate models accessed through costly API calls like GPT-3 and Codex. Results on a 100-example test set will have wide error bars and should be used with caution. See subsection 5.6 for a discussion on result variance. We allow for the evaluation on full test sets of the datasets to compare with state of the art results. For datasets that do not release public test sets, like SMCalFlow, Spider, and CoSQL, we treat the development set as the test set, and sample 10% of the training set and treat it as the development set for splits creation. For datasets that include dialogue interactions, we ensure that all turns of a dialogue belong to the same split. The Overnight train set was already small (< 5k examples), so we do not have a separate medium split for it.

Linearizing Representations We use the dataset representations as is for the MTOP, PTB-3 and the SQL datasets. For AMR, we use the setup provided by van Noord and Bos (2017a,b) to linearize the representations into sequences for training, and to convert output model predictions to AMRs for evaluation. For SMCalFlow and TreeDST, we use the Lispress format (LISP-like serialization for programs) of the data released by Platanios et al. (2021). We linearize dependency parses into a sequence of dependency triples. For the example in Table 1, our representation has the following form:

> (Read, root, root) (some, obj, Read) (of, case, links) (the, det, links) (following, amod, links) (links, nmod, some)

To convert such a representation back to a dependency parse, we find head token indices for each token based on string match with the predicted head token. In case there are multiple mentions of the head token, we select the one that is closest to the token being considered.[1]

3.2 Grammars

We release context-free grammars for all datasets to constrain generation to valid meaning representations. The grammar creation process is specific to each dataset.

1. For SMCalFlow and TreeDST, we use the Lispress-format datasets released by Platanios et al. (2021). We create a non-terminal corresponding to each type present in the training data. For each (sub-)expression with type t, we add a production rule with the non-terminal for t generating the non-terminals of its component (sub-)expression types, or component terminal plan fragments.

2. For MTOP, we add a non-terminal corresponding to each intent and slot. Each intent non-terminal generates an expression comprising slot non-terminals. Similarly each slot non-terminal can generate an expression with nested intent non-terminals. Slot non-terminals can also generate terminal strings copied from the input utterance.

[1] We can correctly roundtrip 95.7% of parses in the test set using this approach.

3. We use a publicly available SQL grammar (antlr, 2022) for Spider and CoSQL. For each example, we add schema-specific constraints to the grammar to generate consistent table and column names. This is similar to "parsing without guards" in Scholak et al. (2021).

4. For constituency parsing, we define a non-terminal for an expression and a constituent label. We add production rules where the label non-terminal can produce any of the constituent labels seen in the training data. The expression non-terminal can either produce terminal tokens or generate a constituent label coupled with a expression non-terminal. This context free grammar covers constituent parse tree representation shown in Table 1. To additionally ensure that all tokens in the input utterance are covered by the generated parse tree, we additionally maintain a state in our parsing algorithm during decoding, allowing tokens to appear in the order seen in the utterance, and allowing the generation to end only when all input tokens have been generated.

5. For dependency parsing, we extract the set of dependency relations from the training set and define a non-terminal that can produce any of them. We then define a non-terminal that can generate a sequence of triples each comprising two tokens from the utterance and a dependency relation. Similar to our approach with constituency parsing, we maintain a parse state during decoding to ensure all tokens from the utterance are covered in order in the generated output.

For all data splits, we use the full training data to derive the grammar. We envision that in realistic scenarios, the grammar will be provided by a domain developer, and hence will have complete coverage of the domain (even when some plan fragments might not have appeared in the low-resource dataset). We also add results with grammars induced from low-resource splits in section 5.5.

4 Experimental Setup

4.1 Language Models Evaluated

We use BenchCLAMP to fine-tune and evaluate five language models with varying number of parameters: T5-base (220M), T5-large (770M), T5-3B (3B) (Raffel et al., 2020), BART-large (406M) (Lewis et al., 2020) and CodeT5-base (220M) (Wang et al., 2021a). The input to our model is the utterance concatenated with the string representation of the context (conversation context, database schema, etc.), and the output is the target parse. We evaluate two large GPT-based language models: GPT-3 (Brown et al., 2020) and Codex (Chen et al., 2021), using few-shot prompting on the 100 example test sets. Unless otherwise state, we use the OpenAI API `text-davinci-001` for GPT-3 and `code-davinci-001` for Codex. For each input (utterance concatenated with context) we select a set of 20 relevant examples from the training set using BM25 (Rubin et al., 2021) or a sentence transformer (Reimers and Gurevych, 2019) based similarity model. We create a prompt using these examples, following the template in Shin et al. (2021) and limiting the total length of the prompt to be 1500 tokens. This leaves room in GPT-3's buffer to generate an output of up to 548 tokens.

We release data splits for all domains of Overnight and all languages in MTOP. But for brevity, we benchmark on a single domain of Overnight (blocks) and a single language from MTOP (English). All other datasets used in the benchmark are in English. We evaluate few-shot prompted GPT-3 / Codex on three BenchCLAMP datasets to save on API costs. All other models are evaluated on the complete evaluation suite.

We use the code released by Shin et al. (2021) to support incremental constrained generation of semantic representations. This code maintains a chart according to Earley's algorithm (Earley, 1970) that can be used to determine the set of legal next tokens and can also be efficiently updated after a particular token is selected. We extend their method to support all autoregressive language models and sequence-to-sequence models. Unless otherwise mentioned, we always use constrained decoding to report metrics.

4.2 Format for Model Inputs

For experiments related to fine-tuned language models with SMCalflow and TreeDST with last user and agent utterance as context, the input to the model has the format $l \mid a \mid u$, where u is the input natural language utterance, l is the last user utterance, a is the last agent utterance and $\mid$ is a separator symbol. When using only last agent utterance as context, the input is $a \mid u$, and for using no context, the input to the model is simply u.

LM	Grammar Constraints?	SMCalflow			TreeDST			MTOP (en)		
		Low	Med	High	Low	Med	High	Low	Med	High
GPT-3† (BM25)	Yes	26.0	48.0	49.0	35.7	54.0	53.0	46.3	56.0	57.0
Codex† (BM25)	Yes	36.7	55.0	56.0	46.3	61.0	62.0	53.3	68.0	72.0
GPT-3† (SentenceT5-xxl)	Yes	33.0	45.0	52.0	38.7	56.0	59.0	50.0	62.0	64.0
Codex† (SentenceT5-xxl)	Yes	39.3	52.0	62.0	47.7	58.0	64.0	55.7	67.0	70.0
T5-base	No	38.2	67.5	77.6	57.2	84.4	89.3	54.7	79.3	84.5
	Yes	41.6	69.7	78.6	62.0	85.8	89.4	57.5	80.1	84.3
CodeT5-base	No	33.3	65.6	80.8	50.3	83.5	90.3	44.1	75.6	80.6
	Yes	37.3	67.5	81.1	56.8	84.4	90.0	47.1	75.8	81.1
BART-large	No	36.1	68.1	82.2	52.0	84.0	90.2	57.8	81.6	85.8
	Yes	42.5	71.4	**83.0**	61.1	86.4	89.8	61.7	82.1	85.3
T5-large	No	42.6	71.5	81.3	59.6	86.2	90.0	55.4	82.1	85.6
	Yes	46.3	73.1	82.1	**64.2**	**87.2**	90.1	59.3	82.5	85.3
T5-3B	No	43.5	73.8	82.6	58.5	85.9	**90.7**	60.9	83.2	**86.3**
	Yes	**48.7**	**75.9**	**83.0**	64.1	**87.2**	90.3	**64.1**	**83.4**	85.6

Table 2: Performance of language models on SMCalFlow, TreeDST and MTOP. † indicates few-shot prompted and evaluated on the 100 example test set; numbers above the horizontal bar are thus not comparable to those below. Remaining LMs are finetuned and evaluated on BenchCLAMP test sets. We report results with both constrained and unconstrained decoding to illustrate the contribution of constraints. Metrics are dataset-specific (see Table 1). The best score in each column is boldfaced.

LM	Grammar Constraints?	Spider			CoSQL			Overnight (blocks)	
		Low	Med	High	Low	Med	High	Low	High
T5-base	No	30.8	54.1	55.6	24.0	39.9	40.8	63.9	63.7
	Yes	33.8	56.8	58.9	26.8	42.7	43.4	**64.4**	63.9
CodeT5-base	No	36.6	57.4	61.9	26.1	45.9	48.1	60.0	64.7
	Yes	37.6	58.0	62.2	27.3	46.2	48.2	60.4	65.2
BART-large	No	41.8	59.1	63.9	30.9	49.9	52.8	60.7	63.4
	Yes	42.5	62.7	63.9	29.1	48.8	51.5	61.2	63.7
T5-large	No	42.4	64.6	65.7	30.7	50.0	53.8	62.1	68.7
	Yes	44.1	65.5	66.5	32.1	52.4	55.5	62.8	**68.9**
T5-3B	No	46.4	68.4	70.9	32.6	54.7	53.4	62.8	66.2
	Yes	**48.6**	**70.3**	**72.3**	**34.7**	**56.4**	**56.2**	63.2	66.2

Table 3: Performance of fine-tuned language models on Spider, CoSQL and Overnight datasets. We report test suite execution accuracy (Zhong et al., 2020) for the SQL datasets and denotation accuracy for Overnight.

LM	Grammar Constraints?	PTB-3			UD-EWT			AMR 2.0		
		Low	Med	High	Low	Med	High	Low	Med	High
T5-base	No	-	-	-	-	-	-	52.7	72.0	75.0
	Yes	83.1	93.1	94.6	80.2	88.2	89.4	51.3	72.0	75.0
CodeT5-base	No	-	-	-	-	-	-	48.0	66.0	74.0
	Yes	70.9	86.7	92.1	73.2	84.0	85.8	46.7	66.0	74.0
BART-large	No	-	-	-	-	-	-	57.0	74.0	81.0
	Yes	78.0	93.9	95.7	**84.1**	89.7	90.6	55.0	75.0	81.0
T5-large	No	-	-	-	-	-	-	57.3	76.0	81.0
	Yes	**84.5**	**94.4**	95.7	80.6	90.1	91.0	57.0	76.0	81.0
T5-3B	No	-	-	-	-	-	-	**60.0**	**77.0**	82.0
	Yes	77.6	93.9	**96.2**	83.1	**90.4**	**91.3**	59.0	**77.0**	**83.0**

Table 4: Performance of fine-tuned language models on constituency (PTB-3), dependency (UD-EWT) and AMR paring. We report bracketing F1 using EVALB (Sekine and Collins, 1997) for PTB-3, labeled attachment score (LAS) for UD-EWT, and Smatch (Cai and Knight, 2013) for AMR parsing.

We use the the following format for Spider and CoSQL: c , d , u, where c is any conversational context if applicable, d is a rendering of the database schema with or without values and u is the user utterance. We use the database schema representation used in Scholak et al. (2021) for d. For c, we concatenate the past utterances in the conversational context with the separator symbol |.

Our few-shot prompting experiments use the prompt template of (Shin et al., 2021). Given a language input, we retrieve relevant prompt examples and create a prompt with the following format:

```
Let's translate what a human user says
    into what a computer might say.

Human: {Prompt Example 1 Input}
Computer: {Prompt Example 1 Output}
Human: {Prompt Example 2 Input}
Computer: {Prompt Example 2 Output}
          ...
Human: {Current Input}
Computer:
```

4.3 Training Details

For fine-tuning experiments, we train the language models with batch size 32 for 10 000 steps using AdaFactor (Shazeer and Stern, 2018), saving a checkpoint every 5000 steps. We use 1000 linear warmup steps and then linear decay the learning rate to 0. We tune all models with learning rates 10^{-4} and 10^{-5}, except for T5-3B for which we only used 10^{-4} to save compute. The best performing checkpoint on the dev set is used to report scores on the test set.

5 Results

5.1 Benchmarking Language Models

We show the performance of language models on BenchCLAMP datasets in tables 2, 3 and 4. We find that performance increases with model size for most fine-tuned language models. Few-shot prompting of GPT-3 and Codex are still not on par with fine-tuned models. For non-SQL datasets, even smaller language models reach close to the best performance in the high resource setting. However, for Spider and CoSQL, model size seems important in all data settings. This is likely because the model has to reason about the database schema to generate SQL queries, making it a harder learning problem. See sections 5.3 and 5.4 for details on how we use context in model inputs for these experiments. We skip unconstrained generation results for constituency and dependency parsing. We observe that models often fail to generate valid parses for the entire sentence for these tasks, and evaluating a valid substring is not supported by released evaluation tools. Entirely rejecting invalid parses leads to very low performance.

Table 5 compares our constrained T5-3B model with the best-performing models in the literature. We outperform state-of-the-art models on the SMCalFlow, TreeDST and Overnight-blocks datasets. For Spider and CoSQL, our scores are lower than the state-of-the-art despite using a similar constrained language model approach. This is likely because we use a general SQL grammar to constrain decoding, whereas the SQL in these datasets covers only a small fraction of the SQL grammar. We believe using a more constrained grammar will improve performance.The state-of-the-art method for AMR also uses a CLAMP model but with additional task specific pretraining. The best models for PTB-3 and UD-EWT are designed specific to the task. Our language model fine-tuning paired with constrained decoding achieves performance close to these methods without any task specific modifications.

Dataset	Current State of the Art	Our T5-3B
SMCalFlow	80.4 (Platanios et al., 2021)	**83.7**
TreeDST	88.1 (Platanios et al., 2021)	**91.5**
MTOP (en)	**86.4** (Pasupat et al., 2021)	85.7
Overnight (blocks)	65.2 (Cao et al., 2019)	**66.2**
Spider	**75.5** (Scholak et al., 2021)	72.2
CoSQL	**56.9** (Scholak et al., 2021)	52.3
AMR	**85.4** (Bai et al., 2022)	83.0
PTB 3	**96.4** (Tian et al., 2020)	96.2
UD-EWT	**91.5** (Mohammadshahi and Henderson, 2021)	91.3

Table 5: Comparison of our fine-tuned T5-3B model with current state of the art models on full test sets. We report exact match accuracy for Spider and CoSQL to match the settings of previous work. The best score in each row is boldfaced.

5.2 Effect of Constraints

Decoding constrained by a grammar is essential for few-shot prompted models like GPT-3 and Codex that are accessed via API calls. Without a grammar, we noticed that these models explore a large number of invalid paths leading to high latency and API cost. We also found constrained decoding essential for source side prediction tasks like constituency

and dependency parsing. They require each token in the input to be covered in the output. We found even fine-tuned language models struggle to learn these constraints leading to very low performance with unconstrained decoding.

Tables 2, 3 and 4 show the effect of constraints while generating from fine-tuned large language models. We find that constrained decoding is most beneficial in low-data regimes, giving on average 2.7% gain over unconstrained decoding. In the high-resource setting, the average gain is less than 1%, suggesting that the full data is nearly sufficient to learn the constraint system.

We find that constrained decoding under performs unconstrained decoding for some settings. We attribute this result to the insufficient coverage of our grammars. Our grammars were induced from the training data, and thereby fail to cover novel components or combinations at test time. Our grammars are also constrained to copy quoted strings from the input utterance. However this is not strictly followed in some datasets. Table 6 shows the fraction of test outputs covered by BenchCLAMP grammars. We could relax the grammar constraints to ensure full coverage; we leave such exploration to future work. In realistic situations, we expect grammars to be provided by domain developers ensuring full coverage.

Dataset	Test Set Coverage %
SMCalFlow	99.6
TreeDST	96.3
MTOP (en)	96.6
Overnight (blocks)	100.0
Spider	98.8
CoSQL	99.4
AMR	99.7
PTB 3	100.0
UD-EWT	99.9

Table 6: Grammar coverage (%) of the gold outputs in the test set of BenchCLAMP datasets.

5.3 Impact of Context

The datasets in BenchCLAMP require a model to use a variety of contexts. SMCalFlow, TreeDST and CoSQL datasets all have conversational context. Spider and CoSQL have database schema context which informs the target SQL prediction. BenchCLAMP allows us to perform a controlled investigation of the effect of context. Table 7 shows that while using the last agent and user utterance is helpful for all settings, the low-data regime does even better when using only the last agent utterance; without more data, training struggles to learn how to utilize (or ignore) the additional context. We find similar results for CoSQL in Table 8. Also, SQL prediction always benefits from including database values in the context along with the database schema information. We use the best settings for context for each data regime for benchmarking results in tables 2, 3 and 4.

Conv. Context	SMCalFlow			TreeDST		
	Low	Med	High	Low	Med	High
No context	37.0	63.8	72.9	42.4	68.4	76.0
Last agent utt.	**42.6**	70.7	80.6	**59.6**	82.7	87.9
Last user & agent utt.	40.0	**71.5**	**81.3**	58.8	**86.2**	**90.0**

Table 7: Lispress match accuracy of unconstrained fine-tuned T5-large with varying conversational context on SMCalFlow and TreeDST. We find more context hurts in low resource settings but helps in medium and high resource settings.

Conversational Context	DB values?	Low	Med	High
No context	no	21.3	35.9	34.4
	yes	25.3	38.9	40.3
Last interaction	no	24.1	40.4	39.1
	yes	**28.2**	44.2	44.4
All interactions	no	24.3	36.8	43.0
	yes	24.9	**44.9**	**48.8**

Table 8: Test suite execution accuracy of unconstrained BART-large on the CoSQL dataset with varying context. We similarly find more context hurts in low resource settings but helps in medium and high resource settings.

5.4 Few-Shot Prompting

In few-shot prompting scenario, we manipulate the context choice and ordering of examples in our prompt to Codex. The results in Table 10 show that ordering the most similar example at the end closest to the generation heads is helpful in the low-data regime, indicating that GPT-3 and Codex pay more attention to the recent past. In higher data regime, all prompt examples are almost equally relevant, hence the order does not matter as much. We find that context does not help; one of the reasons being that we can fit fewer examples in the prompt if we need to include context for each example.

We experiment with BM25 and similarity models for prompt retrieval for few-shot prompting. For similarity models, we pick top models from each

of the three categories in SentenceTransformers leaderboard (Reimers and Gurevych, 2019): all-mpnet-base-v2, multi-qa-mpnet-base-dot-v1 and sentence-t5-xxl (Ni et al., 2022). We find that SentenceT5-xxl surpasses other similarity models for prompt retrieval. SentenceT5 outperforms BM25 in low resource settings, but performs relatively worse when more data is available.

Prompt Order	Conv. Context	Low	Med
Random	No context	35.7	52.0
Best First	No context	34.3	**53.0**
Best Last	No context	**36.7**	52.0
Best Last	Last agent utt.	34.0	41.0
Best Last	Last user & agent utt.	26.0	31.0

Table 9: Lispress match accuracy of the few-shot prompted Codex DaVinci language model on the SMCalFlow dataset with different prompt order and conversational context. More context hurts few-shot prompted models. Ordering the most relevant examples closer to the generation heads improve performance.

Prompt Retrieval Method	Low	Med
BM25	36.7	**55.0**
all-mpnet-base-v2	38.0	49.0
multi-qa-mpnet-base-dot-v1	38.3	50.0
sentence-t5-xxl	**39.3**	52.0

Table 10: Lispress match accuracy of Codex on the SMCalFlow dataset with different prompt retrieval methods. We used `code-davinci-001` with best last prompt order and no context.

Constraint Grammar	SMCalFlow		TreeDST	
	Low	Med	Low	Med
Unconstrained	42.6	71.5	59.6	86.2
Induced from train split	45.6	73.1	62.3	87.2
Induced from full train	46.3	73.1	64.2	87.2

Table 11: Effect of different grammar induction data on the Lispress match constrained decoding accuracy of fine-tuned T5-large.

5.5 Grammars induced from Less Data

The grammars released for SMCalFlow, TreeDST and MTOP were induced using the full train dataset. This grammar is then used even with low and medium resource train splits. We expect the grammar will be provided by a developer of the domain and hence will cover all valid representations. However, for the sake of completeness, we report here the impact of using grammar induced from the corresponding train sets. Table 11 shows the results with constrained decoding with train split induced grammar, and compares the performance with unconstrained decoding and decoding with grammar induced from full train set. The gains from constraints drop by $1-2\%$ for low resource splits when using train split induced grammar instead of full train induced grammar. It does not affect results for the medium resource splits.

5.6 Variance of Results

All low-resource results are a mean of three training data splits. Table 12 reports the average standard deviation for each model over the three low resource splits. We find a high standard deviation of GPT-3 and Codex; one of the factors being the small size of the test set (100 examples). Fine-tuned models show relatively low variance, consistently having standard deviation lower than 2%.

Model	Avg. Standard Deviation
GPT-3	4.7
Codex	3.2
T5-base	1.2
CodeT5-base	1.1
BART-large	1.4
T5-large	2.0
T5-3B	1.5

Table 12: Standard deviation of the scores for each language model over the three low resource splits.

6 Conclusion

We introduce a benchmark comprising nine parsing datasets with varying target representations. We support few-shot prompting, fine-tuning and constraint decoding for all autoregressive language models and sequence-to-sequence models on these datasets. We hope that this work will encourage language model developers to consider parsing tasks as a test-bed in future work.

7 Limitations

Our benchmark includes data in multiple languages (all languages included in the MTOP dataset) but we only evaluate on English datasets due to compute constraints. Few-shot prompted experiments were evaluated on relatively small test sets on three datasets due to API cost limitations. As a result, we noticed high variance in the results (see section 5.6 for variance results).

References

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571.

antlr. 2022. grammars-v4. https://github.com/antlr/grammars-v4.

Xuefeng Bai, Yulong Chen, and Yue Zhang. 2022. Graph pre-training for AMR parsing and generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6001–6015, Dublin, Ireland. Association for Computational Linguistics.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria. Association for 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. *Computing Research Repository*, arXiv:2005.14165.

Deng Cai and Wai Lam. 2020. AMR parsing via graph-sequence iterative inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1290–1301, Online. Association for Computational Linguistics.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752, Sofia, Bulgaria. Association for Computational Linguistics.

Ruisheng Cao, Su Zhu, Chen Liu, Jieyu Li, and Kai Yu. 2019. Semantic parsing with dual learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 51–64, Florence, Italy. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Jianpeng Cheng, Devang Agrawal, Héctor Martínez Alonso, Shruti Bhargava, Joris Driesen, Federico Flego, Dain Kaplan, Dimitri Kartsaklis, Lin Li, Dhivya Piraviperumal, Jason D. Williams, Hong Yu, Diarmuid Ó Séaghdha, and Anders Johannsen. 2020. Conversational semantic parsing for dialog state tracking. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8107–8117, Online. Association for Computational Linguistics.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2020. Enriched in-order linearization for faster sequence-to-sequence constituent parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4092–4099, Online. Association for Computational Linguistics.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. *CoRR*, abs/2111.09543.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.

Petrov Koo, S Petrov, I Sutskever, GE Hinton, O Vinyals, and L Kaiser. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*.

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.

Haoran Li, Abhinav Arora, Shuohui Chen, Anchit Gupta, Sonal Gupta, and Yashar Mehdad. 2021. 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.

Zuchao Li, Jiaxun Cai, Shexia He, and Hai Zhao. 2018. Seq2seq dependency parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3203–3214, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2022. Holistic evaluation of language models.

Jiangming Liu and Yue Zhang. 2017. In-Order Transition-based Constituent Parsing. *Transactions of the Association for Computational Linguistics*, 5:413–424.

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*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Alireza Mohammadshahi and James Henderson. 2021. Recursive non-autoregressive graph-to-graph transformer for dependency parsing with iterative refinement. *Transactions of the Association for Computational Linguistics*, 9:120–138.

Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith Hall, Daniel Cer, and Yinfei Yang. 2022. Sentence-t5: Scalable sentence encoders from pretrained text-to-text models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1864–1874, Dublin, Ireland. Association for Computational Linguistics.

Panupong Pasupat, Yuan Zhang, and Kelvin Guu. 2021. Controllable semantic parsing via retrieval augmentation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7683–7698, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Emmanouil Antonios Platanios, Adam Pauls, Subhro Roy, Yuchen Zhang, Alexander Kyte, Alan Guo, Sam Thomson, Jayant Krishnamurthy, Jason Wolfe, Jacob Andreas, and Dan Klein. 2021. Value-agnostic conversational semantic parsing. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3666–3681, Online. 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.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2021. Learning to retrieve prompts for in-context learning. *CoRR*, abs/2112.08633.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Stella Biderman, Leo Gao, Tali Bers, Thomas Wolf, and Alexander M. Rush. 2021. Multitask prompted training enables zero-shot task generalization.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*,

pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Nathan Schucher, Siva Reddy, and Harm de Vries. 2021. The power of prompt tuning for low-resource semantic parsing. *CoRR*, abs/2110.08525.

Satoshi Sekine and Michael Collins. 1997. Evalb bracket scoring program. *URL: http://www. cs. nyu. edu/cs/projects/proteus/evalb*.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4596–4604. PMLR.

Richard Shin, Christopher Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7699–7715, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Aarohi Srivastava, Abhinav Rastogi, et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *ArXiv*, abs/2206.04615.

Yuanhe Tian, Yan Song, Fei Xia, and Tong Zhang. 2020. Improving constituency parsing with span attention. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1691–1703, Online. Association for Computational Linguistics.

Rik van Noord and Johan Bos. 2017a. Dealing with co-reference in neural semantic parsing. In *Proceedings of the 2nd Workshop on Semantic Deep Learning (SemDeep-2)*, pages 41–49, Montpellier, France. Association for Computational Linguistics.

Rik van Noord and Johan Bos. 2017b. Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *Computational Linguistics in the Netherlands Journal*, 7:93–108.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021a. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. 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 1332–1342, Beijing, China.

Zirui Wang, Adams Wei Yu, Orhan Firat, and Yuan Cao. 2021b. Towards zero-label language learning. *CoRR*, abs/2109.09193.

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306, Austin, Texas. Association for Computational Linguistics.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. 2022. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *EMNLP*.

Songlin Yang and Kewei Tu. 2022. Bottom-up constituency parsing and nested named entity recognition with pointer networks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2403–2416, Dublin, Ireland. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. 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 1962–1979, Hong Kong, China. Association for Computational Linguistics.

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.

Daniel Zeman et al. 2022. Universal dependencies 2.11. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Zhirui Zhang, Shujie Liu, Mu Li, Ming Zhou, and Enhong Chen. 2017. Stack-based multi-layer attention for transition-based dependency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1677–1682, Copenhagen, Denmark. Association for Computational Linguistics.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-sql with distilled test suite. In *The 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.