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  
bevandur@microsoft.com

Abstract

Large language models, prompted with in-context examples, can perform semantic parsing with little training data. They do better when we formulate the problem as paraphrasing into canonical utterances, which cast the underlying meaning representations 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. More recently, models also pre-trained on code, like OpenAI Codex, have risen in prominence. Since accurately modeling code requires understanding of executable semantics, such models may prove more adept at semantic parsing. In this paper, we test this hypothesis and find that Codex performs better at semantic parsing than equivalent GPT-3 models. We find that unlike GPT-3, Codex performs similarly when targeting meaning representations directly, perhaps as meaning representations used in semantic parsing are structured similar to code.

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 pre-trained 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) 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 sub-language of English (canonical utterances) then translated the canonical utterance output into the meaning representation using a synchronous context-free grammar. In this work, we perform similar experiments on the Overnight (Wang et al., 2015) and SMCalFlow (Andreas et al., 2020) datasets, 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.

In this work, we find that:
• We can achieve higher accuracy when applying Codex in the same way as GPT-3 to perform semantic parsing.
• Codex significantly narrows the gap in accuracy between predicting meaning representations directly versus canonical utterances, obviating the need to define canonical utterances and create an SCFG.
• Even with Codex, constrained decoding and a non-greedy search procedure are still valuable in providing improved accuracy.
• Speculative constrained decoding, an adaptation of Anonymous (2022, Appendix F), gives comparable accuracy as beam search but with greater efficiency, on the language model APIs provided by OpenAI.

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 dy-
amically created prompts, (2) constrained decoding, and (3) optionally using a canonical utterance 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\(^1\) 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:

\[
\text{Let’s translate what a human user says into what a computer might say.}
\]

\[
\begin{align*}
\text{Human: when is the standup} & \leftarrow u_1 \\
\text{Computer: start time of "standup"} & \leftarrow c_1 \\
\text{Human: what date is the standup} & \leftarrow u_2 \\
\text{Computer: date of "standup"} & \leftarrow c_2 \\
\text{Human: when is the daily standup} & \leftarrow u
\end{align*}
\]

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. We assume the existence of a function \( \text{nextTokens}(s) = \{w_i\} \) which returns the set of subsequent tokens allowed by the grammar, for a given prefix \( s \). For example, \( \text{nextTokens(start time)} \) would contain of, but not EOS or in. We use \( \text{nextTokens} \) to filter candidates from the language model such that it only generates grammatical outputs.

### 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.

The primary use case for the API is generating completions from a prefix, by sequentially sampling from \( p(w_n|w_1w_2\cdots w_{n-1}) \) 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 \( p(w_n|w_1w_2\cdots w_{n-1}) \), but only returns probabilities for up to 100 most likely tokens; we use this capability for constrained beam search.

\(^1\)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.

### 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.

**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).

**Test set sampling.** 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.

### 3 Experiments

#### 3.1 Comparing GPT-3 and Codex

| 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 1: 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.

Table 1 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 significantly 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.

### 3.2 Targeting canonical utterances versus meaning representations

| Model        | Canonical | Meaning | C – M |
|--------------|-----------|---------|-------|
| Davinci      | 0.81      | 0.68    | 0.13  |
| Davinci Codex| 0.86      | 0.86    | 0.00  |

(a) Overnight Calendar

| Model        | Canonical | Meaning | C – M |
|--------------|-----------|---------|-------|
| Davinci      | 0.340     | 0.245   | 0.095 |
| Davinci Codex| 0.355     | 0.345   | 0.010 |

(b) SMCalFlow

Table 2: 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 2, 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. However, designing a suitable system of canonical utterances is non-trivial, where the smaller performance gap exhibited by Codex changes the cost/benefit calculations.

### 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 suggests using it that way to generate code from comments, a similar task to semantic parsing. Nevertheless, we see in Table 3 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.

### 3.4 Speculative constrained decoding

While constrained decoding and beam search improve accuracy, they are also slow to perform with OpenAI’s API. Extending a partial hypothesis requires one network round-trip per token. Also, the API lacks state and each request needs to include the prompt and all the 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 neural network hidden states for previous tokens are cached, retrieving them from the cache and transferring them to GPUs or other accelerators takes overhead.

As such, we adapt a method from Synchromesh (Anonymous, 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 soft-

### Table 3: 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 is important for attaining higher accuracy.

| Decoding       | Beam | Accuracy 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     |

\[\text{Table 3: Results comparing constrained with unconstrained decoding and multiple beam sizes, when generating meaning representations. Even when using Davinci Codex, trained specifically on code, constrained decoding and beam search is important for attaining higher accuracy.}\]

\[\text{As mentioned in Section 2.2, the primary capability of OpenAI’s API is generating completions from a prefix using sequential sampling. Their documentation suggests using it that way to generate code from comments, a similar task to semantic parsing. Nevertheless, we see in Table 3 that the use of constraints and beam search lead to benefits in accuracy. Even with constrained decoding, greedy argmax sampling (equivalent to a beam size of 1) performs worse than using beam search.}\]

\[\text{3.4 Speculative constrained decoding}\]

While constrained decoding and beam search improve accuracy, they are also slow to perform with OpenAI’s API. Extending a partial hypothesis requires one network round-trip per token. Also, the API lacks state and each request needs to include the prompt and all the 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 neural network hidden states for previous tokens are cached, retrieving them from the cache and transferring them to GPUs or other accelerators takes overhead.

As such, we adapt a method from Synchromesh (Anonymous, 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 soft-
max temperature $T$. 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.

Table 4 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 significantly faster.

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

### 3.5 Putting everything together

As described in Section 2.3, previous results in the paper were computed on smaller subsets of the evaluation sets due to resource limitations. In Table 5, 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 se-
mantic 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.

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 Ondru, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomian, 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.

Anonymous. 2022. Synchromesh: Reliable code generation from pre-trained language models. In Submitted to The Tenth International Conference on Learning Representations. Under review.

Jacob Austin, Augustus Odema, 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.

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, Wu, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alex Radford, Ilya Safro, and Dario Amodei. 2020. Language models are few-shot learners.

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 Cummins, Matthias Plappert, Fotos Chantzi, 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 Mehdiad, 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.

Paresh Kharya and Ali Alvi. 2021. Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, the World’s 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.

Jack Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susan 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, Safron Huang, Jonathan Usato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsa, Siddhant Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhigena Krusco, Aide Nezmatadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Laspeyres, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Manas Jajakras, Tobin Pohlen, Zhihao Gong, Daniel Toyma, Cyprien de Masson d’Autume, Yujia Li, Tayfun Terzi, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, James Bradbury, Matthew Johnson, Laura Weidinger, Jason Gabriel, William Isaac, Ed Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stannaway, Lorrane Bennett, Denis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2021. Scaling language models: Methods, analysis & insights from training Gopher. arXiv submission.
For measuring the items/second of beam search and speculative constrained decoding in Table 4 and Table 7, 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 6 contains all results from using beam search, used to construct Tables 1, 2, and 3. Table 7 is a version of Table 4 with more rows.
### Table 6: All results on Overnight Calendar and SMCalFlow using beam search.

| 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          | 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            | 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    | 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    | 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: 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 4.

| Width | Temperature | Accuracy Items/second | Overnight Calendar | Accuracy Items/second | SMCalFlow | Accuracy Items/second |
|-------|-------------|------------------------|---------------------|------------------------|-----------|-----------------------|
|       | 0.0         | 0.86 0.76 0.520 0.246 | 0.300 0.320 0.193 0.184 | 0.840 0.750 0.237 0.059 | 0.305 0.300 0.116 0.040 | 0.860 0.860 0.133 0.030 |
| 1     | BS          | 0.86 0.79 0.553 0.208 | 0.330 0.325 0.071 0.050 | 0.87 0.80 0.380 0.155 | 0.335 0.315 0.076 0.140 | 0.860 0.860 0.133 0.030 |
| 5     | 0.25        | 0.87 0.87 0.344 0.129 | 0.320 0.340 0.076 0.081 | 0.87 0.85 0.260 0.145 | 0.325 0.330 0.076 0.034 | 0.860 0.860 0.133 0.030 |
| 5     | 0.5         | 0.87 0.87 0.344 0.129 | 0.320 0.340 0.076 0.081 | 0.87 0.85 0.260 0.145 | 0.325 0.330 0.076 0.034 | 0.860 0.860 0.133 0.030 |
| 5     | 1.0         | 0.87 0.87 0.344 0.129 | 0.320 0.340 0.076 0.081 | 0.87 0.85 0.260 0.145 | 0.325 0.330 0.076 0.034 | 0.860 0.860 0.133 0.030 |
| 5     | BS          | 0.860 0.860 0.133 0.030 | 0.355 0.345 0.065 0.008 | 0.87 0.85 0.260 0.145 | 0.325 0.330 0.076 0.034 | 0.860 0.860 0.133 0.030 |
| 10    | 0.25        | 0.87 0.85 0.193 0.068 | 0.370 0.335 0.028 0.014 | 0.87 0.85 0.193 0.068 | 0.370 0.335 0.028 0.014 | 0.860 0.860 0.133 0.030 |
| 10    | 0.5         | 0.87 0.85 0.193 0.068 | 0.370 0.335 0.028 0.014 | 0.87 0.85 0.193 0.068 | 0.370 0.335 0.028 0.014 | 0.860 0.860 0.133 0.030 |
| 10    | 1.0         | 0.87 0.85 0.193 0.068 | 0.370 0.335 0.028 0.014 | 0.87 0.85 0.193 0.068 | 0.370 0.335 0.028 0.014 | 0.860 0.860 0.133 0.030 |