Fine-Tuning Large Language Models for Answering Programming Questions with Code Snippets

Vadim Lomshakov[1][0000−0001−8991−9264], Sergey Kovalchuk[1][0000−0001−8828−4615], Maxim Omelchenko[1], Sergey Nikolenko[2,3][0000−0001−7787−2251], and Artem Aliev[1]

[1] Huawei, St. Petersburg, Russia
[2] AI Center, National University of Science and Technology MISIS, Moscow, Russia
[3] St. Petersburg Department of the Steklov Institute of Mathematics, St. Petersburg, Russia {vadim.lomshakov,sergey.kovalchuk}@huawei.com, maxim.omelchenko@huawei-partners.com,sergey@logic.pdmi.ras.ru, artem.aliev@huawei.com

Abstract. We study the ability of pretrained large language models (LLM) to answer questions from online question answering fora such as *Stack Overflow*. We consider question-answer pairs where the main part of the answer consists of source code. On two benchmark datasets—CoNaLa and a newly collected dataset based on *Stack Overflow*—we investigate how a closed-book question answering system can be improved by fine-tuning the LLM for the downstream task, prompt engineering, and data preprocessing. We use publicly available autoregressive language models such as GPT-Neo, CodeGen, and PanGu-Coder, and after the proposed fine-tuning achieve a BLEU score of 0.4432 on the CoNaLa test set, significantly exceeding previous state of the art for this task.

Keywords: Program synthesis · Question answering · Large language models.

1 Introduction

Modern natural language processing (NLP) widely employs large language models (LLMs) that extract knowledge from text implicitly, via pretraining, and then can perform, e.g., open domain question answering (QA) without access to any external context or knowledge [13, 18, 20]. These LLMs provide an alternative way to the design of QA systems, without an external knowledge base and explicit retrieval components; fine-tuning a pretrained LLM with (question, answer) pairs is usually sufficient to train it for a given domain. For very large models such as GPT-3 [4], this approach may even lead to results competitive with retrieval-based methods on open domain QA without any fine-tuning. The pretraining procedure is always the most important part, and the data and process of pretraining directly affects the quality shown on downstream tasks [25].

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_15

Recent success of *Codex* [5] in program synthesis has demonstrated that pretrained LLMs can be successfully adapted from NLP to the domain of source code. *Codex* is a GPT language model fine-tuned on a large corpus of publicly available code from *GitHub*; it powers *GitHub CoPilot* [5] that allows programmers to generate code in an IDE from a natural language query. Previously, the TranX model attempted to learn a neural semantic parser from scratch [27], but latest works have shown that using pretrained large language models (LLM) as part of the architecture works better. This includes the recently developed BERT+TAE [17], TranX+BERT [1], and MarianCG [21] models, the latter is a pretrained neural machine translation model (MarianMT) fine-tuned on code.

Program synthesis focuses on correctly implementing some functionality defined with a natural language query; researchers often use competitive programming problems to evaluate program synthesis models [5,11,14]. However, empirical studies of programming-related QA websites such as *Stack Overflow*[4] have shown that real life questions of programmers are not limited to defining a functionality. For instance, the work [2] classifies all posts into different categories such as "Conceptual" (*Why...? Is it possible...? Why something works?*"), "API usage" (*How to implement something? Way of using something?*"), "Discrepancy" (*Does not work*, *What is the problem...?*), and others, proposing regular expressions to define these categories. This gap leads to our research question: how effective are pretrained LLMs in answering the questions of real life programmers, even if we focus on questions answered with code snippets?

In this work, we use several publicly available GPT-based LLMs pretrained on code as backbones for solving closed-book QA in the *Stack Overflow* domain, evaluating on the *CoNaLa* dataset [26] and our own QA dataset collected from *Stack Overflow*. We introduce several approaches for fine-tuning, prompt engineering, and data preprocessing, achieving new state of the art results on *CoNaLa*. Below, Section 2 introduces the data, Section 3 outlines our approach, Section 4 discusses the evaluation study, and Section 5 concludes the paper.

2 Dataset

We consider programming-related QA with short code snippets generated as answers to real world problems. Thus, we focus on data with the following properties: (i) "API usage" questions according to the taxonomy shown in [2]; (ii) questions that consist of a short textual description (≤ 200 characters) without explicit source code in them; (iii) answers with explicit code snippets giving a solution to the proposed problem; (iv) to be more focused, we have limited our study to *Python* as one of the most popular programming languages.

First, we use the existing publicly available dataset *CoNaLa*[5] that satisfies our requirements [26]. The dataset consists of 2 879 examples, 2 379 in the training set and 500 in the test set, crawled from *Stack Overflow* and then manually curated by human annotators. Second, we have prepared our own dataset

[4] https://stackoverflow.com/
[5] https://conala-corpus.github.io/

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_15

based on original publicly available *Stack Overflow* data[6]. We have selected questions with the tag "*Python*" and used only the title text as the question. As ground truth, we have selected answers that earned maximum scores according to *Stack Overflow* data. To filter code-containing questions we search the text for `<pre><code>` in HTML and select questions with no explicit code paired with answers with a single code snippet. Finally, we have used the regular expressions proposed in [2] to select the "API usage" category of questions. We have cleaned the code by removing comments and selecting only snippets with correct syntax (no parsing errors). After these steps, we obtained a dataset with 10 522 question-answer pairs. We set aside 1 000 entries for the test set, using only questions from 2021 and later to avoid possible data leaks since we use LLMs trained on publicly available data. Since we use only the titles of *Stack Overflow* posts, the questions are short: 90% of the questions in the final dataset have between 5 and 17 words.

3 Methods

3.1 Large language models

As backbones, we have selected several LLMs for our study that are (i) publicly available, (ii) computationally inexpensive, and (iii) pretrained on code. Specifically, we have used: (i) CoPilot [5], an industrial solution based on *Codex*; (ii) GPT-Neo-2.7B (GPT-Neo below) [3] that shows high performance compared to *Codex* [24]; (iii) CodeGen-mono-2B (CodeGen) [16] that was trained on the *Pile* dataset [9] and separately on the code from *BigQuery* and *BigPython*; (iv) PanGu-Coder-2.6B (PanGu-Coder) [6] that was pretrained on *GitHub* code with a combination of autoregressive (causal) and masked language modeling losses in two stages, with the second stage using paired natural language and source code data. Both PanGu-Coder and CodeGen have equivalent or better performance on the *HumanEval* dataset than similarly sized Codex models [5].

3.2 Fine-tuning, prompt engineering, and data preprocessing

We propose several techniques for solving our particular QA problem for short text questions answered by code snippets. First, we fine-tune the selected pretrained models on training sets from both *CoNaLa* (denoted as FT:C in Table 1) and *StackOverflow* (FT:SO in Table 1). For fine-tuning, we used the AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, with no weight decay, learning rate 5e-06 with linear decay, batch size 40, and half-precision (fp16).

Second, we experimented with various prompt engineering techniques to wrap the question into a context better suited for a specific LLM. For CodeGen and GPT-Neo models, the best prompt has turned out to be simply wrapping the question into a multiline comment: `"""question""" answer` during training and

[6] https://data.stackexchange.com/

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_15

`"""question"""` on inference. For PanGu-Coder, we have used the prompt format used in its pretraining: `<comments> question <python> answer <eoc>`. We have also experimented with prefix tuning from the *OpenPrompt* library [7] but it has not led to improvements on our datasets.

Third, we have found that adding domain-specific knowledge such as the names of code libraries can dramatically improve the quality of generated answers. For that purpose, we have trained a classifier on *StackOverflow* data to predict the usage of *Python* libraries. Specifically, we selected 317 300 posts (this set does not intersect with either *CoNaLa* or *StackOverflow* datasets) with only one `import` statement in the best answer, selected top 200 most used libraries as classifier labels, used a pretrained MPNet-base model [22] to encode the titles of these posts, and trained a support vector machine (SVM) for classification with these embeddings as input and the corresponding libraries as output. The resulting classifier obtains 0.47 Prec@1, 0.76 Prec@5, and 0.48 Recall@1 on a held-out test set. Then we automatically annotate the prompts by adding top 5 predicted import statements before the question; this is shown as +I in Table 1.

Fourth, we have tried variable name substitution similar to [1], i.e., replacing variable names and string literals in both question and answer with special tokens (`var_i`, `lst_i` etc.); all new tokens were added to the LLM vocabulary. We used it only for the *CoNaLa* dataset since it contains special labeling for variable names and constants in the question body (+R in Table 1). Finally, for the *StackOverflow* dataset we also have post body fields in addition to the title; we have tried to add the post bodies to the question, concatenating them with the titles and truncating the result to 270 tokens (+B in Table 1).

3.3 Evaluation procedure

For model evaluation on test sets, we have used both general-purpose NLP metrics—BLEU, BERTScore [28] and Rouge [15]—and metrics developed for program synthesis: CodeBLEU [19] and Ruby [23]. Note that while the *CoNaLa* leaderboard uses BLEU[7], recent studies show that direct application of BLEU and other automated metrics may lead to issues in code generation evaluation [8]. Unfortunately, there is still no good alternative, although recent studies suggest that Ruby is better aligned with human evaluation [12]; in our experiments, all metrics seem to agree on what the best models are. For *CoNaLa*, we used the test set provided by the authors; for the *StackOverflow* dataset, the test dataset is a random sample of 1 000 questions dated 2021 and later, while questions dated 2020 and earlier were used for training. In both cases, we randomly split the training set in the 90:10 ratio into train and validation parts.

4 Results

Table 1 shows our evaluation results; for *CoNaLa*, we show the best BLEU scores from the leaderboard by recently developed BERT+TAE [17], TranX+BERT [1],

[7] https://paperswithcode.com/sota/code-generation-on-conala

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_15

	Model	Options	BertScore	Rouge	CodeBLEU	Ruby	BLEU
	BERT+TAE [17]	—	—	—	—	—	0.3341
	TranX+BERT [1]	—	0.8998	0.4616	0.4742	0.5478	0.3420
	MarianCG [21]	—	0.8982	0.5000	0.5001	0.5121	0.3443
CoNaLa dataset	CoPilot [5]	—	0.8468	0.3376	0.2602	0.2595	0.1642
	CoPilot	*+I*	0.8761	0.4169	0.3531	0.3447	0.2230
	GPT-Neo [3]	—	0.7345	0.0425	0.0723	0.1524	0.0104
	GPT-Neo	*FT:C*	0.8688	0.4304	0.3841	0.4319	0.1633
	CodeGen [16]	—	0.8064	0.3093	0.2767	0.2749	0.1120
	CodeGen	*FT:C*	0.9015	0.5522	0.4850	0.5599	0.3171
	CodeGen	*FT:C+R*	0.8685	0.3711	0.4242	0.4712	0.2085
	CodeGen	*FT:C+I*	0.9115	0.5998	0.5696	0.6325	0.4319
	PanGu-Coder [6]	—	0.8825	0.4599	0.4421	0.4774	0.2121
	PanGu-Coder	*FT:C*	0.9217	0.5981	0.6032	0.6375	0.4098
	PanGu-Coder	*FT:C+I*	**0.9235**	**0.6061**	**0.6122**	**0.6511**	**0.4432**
StackOverflow dataset	TranX+BERT [1]	—	0.8286	0.1001	0.3829	0.2390	0.0515
	CoPilot [5]	—	0.7939	0.0965	0.0827	0.0864	0.0234
	GPT-Neo [3]	—	0.7552	0.0472	0.1187	0.1165	0.0107
	GPT-Neo	*FT:SO*	0.8052	0.1130	0.1362	0.0956	0.0464
	GPT-Neo	*FT:C*	0.7956	0.1237	0.2919	0.1676	0.0477
	CodeGen [16]	—	0.7438	0.0688	0.1469	0.1131	0.0205
	CodeGen	*FT:SO*	0.8021	0.1242	0.1562	0.0994	0.0508
	CodeGen	*FT:C*	0.8217	0.1490	0.3310	0.2025	0.0719
	PanGu-Coder [6]	—	0.8305	0.1497	0.2149	0.1387	0.0717
	PanGu-Coder	*FT:SO*	0.8448	0.1825	0.3280	0.2033	0.1087
	PanGu-Coder	*FT:C*	0.8386	0.1607	**0.4323**	**0.2467**	0.0555
	PanGu-Coder	*FT:SO+I*	0.8445	0.1843	0.3325	0.2041	0.1099
	PanGu-Coder	*FT:SO+B*	**0.8540**	**0.2452**	0.3217	0.2418	**0.1519**

Table 1. Evaluation study. Best results are shown in bold, results that differ from them statistically insignificantly are underlined; FT:C – fine-tuned on CoNaLa; FT:SO – on SO; +I – import classifier; +R – variable replacement; +B – using the question body.

and MarianCG [21] models; we have reproduced the results of TranX+BERT and used the MarianCG test set results to supplement BLEU with other metrics. Here, the best results were obtained by PanGu-Coder FT:C+I, followed by CodeGen FT:C+I; both set a new state of the art. PanGu-Coder was also best among the original pretrained models, but our proposed classification-based procedure (+I) gives significant improvements. GPT-Neo loses significantly, perhaps because it was originally pretrained only on the *Pile* dataset, while CodeGen and PanGu-Coder have comparable results; both were pretrained on code, but CodeGen also used the *Pile*, so we see that pretraining on a natural language corpus does not affect quality as much as suggested in [5]. Note how our proposed pretraining and fine-tuning heuristics make CodeGen results very close to PanGu-Coder, although pretrained versions show very different performance.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_15

Example 1. **Q**: *download a file 'url' over HTTP and save to "10MB"*
A:
```
urllib.request.urlretrieve(url, '10MB')
```

Example 2. **Q**: *delete all specific element from Python list*
I have a list like this. `a = ['\n', 'a', 'b', '\n', 'c', '\n']` As far as I know, the remove, pop, delete method only removes one by one. If I want to remove all '\n' from this list, the only way to do
A:
```
a = ['\n', 'a', 'b', '\n', 'c', '\n']
a = [x for x in a if x!= '\n']
```

Example 2.1. **Q**: *delete all specific element from Python list*
I: re json itertools sys random
A:
```
a = [1,2,3,4,5,6,7,8,9,10]
a[4:] = a[4:] + a[4:][4:]
```

Example 3. **Q**: *iterate odd indexes and add them back to original list*
I want to add "..." after every odd index in my list (or after every second word), but I can't quite figure out how to do it.
A:
```
def add_odds(lst):
    return [lst[i] + '...' if i\%2 else lst[i] for i in range(len(lst))]
```

Example 3.1. **Q**: *iterate odd indexes and add them back to original list*
I:
```
itertools collections random scipy os
```
A:
```
a = [1,2,3,4,5,6,7,8,9,10]
for i in range(len(a)-2):
    if a[i] % 2 == 0:
        a.append(a.pop(i)+a.pop(i+1))
print(a)
```

Table 2. Sample answers of PanGu-Coder FT:C+I (Ex. 1), FT:SO+B (Ex. 2, 3) and FT:SO+I (Ex. 2.1, 3.1).

On the *StackOverflow* dataset (Table 1), the best results were obtained by PanGu-Coder FT:SO+B, i.e., fine-tuned on *StackOverflow* and using concatenated question bodies. In general, results for *StackOverflow* are worse than for *CoNaLa*: real questions from Q&A web fora are more challenging for models than specially rephrased questions. In particular, a question's title often does not contain complete information, or the wording of the question is not familiar to the model (i.e., they do not represent similar doc-strings). On the other hand, the +I heuristic here helps much less than for *CoNaLa*, probably because titles of real questions often already contain library names while the *CoNaLa* dataset removes them to achieve "pure" program synthesis. We have run the bootstrap test for difference between means [10] with 100 000 samples and confidence levels 0.055 (*CoNaLa*) and 10^{-4} (*StackOverflow*). In Table 1, best results are shown in bold, and results that do *not* significantly differ from them are underlined.

Table 2 shows sample PanGu-Coder answers that illustrate our techniques. In Example 1, the classifier has produced `urllib` as a suggestion, and the model has used it successfully. In Examples 2 and 3, the titles (italicized first line) are not informative enough, as shown by the answers of FT:SO+I (Examples 2.1 and 3.1 in Table 2), but adding the question body leads to correct answers.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_15

5 Conclusion

In this work, we have demonstrated that simple fine-tuning on downstream tasks such as closed-book QA dramatically improves the quality of LLMs applied to code generation. In particular, we have achieved a new state of the art result on the CoNaLa dataset with BLEU score 0.4432. We have presented several fine-tuning and prompt engineering techniques that can improve the performance of a variety of LLMs, and we believe that similar approaches can work in other scenarios; e.g., fine-tuning also helps significantly on our newly collected *StackOverflow* dataset (very different from *CoNaLa*). Understanding real-life noisy natural language queries remains a challenging task for LLMs, evidenced by much better performance on *CoNaLa* where the questions are manually curated, and relatively low results overall (despite them being state of the art). Another problem is that common metrics, even code-specific ones [8], often lead to incorrect evaluation due to the diversity of correct code. We view these problems as important directions for further work.

Acknowledgements The work of Sergey Nikolenko was prepared in the framework of the strategic project "Digital Business" within the Strategic Academic Leadership Program "Priority 2030" at NUST MISiS.

References

1. Beau, N., Crabbé, B.: The impact of lexical and grammatical processing on generating code from natural language. In: Findings of the Association for Computational Linguistics: ACL 2022. pp. 2204–2214. Association for Computational Linguistics, Dublin, Ireland (May 2022). https://doi.org/10.18653/v1/2022.findings-acl.173,
2. Beyer, S., Macho, C., Di Penta, M., Pinzger, M.: What kind of questions do developers ask on stack overflow? a comparison of automated approaches to classify posts into question categories. Empirical Software Engineering **25** (05 2020). https://doi.org/10.1007/s10664-019-09758-x
3. Black, S., Gao, L., Wang, P., Leahy, C., Biderman, S.: GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow (Mar 2021). https://doi.org/10.5281/zenodo.5297715
4. Brown, T.B. et al.: Language models are few-shot learners (2020). https://doi.org/10.48550/ARXIV.2005.14165
5. Chen, M. et al.: Evaluating large language models trained on code. CoRR **abs/2107.03374** (2021), https://arxiv.org/abs/2107.03374
6. Christopoulou, F. et al.: Pangu-coder: Program synthesis with function-level language modeling (2022). https://doi.org/10.48550/ARXIV.2207.11280
7. Ding, N. et al.: Openprompt: An open-source framework for prompt-learning. arXiv preprint arXiv:2111.01998 (2021)
8. Evtikhiev, M., Bogomolov, E., Sokolov, Y., Bryksin, T.: Out of the bleu: how should we assess quality of the code generation models? (2022). https://doi.org/10.48550/ARXIV.2208.03133
9. Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al.: The pile: An 800gb dataset of diverse text for language modeling. arXiv preprint arXiv:2101.00027 (2020)

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_15

10. Hall, P., Hart, J.D.: Bootstrap test for difference between means in nonparametric regression. Journal of the American Statistical Association **85**(412), 1039–1049 (1990). https://doi.org/10.1080/01621459.1990.10474974
11. Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Burns, C., Puranik, S., He, H., Song, D., Steinhardt, J.: Measuring coding challenge competence with apps (2021). https://doi.org/10.48550/ARXIV.2105.09938
12. Kovalchuk, S.V., Lomshakov, V., Aliev, A.: Human perceiving behavior modeling in evaluation of code generation models. In: Proceedings of the 2nd Workshop on Natural Language Generation, Evaluation, and Metrics (GEM). pp. 287–294. ACL, Abu Dhabi, UAE (Dec 2022), https://aclanthology.org/2022.gem-1.24
13. Lee, N., Li, B.Z., Wang, S., Yih, W.t., Ma, H., Khabsa, M.: Language models as fact checkers? (2020). https://doi.org/10.48550/ARXIV.2006.04102
14. Li, Y. et al.: Competition-level code generation with alphacode (2022). https://doi.org/10.48550/ARXIV.2203.07814
15. Lin, C.Y.: ROUGE: A package for automatic evaluation of summaries. In: Text Summarization Branches Out. pp. 74–81. Association for Computational Linguistics, Barcelona, Spain (Jul 2004), https://aclanthology.org/W04-1013
16. Nijkamp, E. et al.: Codegen: An open large language model for code with multi-turn program synthesis (2022). https://doi.org/10.48550/ARXIV.2203.13474
17. Norouzi, S., Cao, Y.: Semantic parsing with less prior and more monolingual data. CoRR **abs/2101.00259** (2021), https://arxiv.org/abs/2101.00259
18. Petroni, F. et al.: Language models as knowledge bases? (2019). https://doi.org/10.48550/ARXIV.1909.01066
19. Ren, S. et al.: Codebleu: a method for automatic evaluation of code synthesis (2020). https://doi.org/10.48550/ARXIV.2009.10297
20. Roberts, A., Raffel, C., Shazeer, N.: How much knowledge can you pack into the parameters of a language model? (2020). https://doi.org/10.48550/ARXIV.2002.08910
21. Soliman, A.S., Hadhoud, M.M., Shaheen, S.I.: Mariancg: a code generation transformer model inspired by machine translation. Journal of Engineering and Applied Science **69**(1), 1–23 (2022)
22. Song, K., Tan, X., Qin, T., Lu, J., Liu, T.Y.: Mpnet: Masked and permuted pre-training for language understanding. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. NIPS'20, Curran Associates Inc., Red Hook, NY, USA (2020)
23. Tran, N., Tran, H., Nguyen, S., Nguyen, H., Nguyen, T.: Does bleu score work for code migration? In: 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC). pp. 165–176 (2019).
24. Xu, F.F., Alon, U., Neubig, G., Hellendoorn, V.J.: A systematic evaluation of large language models of code (2022). https://doi.org/10.48550/ARXIV.2202.13169
25. Ye, Q. et al.: Studying strategically: Learning to mask for closed-book qa (2020). https://doi.org/10.48550/ARXIV.2012.15856
26. Yin, P., Deng, B., Chen, E., Vasilescu, B., Neubig, G.: Learning to mine aligned code and natural language pairs from stack overflow. In: 2018 IEEE/ACM 15th Intl. Conf. on Mining Software Repositories (MSR). pp. 476–486 (2018)
27. Yin, P., Neubig, G.: TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 7–12. ACL, Brussels, Belgium (Nov 2018). https://doi.org/10.18653/v1/D18-2002
28. Zhang, T., Kishore, V., Wu, F., Weinberger, K.Q., Artzi, Y.: Bertscore: Evaluating text generation with bert (2019). https://doi.org/10.48550/ARXIV.1904.09675

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_15