SpreadNaLa: A Naturalistic Code Generation Evaluation Dataset of Spreadsheet Formulas

Sebastian Schuster,[◇,○] **Ayesha Ansar**[○], **Om Agarwal**[*], **Vera Demberg**[○]
[◇] University College London [○]Saarland University [*] Northeastern University
s.schuster@ucl.ac.uk, vera@lst.uni-saarland.de

Abstract

Automatic generation of code from natural language descriptions has emerged as one of the main use cases of large language models (LLMs). This has also led to a proliferation of datasets to track progress in the reliability of code generation models, including domains such as programming challenges and common data science tasks. However, existing datasets primarily target the use of code generation models to aid expert programmers in writing code. In this work, we consider a domain of code generation which is more frequently used by users without sophisticated programming skills: translating English descriptions to spreadsheet formulas that can be used to do everyday data processing tasks. We extract naturalistic instructions from StackOverflow posts and manually verify and standardize the corresponding spreadsheet formulas. We use this dataset to evaluate an off-the-shelf code generation model (GPT 3.5 text-davinci-003) as well as recently proposed pragmatic code generation procedures and find that Code Reviewer reranking (Zhang et al., 2022) performs best among the evaluated methods but still frequently generates formulas that differ from human-authored ones.

Keywords: code generation, spreadsheet formulas, dataset, English

1. Introduction

One of the common use cases of large language models (LLMs) and LLM-based chatbots is the automatic generation of code from natural language descriptions (Chen et al., 2021a), and there is some preliminary evidence that LLMs have considerable positive effects on the productivity of expert programmers (Kreitmeir and Raschky, 2023). However, also users who do not have any or much expertise in programming sometimes automate small data processing tasks, as for example, when using formulas in spreadsheet applications such as Microsoft Excel or Google Spreadsheets. Such applications provide a very low barrier of entry to automate simple tasks with short commands such as `SUM` or `AVERAGE` to compute aggregate statistics. At the same time, the underlying interpreter supports a quite expressive language that allows expert spreadsheet users to perform relatively complex tasks. But also in this environment, only users with considerable experience are able to write spreadsheet formulas that can perform more complex data processing tasks. This would be true to a much lesser extent if users could describe their data processing needs in natural language but it remains an open question to what extent current models are capable of translating natural language descriptions to spreadsheet formulas.

In this work, we therefore introduce SpreadNaLa, a naturalistic code generation evaluation dataset of **spread**sheet formulas with **na**tural **la**nguage descriptions. We compile this dataset from English question-answer pairs on the programming help community StackOverflow,[1] and manually correct and harmonize the spreadsheet formulas. We then use this dataset to evaluate GPT-3.5 (Ouyang et al., 2022) as well as two reranking methods on their abilities to automatically generate spreadsheet formulas. We find that the combination of GPT-3.5 together with the recently proposed Code Review reranking method (Zhang et al., 2022) is more reliable at producing well-formed and correct spreadsheet formulas than GPT 3.5 by itself. At the same time, all evaluated methods produce formulas that considerably deviate from gold formulas for the majority of questions, suggesting that generating more complex spreadsheet formulas still poses a challenge for even very recent LLMs. We release the SpreadNaLa dataset, an evaluation script, and the experiment code at `https://github.com/sebschu/SpreadNaLa`.

2. Related Work

Translating natural language expressions to executable code or logical forms has been a long-standing research objective in natural language processing research leading to the development of many semantic parsers. Early systems generally output specific logical forms or database queries to interact with knowledge bases or other external systems (e.g., Zelle and Mooney, 1996; Zettlemoyer and Collins, 2007; Liang et al., 2013; Berant et al., 2013; Reddy et al., 2014; Yu et al., 2018). More recently, there have been more general code generation benchmarks that require the model to

[1] https://stackoverflow.com

LREC-COLING 2024, pages 15216–15225
20-25 May, 2024.

# Examples	1,300
Description length (words)	97.8 (42–182)
Formula length (tokens)	21.8 (5-55)

Table 1: Dataset statistics. Length statistics show the means and (in parentheses) the interquantile range between the 5th and 95th percentile. The formula length was measured in tokens according to the spreadsheet formula tokenizer that we used for evaluation (see Section 4.2).

output much longer programs in open domain programming languages such as Java or Python (e.g., Agashe et al., 2019). Most of these benchmarks are based on programming interview or challenge problems (e.g., Hendrycks et al., 2021; Li et al., 2022) and therefore do not really measure everyday programming tasks. In the area of everyday programming tasks, there has been some work for building systems for automatic code documentation (e.g., Liu et al., 2021) and more interactive human-machine collaboration on programming tasks (e.g, Li et al., 2023).

More closely related to our work are the CoNaLa (Yin et al., 2018) and DS-1000 (Lai et al., 2023) datasets whose examples are based on naturalistic questions posted on StackOverflow. However, unlike our work, they focus on general Python programming and common data science problems. Lastly, there also exist other single-line code generation datasets such as NL2Bash (Lin et al., 2018) that can be used to evaluate systems translating from natural language to Bash commands.

To the best of our knowledge, there is no existing dataset for evaluating translation of natural language descriptions to spreadsheet formulas. There have been, however, several works developing datasets and methods for inferring spreadsheet formulas from examples (Gulwani, 2011; Gulwani et al., 2012; Chen et al., 2021b). Furthermore, Gulwani and Marron (2014) collected a dataset of natural language descriptions translated to a domain-specific formal language for spreadsheet manipulations and showed that a semantic parser can translate these descriptions into the formal language with high accuracy. However, the expressivity of their language is more limited than the language of spreadsheet formulas that we are considering.

3. Dataset

For collecting SpreadNaLa, we use a data collection protocol inspired by Yin et al. (2018). We scraped questions and answers from StackOverflow, then automatically and manually filtered the questions, and then manually corrected and harmonized the spreadsheet formulas.

3.1. Scraping and filtering

We automatically scraped all questions and associated answers tagged with "excel-formula" using the StackOverflow API in November 2022, resulting in a dataset of 14,294 questions. This included questions without answers, questions that were actually on another topic, and other unsuitable data points. We therefore performed several relatively aggressive filtering steps with the goal of curating a high quality evaluation dataset.

We automatically removed questions with an embedded image (which was often a screenshot of a spreadsheet indicating the data format) or a table since unimodal language models would not be able to parse that information; we removed questions without a "verified" answer, i.e., an answer that the question author marked as resolving their problem; and we removed questions whose verified answer had more downvotes than upvotes. Finally, since in most cases the spreadsheet formula for solving the problem was embedded with a `<code>` HTML block, we also excluded answers without such a block. After these automatic exclusions, we were left with 5,196 question-answer pairs.

3.2. Expert corrections

For the remaining examples, we extracted the title and body from each question to be used as the description, and extracted the contents of the first code block in the answer as the target formula. We then performed two rounds of manual annotations: First, one of the authors manually inspected every question and discarded examples that were not asking for a spreadsheet formula or were not self-contained (e.g., linked to external web pages). They also removed all examples where the answer did not provide a spreadsheet formula, but some other solution (e.g., a Visual Basic for Application (VBA) code snippet). Second, another annotator with extensive experience in spreadsheet formulas checked and corrected all formulas, and performed further exclusions. This resulted in the final dataset of 1,300 description-formula pairs. See Table 1 for more aggregate statistics of our dataset.

Spreadsheet formulas are generally used to process information from specific cells or ranges of cells and often take additional arguments, such as string constants, numbers, or conditional statements. However, in many cases, the description lacks information on the specific arguments and hence the answers may contain placeholders or example cell references that the answer writer chose to illustrate their solution. Considering that this information is missing from the description, neither a model nor a human could reliably reproduce the solution of the answer writer. We therefore replaced specific instances (e.g., a reference to cell `A1`) with

variables such as <cell1> or <string1> if they were not mentioned in the description. Our evaluation metrics (see Section 4.2) consider these variables such that models are not penalized for generating placeholders for references and arguments that were not specified in the description.

4. Models and Experiments

We use our dataset to evaluate three models based on GPT-3.5 text-davinci-003 (Ouyang et al., 2022): a baseline model without additional processing, the recently proposed Code Reviewer reranking approach (Zhang et al., 2022), and a model within the Rational Speech Act framework (RSA; Goodman and Frank, 2016) intended to better capture the true intention of the user. For all models, the goal is to infer a spreadsheet formula f from a natural language description d.

4.1. Models

Baseline. As a baseline model, we use GPT-3.5 text-davinci-003 together with a fixed prompt template (see Appendix A). This model is assumed to have 175B parameters, was pre-trained on large amounts of text and code, and was further finetuned using a proprietary instruction finetuning dataset and a reinforcement learning from human feedback procedure.

We use a prompt that instructs the model to generate a spreadsheet formula f based on a description d. To illustrate the format, we provide one example description and the corresponding formula as a 1-shot in-context learning example. We manually created this example and it is not part of the evaluation dataset. We take the raw generated output as the model's predicted formula. See Appendix B for the hyperparameters of the generation process.

Code Reviewer. One challenge with code generation, especially from descriptions written by non-expert programmers, is that some requirements are not explicitly stated. We therefore also evaluated two recently proposed models that aim to better infer the true intention of the user. The first model is the Code Reviewer reranking model (Zhang et al., 2022). For this model, we sample 10 completions (i.e., formulas $f_i \in F$) from GPT-3.5 for each description d and compute the probability $P(f_i \mid d)$ from the probabilities of each token $t_1, ..., t_N$ of f_i,

$$P(f_i \mid d) = \prod_{j=1}^{N} GPT(t_j \mid d, t_{1,...,j-1}).$$

We then compute the likelihood of generating the original description d given each formula f_i. We again use GPT-3.5 to approximate this probability by using a prompting template that instructs the model to generate a description of a formula (see Appendix A). Given a formula f_i and the tokens of the description $u_1, ..., u_M$, we compute the likelihood

$$P(d \mid f_i) = \prod_{j=1}^{M} GPT(u_j \mid f_i, u_{1,...,j-1}).$$

Note that we are using GPT-3.5 here to only score the description; we do not sample any new tokens from the model when computing the likelihood. To compute the final prediction, the model returns the formula f that maximizes the product of the two computed quantities:

$$f = \arg\max_{f_i} P(f_i \mid d) \times P(d \mid f_i).$$

RSA. We also evaluated a model that is based on the Rational Speech Act (RSA) framework. RSA models have been used to successfully model the interpretation of pragmatic language use which–like descriptions of intended code behavior–is often underspecified and involves ambiguities. Furthermore, in a toy domain, an RSA code generation model has been shown to allow users to generate programs more efficiently (Pu et al., 2020). The core idea of this model is similar to the Code Reviewer model, such that the model not only considers the probability of a formula given the description but also how likely someone would be to describe a formula with the given description. RSA models achieve this by explicitly modeling a recursive reasoning process where an interpreter $L_1(f_i \mid d_k)$ (called listener in RSA terms) reasons about a speaker model $S_1(d_k \mid f_i)$ which in return reasons about another listener $L_0(f_i \mid d_k)$:

$$L_1(f_i \mid d_k) = \frac{S_1(d_k \mid f_i)}{\sum_{f_j \in F} S_1(d_k \mid f_j)}$$

$$S_1(d_k \mid f_i) = \frac{L_0(f_i \mid d_k)}{\sum_{d_l \in D} L_0(f_i \mid d_l)}$$

The lowest level of this recursive reasoning process, the $L_0(f_i \mid d_k)$ listener, is approximated by the GPT-3.5 model:

$$L_0(f_i \mid d_k) \propto GPT(f_i \mid d_k)$$

One challenge with RSA models is the normalization in each speaker and listener model which requires a distribution over every possible formula f_j and every possible description d_l. Given that there are both infinitely many valid spreadsheet formulas

Model	⟲ Edit Distance ↓	Exact Match ↑	Invalid ↓
GPT3.5 text-davinci-003	16.93	5.77%	0.77%
GPT3.5 text-davinci-003 + Code Reviewer	**15.85**	**5.85**%	**0.69**%
GPT3.5 text-davinci-003 + RSA	17.95	3.15%	1.62%

Table 2: Performance of the baseline model and the two reranking methods on the SpreadNaLa dataset.

and infinitely many possible natural language descriptions, it is impossible to compute these distributions. We therefore sample a finite set of formulas F and a finite set of descriptions D from GPT-3.5 and perform the RSA computations based on distributions over these finite sets. See Appendix C for the specifics of this approximation.

The main motivation of the Code Reviewer model and the RSA model is the same, namely to also consider how likely someone would describe a candidate formula with the given description, based on the assumption that the higher this likelihood is, the better the candidate formula matches the description. The difference between these two models, though, is that the RSA model also considers alternative descriptions d_k that could have been used to describe the other candidate formulas.

4.2. Evaluation

For evaluating the model output, we lowercase and tokenize both the model output and the gold formula using the spreadsheet formula tokenizer from the Python openpyxl package.[2] We then compare the list of tokens from the model output and the gold formula and compute a) the exact match, i.e., whether the list of tokens is the same and b) the Levenshtein edit distance between the two lists. Further, as mentioned above, our gold formulas often contain placeholders for cell references or arguments that have not been specified in the description. In the evaluation procedure, we therefore allow placeholders to align with any token in the model output. Lastly, we also compute the percentage of invalid formulas, i.e., formulas that are not well-formed due to unmatched parentheses or quotation marks or other syntactic issues.

4.3. Results and Discussion

Table 2 shows the results for the evaluated models. We find that the baseline GPT-3.5 model produces syntactically well-formed formulas almost all the time and fewer than 1% of the formulas are invalid. At the same time, the percentage of examples for which the model output exactly the gold formula is still low and the predicted and gold formula differ on average by almost 17 tokens. This number is particularly high when put in relation to the average length of a formula of 21.76 tokens in our dataset.

Combining the baseline model with Code Reviewer reranking brings small but consistent gains across all metrics, which suggests that considering how well the description fits the generated formula brings additional advantages. Given this finding, it is a bit surprising that the RSA model which provides a more sophisticated method for assessing the fit of a formula and a description performs the worst of all the three models. One reason may be that for the RSA model, we also sample descriptions from GPT-3.5 and these descriptions ended up being different in style from the original descriptions that were derived from StackOverflow questions. Therefore, these alternative descriptions may not really be competing descriptions and may sometimes lead the model astray.

Qualitative analysis. To assess what kind of errors the model makes and whether such a model may be useful in practice, we sampled 100 examples from the dataset and also performed a manual analysis of the output of the three models on these examples. For this purpose, a research assistant (RA) noted whether a formula was an exact match modulo some minor syntactic differences (**EM**; e.g., using semicolons instead of commas to separate arguments); whether the model output a correct formula that differed from the gold formula in our dataset (**Co**); whether the model output a non-general solution that produces the correct output for the examples that the user specified in the description but would not necessarily generalize to other examples (**NGS**, e.g., because some values are hard-coded instead of performing a lookup), or whether it was incorrect (**Inc**). The RA further annotated whether what we considered the most important function was part of the model's solution (**MIF**)[3]; whether the formula was well-formed and each function had the correct number of arguments (**WF**); and whether the arguments of all functions were correct (**CA**). Note that while the first set of annotations is mutually exclusive, the last three categories can be simultaneously true. To avoid any bias, one of the authors prepared the model output

[2] https://openpyxl.readthedocs.io

[3] We included this metric, which was suggested by an anonymous reviewer, since identifying the correct function may already be an important step in helping a user find a solution.

Model	EM	Co	NGS	Inc	MIF	WF	CA
GPT3.5 text-davinci-003	12%	24%	12%	52%	**57%**	82%	55%
GPT3.5 text-davinci-003 + Code Reviewer	**14%**	**25%**	**13%**	**48%**	54%	**86%**	**58%**
GPT3.5 text-davinci-003 + RSA	8%	19%	10%	63%	51%	78%	44%

Table 3: Results from manual analysis of 100 sampled outputs from each model. **EM**: Exact match, **Co**: Correct but different from gold formula, **NGS**: Non-general solution, **Inc**: Incorrect, **MIF**: contains most important function, **WF**: Syntactically well-formed formula, **CA**: Correct arguments. Note that **EM**, **Co**, **NGS**, and **Inc** are mutually exclusive categories, whereas **MIF**, **WF** and **CA** can all simultaneously hold for a given formula.

sample and the RA did not know which model had generated the examples.

Table 3 shows the results from this manual analysis, which confirms the general trends that we found with the automatic analysis: The Code Reviewer reranking method outperformed the other two methods according to every metric other than whether it identified the most important function, and the RSA method generally yielded lower results than both the baseline and the Code Reviewer reranking method. This manual analysis also suggests that the automatic exact match metric underestimates the usefulness of the model: For 29% of the examples, the Code Reviewer reranking method output a formula that either closely followed the gold formula or was a correct alternative solution and in additional 13% of the cases, it produced a solution that worked for the example the the user specified. At the same time, well-formedness was lower for the models than the low rate of invalid formulas according to the syntactic parser may suggest, and in the sample of outputs that we analyzed the model still frequently output incorrect cell references, hallucinated arguments, or oversimplified formulas compared to the gold formulas. These findings show that while the model seems to perform better than the automatic metrics suggest, there is still significant room for improvement and current models are likely not reliable enough to enable non-expert spreadsheet users to perform sophisticated data processing tasks.

5. Conclusion

In this work we presented SpreadNaLa, a novel dataset for evaluating the ability of code generation models to translate natural language descriptions to spreadsheet formulas. We collected this dataset from naturalistic questions on StackOverflow and harmonized and corrected the target output. We used SpreadNaLa to evaluate several code generation models based on GPT-3.5 with a focus on evaluating pragmatic code generation methods that have been shown to better handle information that is not made explicit in natural language. While we found that the Code Reviewer reranking method improved over a GPT-3.5 baseline, the more complex RSA model did not lead to improvements, which appears to be in part caused by non-optimal alternative descriptions that were generated by GPT-3.5.

Considering that none of the models performed close to ceiling, SpreadNaLa constitutes a benchmark well-suited for tracking the progress of future code generation models and methods.

6. Limitations

One limitation of SpreadNaLa is that it only supports string-based evaluation metrics. As rightfully pointed out by Lai et al. (2023), string-based metrics penalize alternative solutions that also lead to the correct result but nevertheless differ from the gold formula. This limitation stems from the fact that it is not possible to interpret spreadsheet formulas outside of spreadsheet applications, and thus it was not possible to design an execution-based evaluation.

In our evaluations, we used only one prompt template to generate alternative descriptions for the RSA model and it could be that other prompts would have produced better alternative descriptions which in return could have led to better performance of the RSA reranking model. Unfortunately, the generation of many descriptions for each examples requires generating many tokens and it would have been prohibitively expensive to optimize the template using a commercial model such as GPT-3.5.

Furthermore, we only used the fully closed model GPT-3.5 and we do not know whether the model was trained on some or all of the StackOverflow webpages from which we scraped the descriptions and formulas. Considering that all evaluated models rarely produced exactly the gold formula, it seems unlikely that the model memorized all answers but it could be that in practice the model performs even worse. With the recent releases of more oopen models such as Pythia (Biderman et al., 2023) and Code LLama (Rozière et al., 2023), it will therefore be important to also evaluate models for which more information on their training data is available, also in light of the deprecation of GPT-3.5 shortly after the submission of this paper.

Acknowledgments

We thank the anonymous reviewers for their thoughtful feedback, and Zhuchen Cao for his help with analyzing the model outputs. This research was supported by the the European Research Council (ERC) under the European Union's Horizon 2020 Research and Innovation Program (Grant Agreement #948878).

7. Bibliographical References

Rajas Agashe, Srinivasan Iyer, and Luke Zettlemoyer. 2019. JuICe: A large scale distantly supervised dataset for open domain context-based code generation. 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 5436–5446, Hong Kong, China. Association for Computational Linguistics.

Jacob Andreas and Dan Klein. 2016. Reasoning about pragmatics with neural listeners and speakers. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182, Austin, Texas. Association for Computational Linguistics.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, Usvsn Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, pages 2397–2430.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri 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, Josh 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. 2021a. Evaluating large language models trained on code. arXiv:2107.03374.

Xinyun Chen, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. 2021b. Spreadsheetcoder: Formula prediction from semi-structured context. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1661–1672.

Daniel Fried, Jacob Andreas, and Dan Klein. 2018. Unified pragmatic models for generating and following instructions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1951–1963, New Orleans, Louisiana. Association for Computational Linguistics.

Noah D. Goodman and Michael C. Frank. 2016. Pragmatic Language Interpretation as Probabilistic Inference. *Trends in Cognitive Sciences*, 20(11):818–829.

Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, page 317–330.

Sumit Gulwani, William R. Harris, and Rishabh Singh. 2012. Spreadsheet data manipulation using examples. *Commun. ACM*, 55(8):97–105.

Sumit Gulwani and Mark Marron. 2014. NLyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, page 803–814, New York, NY, USA. Association for Computing Machinery.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with APPS. In *Thirty-fifth Conference on Neural Information Process-*

ing Systems Datasets and Benchmarks Track (Round 2).

David H. Kreitmeir and Paul A. Raschky. 2023. The unintended consequences of censoring digital technology – evidence from italy's chatgpt ban. arXiv:2304.09339.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-Tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. DS-1000: A natural and reliable benchmark for data science code generation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 18319–18345. PMLR.

Haau-Sing (Xiaocheng) Li, Mohsen Mesgar, André Martins, and Iryna Gurevych. 2023. Python code generation by asking clarification questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14287–14306, Toronto, Canada. Association for Computational Linguistics.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097.

Percy Liang, Michael I. Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. 2018. NL2Bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Xuye Liu, Dakuo Wang, April Wang, Yufang Hou, and Lingfei Wu. 2021. HAConvGNN: Hierarchical attention based convolutional graph neural network for code documentation generation in Jupyter notebooks. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4473–4485, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155.

Yewen Pu, Kevin Ellis, Marta Kryven, Josh Tenenbaum, and Armando Solar-Lezama. 2020. Program synthesis with pragmatic communication. In *Advances in Neural Information Processing Systems*, volume 33, pages 13249–13259.

Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. arXiv:2308.12950.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from Stack Overflow. In *Proceedings of the 15th International Conference on Mining Software Repositories*, page 476–486.

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

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*, pages 1050–1055.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687, Prague, Czech Republic. Association for Computational Linguistics.

Tianyi Zhang, Tao Yu, Tatsunori B. Hashimoto, Mike Lewis, Wen tau Yih, Daniel Fried, and Sida I.

Wang. 2022. Coder reviewer reranking for code generation. arXiv:2211.16490.

A. Prompt templates

Tables 4 and 5 show the prompt templates that we used to generate the most likely formula in the baseline model, and sample formulas and descriptions in the reranking models. We further used the template in Table 5 for computing the likelihood of a description d given a formula f, $P(d \mid f)$.

B. Hyperparameters

Table 6 shows the set of hyperparameters for the baseline model, the formula generation model, and the description generation model. We used the same hyperparameters for the three models except for the temperature parameter, which guides the diversity of generations. For the baseline model, where we generate only one formula, we set the temperature to 0 such that the model uses a greedy generation process that always returns the most probable token. For the formula generation and description generation models used in the Code Review and RSA reranking models, we need a diverse set of formulas and descriptions, and therefore, we set the temperature parameter to higher values of 0.8 and 1.0, respectively.

C. Approximate RSA model

As mentioned in the main text, it would be intractable to exactly compute the distributions for the pragmatic speaker S_1 and the pragmatic listener L_1, since we would have to normalize over an infinite amount of possible descrtiptions and formulas. We therefore approximate these distributions similarly to other models combining RSA with language models (e.g., Andreas and Klein, 2016; Fried et al., 2018). For this approxmation, we sample both up to 10 formulas and up to 10 different descriptions from GPT-3.5. To obtain the set of formulas F, we sample 10 formulas f_i given the prompt used for the baseline. As in some cases, some of the samples only differed in their casing, we de-duplicate this set such that it includes no two formulas that only differ in their casing. For some prompts, this resulted in a set F that consists of fewer than 10 formulas. To obtain the set of descriptions D, we sample a description d_k from GPT-3.5 for each $f_i \in F$ using the prompt template in Table 5. Using these sets F and D, we then approximate the components of the RSA model as follows.

$$L_0(f_i \mid d_k) = \frac{GPT(f_i \mid d_k)}{\sum_{f_j \in F} GPT(f_j \mid d_k)}$$

$$S_1(d_k \mid f_i) = \frac{L_0(f_i \mid d_k)}{\sum_{d_l \in D} L_0(f_i \mid d_l)}$$

$$L_1(f_i \mid d_k) = \frac{S_1(d_k \mid f_i)}{\sum_{f_j \in F} S_1(d_k \mid f_j)}$$

To compute $GPT(f_i \mid d_k)$ we again use the prompt template in Table 4 but this time compute the likelihood of f_i given the prompt instead of sampling from the language model.

1-shot prompt used to compute $P(f \mid d)$

Translate the following descriptions into a spreadsheet formula.

Description: Write a spreadsheet formula to select the first letter from cell A4 and last letter from cell B6.
Formula: concat(left(a4,1),right(b6,1))

Description: `{description}`
Formula:

Table 4: Baseline/formula generation prompt template. We also use this template to compute the likelihood of a formula given a description.

1-shot prompt used to compute $P(d \mid f)$

Translate the following spreadsheet formula into a description.

Formula:concat(left(a4,1),right(b6,1))
Description: Write a spreadsheet formula to select the first letter from cell A4 and last letter from cell B6.

Formula: `{formula}`
Description:

Table 5: Description generation prompt template.

Parameter	Baseline	Formula Generation	Description Generation
Temperature	0	0.8	1.0
Max tokens	256	256	256
top-p	1.0	1.0	1.0
Frequency penalty	0	0	0
Presence penalty	0	0	0

Table 6: OpenAI API hyperparameters used for sampling formulas and descriptions from the baseline model, the formula generation model, and the description generation model.