Few-Shot Semantic Parsing with Language Models Trained on Code

Richard Shin
Microsoft Semantic Machines
richard.shin@microsoft.com

Benjamin Van Durme
Microsoft Semantic Machines
ben.vandurme@microsoft.com

Abstract

Large language models can perform semantic parsing with little training data, when prompted with in-context examples. It has been shown that this can be improved by formulating the problem as paraphrasing into *canonical utterances*, which casts the underlying meaning representation into a controlled natural language-like representation. Intuitively, such models can more easily output canonical utterances as they are closer to the natural language used for pre-training. Recently, models also pre-trained on code, like OpenAI Codex, have risen in prominence. For semantic parsing tasks where we map natural language into code, such models may prove more adept at it. In this paper, we test this hypothesis and find that Codex performs better on such tasks than equivalent GPT-3 models. We evaluate on Overnight and SMCalFlow and find that unlike GPT-3, Codex performs similarly when targeting meaning representations directly, perhaps because meaning representations are structured similar to code in these datasets.

1 Introduction

Semantic parsing is the task of mapping natural language to a target meaning representation. Many approaches have been explored by the community, including a recent focus on the use of large autoregressive language models (LMs). Such pretrained LMs can achieve surprising levels of accuracy with relatively small numbers of examples. Further gains have come from constraining a decoder to only consider syntactically valid outputs.

Historically, *language* models have been constructed using a large collection of *natural* language. And yet, the term "language" clearly applies to non-natural languages as well. Very large models have been trained on mixed corpora, explicitly curated to include code (programming language) as well as natural language. Examples include GPT-J (Wang and Komatsuzaki, 2021), MT-NLG (Kharya and Alvi, 2021), and Gopher (Rae et al., 2021), with OpenAI Codex (Chen et al., 2021), PaLM-Coder (Chowdhery et al., 2022), and Austin et al. (2021) particularly focused on code.

We revisit few-shot semantic parsing experiments from Shin et al. (2021), which used GPT-3 with constrained decoding into a controlled sublanguage of English (canonical utterances) then translated the canonical utterance output into the meaning representation using a synchronous context-free grammar (SCFG). In this work, we perform similar experiments on the Overnight (Wang et al., 2015) and SMCalFlow (Andreas et al., 2020) datasets,[1] but using OpenAI Codex instead. As Codex has been trained on code, including natural language comments that explain its intent, we hypothesize that Codex will be more adept at semantic parsing for meaning representations resembling code.

In this work, we find that:

- Codex substantially narrows the gap in accuracy between predicting meaning representations directly versus canonical utterances, thus obviating the need to define canonical utterances. We observe this even though the meaning representations use bespoke languages, rather than ones like Python which frequently appeared in the training data.
- Surprisingly, Codex also generates canonical utterances more accurately than GPT-3, even though those look more like English than code.
- Even with Codex, constrained decoding with a CFG and a non-greedy search procedure are still valuable in providing improved accuracy.
- *Speculative constrained decoding*, an adaptation of Poesia et al. (2022, Appendix F), gives comparable accuracy as beam search but with greater efficiency, on the language model APIs provided by OpenAI.

---

[1] Both are in English and available under CC BY-SA 4.0.

Dataset	Natural language	Canonical utterance	Meaning representation
SMCalFlow	Schedule Hide and Seek in the mall for Saturday night	create event called "Hide and Seek" starting next Saturday night at "mall"	`(Yield :output (CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event] :subject (?= #(String "Hide and Seek")) :start (DateTimeConstraint :constraint (Night) :date (NextDOW :dow #(DayOfWeek "SATURDAY"))) :location (?= #(LocationKeyphrase "mall"))))))`
Overnight Cal.	which meeting has the earliest end time	meeting that has the smallest end time	`(call listValue (call superlative (call getProperty (call singleton en.meeting) (string !type)) (string min) (call ensureNumericProperty (string end_time))))`

Table 1: Examples from the Overnight Calendar and SMCalFlow datasets.

2 Preliminaries

2.1 Constrained language model parsing

In semantic parsing, our goal is to convert an utterance u into the meaning representation m. We use the same approach as Shin et al. (2021): (1) priming the underlying language model with dynamically created prompts, (2) constrained decoder, and (3) optionally using a canonical utterance c as the target output instead of m.

Since GPT-3 and Codex can perform *in-context few-shot learning* (Brown et al., 2020), we retrieve 20 (u_i, m_i) pairs most similar[2] to u from the training set, then translate m_i into c_i if using canonical utterances, to form the prompt p which looks like:

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

Human: when is the standup ← $u_1$
Computer: start time of "standup" ← $c_1$
Human: what date is the standup ← $u_2$
Computer: date of "standup" ← $c_2$
[...]
Human: how long is the daily standup ← $u$
Computer:
```

where $italics$ are annotations for exposition in this paper, and not included verbatim in the prompt.

We then generate a completion for p using the language model, which we will take as the predicted value of canonical utterance c or meaning representation m, depending on our choice for (3). To ensure that the generated completion is well-formed, we assume the existence of a function $\texttt{nextTokens}(s) = \{w_i\} \subseteq \mathcal{V} \cup \{\texttt{EOS}\}$. For a given prefix s of a canonical utterance or meaning representation, this function returns the set of subsequent tokens that we can can append to s such that it remains a prefix of a well-formed c or m. It also indicates whether s is already a complete, well-formed c or m by including `EOS` in the result; if $\texttt{nextTokens}(s) = \{\texttt{EOS}\}$, then s is a valid canonical utterance or meaning representation with no possible extensions.

As an example, `nextTokens`(“`start time`”) would contain `of`, but not `EOS` or `in`. We use `nextTokens` to filter candidates from the language model such that it only generates grammatical outputs; if we build the completion by appending what `nextTokens` advises, we are guaranteed to obtain a grammatically conformant output. We implement `nextTokens` using a trie and a CFG for Overnight and SMCalFlow, respectively.

2.2 OpenAI language models

OpenAI operates a service offering GPT-3 (Brown et al., 2020) through a networked API. The API includes multiple variants of GPT-3, named Ada, Babbage, Curie, and Davinci, with the model size increasing in that order. Two Codex (Chen et al., 2021) models, which had code from GitHub included in their training data, are also offered. They are named Cushman Codex and Davinci Codex.[3]

The primary use case for the API is generating completions from a prefix, by sequentially sampling from $p(w_n | w_1 w_2 \cdots w_{n-1})$ until some limit is reached. The API provides for specifying a softmax temperature to modify this distribution, for example enabling greedy argmax sampling with a temperature of 0.0. The API also allows for directly querying $p(w_n | w_1 w_2 \cdots w_{n-1})$, but only returns probabilities for up to 100 most likely tokens; we use this capability for constrained beam search.

---

[2] We use GPT-3 itself for this, following Shin et al. (2021). The similarity function is identical for all our experiments, regardless of whether we use GPT-3 or Codex for decoding.

[3] We used the models available in late 2021; OpenAI may change them from time to time.

2.3 Experimental setup

We used two of the datasets from Shin et al. (2021) for our experiments. We build on their released code and use the same subsets of the training data. We briefly describe some of the details below.

Overnight. This dataset from Wang et al. (2015) contains 13,682 examples across eight different domains, curated to exhibit a variety of linguistic phenomena and semantic structures. We used 200 randomly-sampled training examples for each domain, and evaluate on the domains separately. For evaluation, we use denotational accuracy, based on comparing the execution results of the predicted and reference programs.

SMCalFlow. Introduced in Andreas et al. (2020), this task-oriented dialogue dataset consists of conversations about calendars, weather, places, and people. Each utterance u is annotated with dataflow programs m containing function composition, complex constraints, and references to computations from previous turns. Of the 133,821 (u_i, m_i) pairs in training, we use a stratified sample of 300 for our experiments, following Shin et al. (2021). For evaluation, we use syntactical match between the predicted and reference programs, which requires them to be structurally identical but allows differences of spacing and named arguments in function calls.

Test set sampling for certain experiments. As usage of GPT-3 and Codex requires significant resources, we conduct our initial experiments on smaller subsets of the evaluation sets. For Overnight, we used 100 uniformly sampled examples from test set for the calendar domain. For SMCalFlow, we used 200 uniformly sampled examples from the validation set.

We used the subsets for the experiments described in Sections 3.1 to 3.4. In the final experiments of Section 3.5, we use the full test set for Overnight and the full validation set for SMCalFlow.

3 Experiments

3.1 Comparing GPT-3 and Codex

Table 2 summarizes our initial comparison of the GPT-3 and Codex models when applied to semantic parsing. Davinci Codex performs better than Davinci on both Overnight Calendar and SMCalFlow when using identical settings. More interestingly, Cushman Codex, which is one step down from Davinci Codex, performs substantially better than Curie, which is one step down from Davinci. These results support our hypothesis that language models trained on code can perform better at semantic parsing.

	Accuracy	
Model	Overnight Cal.	SMCalFlow
Davinci	0.81	0.340
Curie	0.66	0.260
Davinci Codex	0.86	0.355
Cushman Codex	0.87	0.320

Table 2: Comparing various OpenAI models using constrained decoding to generate canonical utterances, with beam search having beam size 5. These results are on 100 sampled test examples. The larger Davinci models do better, the Codex models show better performance.

3.2 Targeting canonical utterances versus meaning representations

		Accuracy	
Model	Canonical	Meaning	$C - M$
Davinci	0.81	0.68	0.13
Davinci Codex	0.86	0.86	0.00

(a) Overnight Calendar

		Accuracy	
Model	Canonical	Meaning	$C - M$
Davinci	0.340	0.245	0.095
Davinci Codex	0.355	0.345	0.010

(b) SMCalFlow

Table 3: Differences in accuracy when using canonical utterances versus directly using meaning representations. Davinci Codex performs better on canonical utterances, but the gap is much smaller than with Davinci. Results using constrained decoding with beam size 5.

Shin et al. (2021) demonstrated that as language models have (conventionally) been trained to generate natural language, we would benefit by formulating semantic parsing as paraphrasing into a controlled sublanguage of English. In Table 3, we investigate whether that still holds true when using Codex. We observe that when using GPT-3 (Davinci), targeting meaning representations can result in more than a 25% relative drop in accuracy. In contrast, Davinci Codex exhibits no or a very small drop in accuracy when targeting meaning representations.

Notably, the meaning representations used for Overnight and SMCalFlow are in Lisp-like languages, rather than in programming languages common on GitHub. Our experiments indicate that Codex can nevertheless pick up on the semantics with only a few examples in the prompt.

Having canonical utterances as the target output still performs better than meaning representations. This holds true even though our evaluation procedure first translates canonical utterances back into meaning representations, which is a lossy procedure for SMCalFlow as described in (Shin et al., 2021). However, designing a suitable system of canonical utterances is a non-trivial effort. The smaller performance gap we observe with Codex changes the cost/benefit calculations on authoring SCFGs.

3.3 Value of constraints and beam search

As mentioned in Section 2.2, the primary capability of OpenAI's API is generating completions from a prefix using sequential sampling. Their documentation[4] suggests using it that way to generate code from comments, a similar task to semantic parsing. Nevertheless, we see in Table 4 that the use of constraints and beam search lead to benefits in accuracy. Even with constrained decoding, greedy argmax sampling (equivalent to a beam size of 1) performs worse than using beam search.

		Accuracy	
Decoding	Beam	Overnight Cal.	SMCalFlow
Constrained	5	0.86	0.345
Constrained	1	0.75	0.300
Unconstrained	5	0.80	0.315
Unconstrained	1	0.73	0.280

Table 4: Results comparing constrained with unconstrained decoding and multiple beam sizes, when generating meaning representations. Even when using Davinci Codex, trained specifically on code, constrained decoding and beam search lead to higher accuracy.

3.4 Speculative constrained decoding

While constrained decoding and beam search improve accuracy, they are slow to perform with OpenAI's API. Extending a partial hypothesis requires one network round-trip per token. The API lacks state and so each request includes the prompt and all previously generated tokens. In the worst case, the statelessness implies decoding will take $O(n^3)$ complexity rather than the typical $O(n^2)$ of transformers due to needing to re-encode the prefix each time. Even if the hidden states for previous tokens were cached, their retrieval and transfer to GPUs or other accelerators takes overhead.

As such, we adapt a method from Synchromesh (Poesia et al., 2022, Appendix F) to obtain the benefits of beam search and constrained decoding with greater efficiency. We extend Synchromesh's approach with a *width* parameter W, which functions similar to the beam size. We call it *speculative constrained decoding*.

To expand a partial hypothesis in the search, we query the API to create W completions with softmax temperature T.[5] The API samples from the model, without reference to any grammars, until `EOS` is sampled or a length limit is reached. Using the `nextTokens` function, we check each of the W completions left-to-right until we encounter an invalid token, and truncate there so that we only have valid tokens; we return the truncated completions as new hypotheses. If no completion contains any valid tokens, then we query the API for the W best tokens and return those as new hypotheses. As done in beam search, we start with a single empty hypothesis, and keep the W best expansions at each step. We stop after 16 steps if W complete hypotheses were not generated by then. More details are in Appendix D.

Table 5 shows the results from trying various values for W and T, along with beam search for $W = 1$ and $W = 5$. When $W = 1$ and $T = 0$, which is equivalent to Synchromesh's approach, we obtain very similar results to constrained greedy decoding (beam size 1). However, speculative constrained decoding is substantially faster.

In order to obtain results comparable to beam search with beam size 5, we require $W = 5$ or 10. In comparison, Synchromesh only supports $W = 1$. We see notable speedups compared to beam search, but typically obtain comparable accuracy.

We also observe that we can generate generate canonical utterances more quickly than meaning representations, as the canonical utterances are shorter. However, these timing results do not include the time required to convert canonical utterances into meaning representations.

---

[4] `https://beta.openai.com/docs/guides/completion/working-with-code`

[5] The softmax function with temperature T computes $\frac{\exp(x_i/T)}{\sum_{j=1}^{|V|}\exp(x_j/T)}$, to compute probabilities for each of the $|V|$ tokens in the vocabulary. As T approaches 0, the output becomes 1 for the largest value of x_i and 0 for all others, effectively computing the argmax.

		Overnight Calendar				SMCalFlow			
		Accuracy		Items/second		Accuracy		Items/second	
Width	Temperature	Canonical	Meaning	Canonical	Meaning	Canonical	Meaning	Canonical	Meaning
1	0.0	0.86	0.76	0.520	0.246	0.300	0.320	0.193	0.184
1	BS	0.84	0.75	0.237	0.059	0.305	0.300	0.116	0.040
5	0.5	0.87	0.80	0.380	0.155	0.335	0.315	0.076	0.140
5	1.0	0.87	0.85	0.260	0.145	0.325	0.330	0.076	0.034
5	BS	0.86	0.86	0.133	0.030	0.355	0.345	0.065	0.008
10	0.5	0.87	0.86	0.355	0.150	0.345	0.345	0.038	0.085
10	1.0	0.87	0.85	0.193	0.068	0.370	0.335	0.028	0.014

Table 5: Comparing various settings on speculative constrained decoding with beam search. "BS" indicates use of beam search. Speculative constrained decoding gets similar accuracy as beam search, but at higher speed.

3.5 Putting everything together

	Accuracy	
Model	Overnight Avg.	SMCalFlow
Shin et al. (2021), Constrained Canonical	0.765	0.32
Shin et al. (2021), Constrained Meaning	0.657*	0.25*
Ours, Canonical	0.785	0.342
Ours, Meaning	0.750	0.330

Table 6: Comparison to Shin et al. (2021). Results are on the entire test set for Overnight and the entire dev set for SMCalFlow. For Overnight, we took a simple average of the accuracy for each of the 8 domains. Results marked with * are on subsampled evaluation sets. We used speculative constrained decoding with a width of 10 and a temperature of 0.5.

As explained in Section 2.3, earlier results in this article are based on smaller subsets of the evaluation sets due to resource limitations. In Table 6, we evaluate on the full evaluation sets using lessons learned from our previous experiments. We achieve better accuracies than when Shin et al. (2021) used GPT-3. We re-confirm Section 3.2 that Codex performs nearly as well at meaning representations as canonical utterances.

4 Related Work

Chen et al. (2020) observed that for low-resource semantic parsing, fine-tuning a pretrained sequence-to-sequence model improved over the use of a pretrained encoder only. Scholak et al. (2021), Wu et al. (2021), and Shin et al. (2021) each proposed the use of constrained decoding for semantic parsing with LMs. The latter two works argued that language models were best used to parse language into controlled natural language, rather than directly to a code-like representation. Here we consider whether that conclusion changes based on new LMs that are trained with code.

Pasupat et al. (2021) proposed a retrieval-augmented solution to semantic parsing, which relates to the dynamic prompt selection of Shin et al. (2021), and which we followed here without alteration. Future work may consider the impact of more advanced prompt selection techniques.

5 Conclusion

We investigate the use of OpenAI Codex, a large language model trained on code, for few-shot semantic parsing. We find that it performs better than GPT-3 for our tasks. While constrained decoding and a non-greedy decoding procedure still non-trivially improve accuracy, mapping to canonical natural language is no longer as important with Codex, thereby lightening the burden on developing few shot semantic parsers based on large LMs.

Ethical Considerations

Our work heavily relies on OpenAI's GPT-3 and Codex models, which are large language models trained on big datasets. Such language models may reflect biases present in their training data (Brown et al., 2020; Bender et al., 2021). However, our use of constrained decoding largely mitigates the risks from such bias as we only allow the model to generate outputs allowed by a small grammar. Furthermore, the outputs are interpreted by machines rather than directly shown to humans. The potential for harm may increase when the grammars used in constrained decoding allow for a wider variety of outputs (such as including unconstrained free-text fields), and if semantic parsing is used for particularly sensitive domains.

Acknowledgements

We would like to thank Adam Pauls, Jason Eisner, Matt Gardner, and other colleagues at Microsoft Se-

mantic Machines, who provided helpful feedback and engaged in stimulating discussions which have greatly improved the paper.

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.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610623, New York, NY, USA. Association for Computing Machinery.

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. *CoRR*.

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.

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.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311.

Paresh Kharya and Ali Alvi. 2021. Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, the Worlds Largest and Most Powerful Generative Language Model. `https://developer.nvidia.com/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model`.

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.

Gabriel Poesia, Alex Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Chris Meek, and Sumit Gulwani. 2022. Synchromesh: Reliable code generation from pre-trained language models. In *ICLR 2022*.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, H. Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant M. Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake A. Hechtman, Laura Weidinger, Iason Gabriel, William S. Isaac, Edward Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2021. Scaling language models: Methods, analysis & insights from training gopher. *CoRR*, abs/2112.11446.

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.

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.

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. `https://github.com/kingoflolz/mesh-transformer-jax`.

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. Association for Computational Linguistics.

Shan Wu, Bo Chen, Chunlei Xin, Xianpei Han, Le Sun, Weipeng Zhang, Jiansong Chen, Fan Yang, and Xunliang Cai. 2021. From paraphrasing to semantic parsing: Unsupervised semantic parsing via synchronous semantic decoding. 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 5110–5121, Online. Association for Computational Linguistics.

A Measuring performance of beam search and speculative constrained decoding

For measuring the items/second of beam search and speculative constrained decoding in Table 5 and Table 8, we used the first 10 items of the evaluation sets. As we only had access to shared instances of GPT-3 and Codex, we were unable to guarantee lack of interference from other users. While the numbers are not precise, we believe they are generally indicative of the expected performance of the two methods.

B Prompt for Codex when using meaning representations

Instead of the prompt in Section 2.1, we used the prompt depicted below:

```
;;; Translate questions into Lisp
    expressions

; [utterance from training example]
[meaning representation from example]
; [utterance from training example]
[meaning representation from example]
[...]
; [test utterance]
```

The text in square brackets are for exposition and not included verbatim in the prompt.

C Supplementary results

Table 7 contains all results from using beam search, used to construct Tables 2, 3, and 4. Table 8 is a version of Table 5 with more rows.

D Speculative constrained decoding algorithm

To further expand on the description in Section 3.4, we express the speculative constrained decoding method in Python-like pseudocode in Listing 1.

				Accuracy	
Model	Output	Decoding	Beam size	Overnight Cal.	SMCalFlow
Davinci	Canonical	Constrained	5	0.81	0.340
Davinci	Canonical	Constrained	1	0.76	0.290
Davinci	Canonical	Unconstrained	5	0.72	0.295
Davinci	Canonical	Unconstrained	1	0.72	0.255
Davinci	Meaning	Constrained	5	0.68	0.245
Davinci	Meaning	Constrained	1	0.62	0.210
Davinci	Meaning	Unconstrained	5	0.53	0.230
Davinci	Meaning	Unconstrained	1	0.48	0.190
Curie	Canonical	Constrained	5	0.66	0.260
Curie	Canonical	Constrained	1	0.58	0.210
Curie	Canonical	Unconstrained	5	0.50	0.225
Curie	Canonical	Unconstrained	1	0.47	0.210
Curie	Meaning	Constrained	5	0.44	0.200
Curie	Meaning	Constrained	1	0.39	0.165
Curie	Meaning	Unconstrained	5	0.38	0.185
Curie	Meaning	Unconstrained	1	0.31	0.160
Davinci Codex	Canonical	Constrained	5	0.86	0.355
Davinci Codex	Canonical	Constrained	1	0.84	0.305
Davinci Codex	Canonical	Unconstrained	5	0.79	0.310
Davinci Codex	Canonical	Unconstrained	1	0.77	0.295
Davinci Codex	Meaning	Constrained	5	0.86	0.345
Davinci Codex	Meaning	Constrained	1	0.75	0.300
Davinci Codex	Meaning	Unconstrained	5	0.80	0.315
Davinci Codex	Meaning	Unconstrained	1	0.73	0.280
Cushman Codex	Canonical	Constrained	5	0.87	0.320
Cushman Codex	Canonical	Constrained	1	0.80	0.290
Cushman Codex	Canonical	Unconstrained	5	0.83	0.300
Cushman Codex	Canonical	Unconstrained	1	0.77	0.285
Cushman Codex	Meaning	Constrained	5	0.80	0.340
Cushman Codex	Meaning	Constrained	1	0.73	0.280
Cushman Codex	Meaning	Unconstrained	5	0.72	0.305
Cushman Codex	Meaning	Unconstrained	1	0.70	0.250

Table 7: All results on Overnight Calendar and SMCalFlow using beam search.

		Overnight Calendar				SMCalFlow			
		Accuracy		Items/second		Accuracy		Items/second	
Width	Temperature	Canonical	Meaning	Canonical	Meaning	Canonical	Meaning	Canonical	Meaning
1	0.0	0.86	0.76	0.520	0.246	0.300	0.320	0.193	0.184
1	BS	0.840	0.750	0.237	0.059	0.305	0.300	0.116	0.040
5	0.25	0.86	0.79	0.553	0.208	0.330	0.325	0.071	0.050
5	0.5	0.87	0.80	0.380	0.155	0.335	0.315	0.076	0.140
5	0.75	0.86	0.84	0.344	0.129	0.320	0.340	0.076	0.081
5	1.0	0.87	0.85	0.260	0.145	0.325	0.330	0.076	0.034
5	BS	0.860	0.860	0.133	0.030	0.355	0.345	0.065	0.008
10	0.25	0.88	0.81	0.537	0.213	0.345	0.310	0.020	0.040
10	0.5	0.87	0.86	0.355	0.150	0.345	0.345	0.038	0.085
10	0.75	0.87	0.82	0.266	0.103	0.350	0.355	0.039	0.034
10	1.0	0.87	0.85	0.193	0.068	0.370	0.335	0.028	0.014

Table 8: Comparing various settings on speculative decoding with beam search. “BS” for temperature indicates use of beam search. This table is an expanded version of Table 5

```
# Parameters:
# - W = width of the search
# - T = softmax temepature
# - MAX_STEPS = How many times we invoke the model. We set this to 16.
#
# Helper functions:
# - nextTokens: as defined in Section 2.1
# - model_completions: ask the model to generate completions with the given
#   prefix. Returns a list of token sequences sampled after the prefix.
# - length_normalized_logprob: compute the log probability of a token sequence,
#   where longer sequences receive a bonus.
# - is_finished: check if a token sequence is finished according to the grammar.
#
# `search` is invoked with tokens for the prompt p for a given example.

def expand(tokens):
    samples = model_completions(tokens, temperature=T, num_completions=W)

    results = []
    for sample in samples:
        valid_prefix = tokens
        for token in sample:
            if token not in nextTokens(prefix):
                break
            valid_prefix += [token]
        if valid_prefix == tokens:
            # No tokens in the completion were grammatically valid.
            # Back off to regular constrained decoding to advance by one token,
            # and append to results
            ...
        else:
            results += [valid_prefix]
    return results


def search(prompt):
    # We start with one hypothesis containing tokens from the prompt.
    beam = [prompt]
    finished = []

    for _ in range(MAX_STEPS):
        candidates = []
        for state in beam:
            candidates += expand(state)
        candidates.sort(key=length_normalized_logprob, reverse=True)

        new_beam = []
        for cand in candidates:
            if is_finished(cand):
                finished.append(cand)
            else:
                new_beam.append(cand)
            if len(finished) + len(new_beam) == W:
                break

        if len(new_beam) == 0:
            break
        else:
            beam = new_beam

    return finished
```

Listing 1: Pseudocode for speculative constrained decoding