Project Proposal: Non-parametric Language Models for Natural Language to Code Generation

Alex Xie
https://axie66.github.io/07400-project/

November 10, 2021

1 Project Description

I will be working with Professor Vincent Hellendoorn in the Institute for Software Research on constructing and augmenting non-parametric language models for natural language to code generation.

Developers perform the task of natural lanugage to code generation on a daily basis to translate their ideas into working source code. However, they very rarely do so in a vacuum, instead continuously referring to external resources such as API documentation, StackOverflow, and Github as they work. Most code generation models (and most NLP models in general), on the other hand, are self-contained, relying solely upon representations derived from their training to make predictions. This puts a large burden on the model to capture *all* of the patterns present in the training data; this is particularly difficult for the task of code generation, which requires fine-grained lexical, syntactic, and semantic knowledge of both programming and natural languages. Perhaps as a result of this burden, current state-of-the-art code generation models have been found to produce relatively low quality and simple code [13].

Inspired by the developer workflow, we propose extending existing code generation models by adding a non-parametric code retrieval component. Specifically, we use a k nearest neighbor language model (kNN-LM), which combines the output distribution of a trained parametric model with a distribution corresponding to the k most similar contexts retrieved from an external datastore [4]. The kNN component serves to reduce the burden on the parametric model, allowing it to explicitly reference rare patterns in the datastore rather than implicitly capture such patterns in its weights. While kNN-LMs were originally designed for natural language modeling and to the best of our knowledge have not been applied to code generation, recent work has shown that they are readily adapted to both code language modeling [11] and sequence to sequence tasks such as machine translation [3].

Additionally, datasets for code generation are relatively small due to the difficulty in manually annotating natural language/code pairs. For instance, the CoNaLa dataset [14] contains only around 2k human-annotated training examples. However, it contains another 600k noisy, unannotated training examples automatically mined from StackOverflow; more generally, trillions of lines of code can be inexpensively scraped from the web, but cannot be used by most existing models. kNN-LMs offer a way to incorporate this vast amount of data by including it in the datastore along with annotated data, allowing it to be retrieved and improve predictions at test time.

If we are successful in this work, we will be a step closer to creating reliable and interpretable code generation and autocompletion tools for developers, applications which would serve to greatly increase developer productivity. Further, if we can successfully incorporate unannotated data into our model, we will have made progress toward reducing the financial and labor costs

of creating not just code generation models, but NLP systems in general. Specifically, this could enable the creation of smaller expert-annotated datasets for specific tasks, which might then be used in conjunction with cheap, widely available data to train robust models.

Most of our work will revolve around the CoNaLa dataset [14], a commonly used dataset for natural language to code generation created by researchers at CMU. Further, rather than developing entirely new kNN-LM architectures, we plan to augment existing code generation models with kNN mechanisms. Specifically, we take the BERT-TAE model introduced by Norouzi et al. [7], the current state-of-the-art for CoNaLa; if time permits, we may be able to experiment with augmenting other models such as TranX [15], a syntax tree based model.

A major challenge of this project will be effectively incorporating unannotated code-only data into the datastore. Since contexts are stored in the datastore as fixed length vector representations, it is important that these representations are not only expressive, but also *consistent* for both paired and unpaired data; otherwise, the model might fail to retrieve relevant contexts. Additionally, since kNN-LMs generally use the trained parametric model to generate the context representations, we may face issues if the parametric model itself is not of high enough quality, i.e. it fails to produce semantically meaningful representations. Given the difficulty of the CoNaLa dataset, there is a definite possibility that this could occur, necessitating more advanced methods for learning representations.

2 Project Goals

2.1 75% Goals

- Apply kNN-LMs to code generation initially using only the 2k annotated training examples from the CoNaLa dataset
- Explore two methods for incorporating the 600k automatically-mined examples from CoNaLa into the kNN-LM datastore: (1) naively encoding with an empty natural language intent and (2) back-translating using an existing code summarization model to obtain an intent

2.2 100% Goals

- Apply kNN-LMs to code generation initially using only the 2k annotated training examples from the CoNaLa dataset
- Explore two methods for incorporating the 600k automatically-mined examples from CoNaLa into the kNN-LM datastore: (1) naively encoding with an empty natural language intent and (2) back-translating using an existing code summarization model to obtain an intent
- Augment datastore with additional data from the CodeSearchNet Corpus by retrieving examples from the corpus related to CoNaLa intents using models from the CodeSearchNet Challenge
- Investigate usage of kNN-LM as a decoder-only augmentation in which the datastore contains only source code (without the natural language intent) and serves only as a code language model
- Explore the effects of varying the amounts and proportions of annotated, unannotated, and mined/additional data used in the datastore

2.3 125% Goals

- Apply kNN-LMs to code generation initially using only the 2k annotated training examples from the CoNaLa dataset
- Explore two methods for incorporating the 600k automatically-mined examples from CoNaLa into the kNN-LM datastore: (1) naively encoding with an empty natural language intent and (2) back-translating using an existing code summarization model to obtain an intent
- Augment datastore with additional data from the CodeSearchNet Corpus by retrieving examples from the corpus related to CoNaLa intents using models from the CodeSearchNet Challenge
- Investigate usage of kNN-LM as a decoder-only augmentation in which the datastore contains only source code (without the natural language intent) and serves only as a code language model
- Explore the effects of varying the amounts and proportions of annotated, unannotated, and mined/additional data used in the datastore
- Evaluate kNN-LM performance in few-shot and zero-shot environments
- Explore additional methods for incorporating unannotated data into the datastore, such as metric learning approaches
- Augment other code generation model architectures with kNN-LM mechanisms
- Evaluate on other datasets or programming languages to confirm the extensibility of our approach

3 Project Milestones

3.1 First Technical Milestone

This research is currently in progress in the form of a final project for another class that I am currently taking this fall, 11-711 (Advanced NLP); this project is supervised by Frank Xu, a LTI PhD student working on code generation. The stated goals of this project are to augment the current state of the art model for the CoNaLa dataset [7] with a kNN-LM mechanism as well as to incorporate unannotated data into the kNN datastore (i.e. near the 75% project goal). However, given that the duration of the project is only a month, it is likely that it will not be able to fully meet the 75% goals, thus providing a decent start while leaving plenty of room for further work in 07-400.

3.2 First Biweekly Milestone:

By the first milestone, I plan to have results for that the kNN-augmented model using only annotated data, ideally demonstrating that it performs just as well as if not better than the original model on CoNaLa. Additionally, at this point, I hope to have preliminary results for the kNN model using both annotated and unannotated data.

3.3 Second Biweekly Milestone:

By the second milestone, I hope to have publishable results for the kNN model using both annotated and unannotated data, complete with a comparison between the two methods for incorporating unannotated data, analyses of where each method fails, and what type of data each method favors. In addition, if neither method provides strong results, I may propose additional methods to address their shortcomings, i.e. learning more significant representations using a contrastive objective or Siamese networks.

3.4 Third Biweekly Milestone:

By this milestone, I hope to have preliminary results regarding the usage of kNN-LMs with code-only datastores as decoder-only augmentations. In addition, if any additional methods for incorporating unannotated data have been proposed, I hope to have preliminary results evaluating their effectiveness.

3.5 Fourth Biweekly Milestone:

I hope to have augmented the datastore with additional similar examples from the CodeSearchNet Corpus (and perhaps also examples mined from StackOverflow or Github) and obtained preliminary results for kNN-LMs using this datastore.

3.6 Fifth Biweekly Milestone:

I hope to have completed some auxiliary analyses of the model, including studies comparing the effects of using different proportions of labeled, unlabeled, and external data in the datastore. In addition, if time permits, I would like to investigate model performance in few-shot or low-data settings, which should presumably benefit most from the datastore.

3.7 Sixth Biweekly Milestone:

By this point, I hope to have compiled all my work so far into a draft paper. In addition, for completeness, it may be necessary to train or evaluate other baseline models for comparison, as not all of them report numbers for CoNaLa.

3.8 Seventh Biweekly Milestone:

By the final milestone, I hope to have my project at the very least complete as described under 100% goals. In addition, based on my findings and model evaluations to that point, I hope to propose additional avenues for research into non-parametric language models or code generation.

4 Literature Search

Recent approaches to natural language to code generation can be broadly separated into two categories, generation-based and retrieval-based. Generation based approaches rely on the model to predict code using only information implicitly stored in its parameters; due to the difficulty of such a task, many approaches have used inductive biases such as tree structures [15], though this very often is not easily extensible due to the amount of hand-crafting required. More recent

generation approaches have adopted the pre-train/fine-tune paradigm, pre-training a model in an unsupervised manner on large amounts of code and text before fine-tuning it for specific downstream tasks [7][10][12]. Meanwhile, retrieval-based approaches range from traditional information retrieval methods to deep learning approaches [1] [2].

More recently, there have been efforts to combine generation and retrieval methods, for both NLP tasks in general [3][4][5] and code generation in particular. One recent approach generates code conditioned upon examples retrieved by searching a database with the natural language intent [8]; while this is similar to a kNN-LM, it differs in that the retrieval occurs *prior* to generation, using only the intent (as opposed to at each generation step, using both the intent and the context in a kNN-LM); hence, it is a coarser retrieval mechanism than kNN-LM. Due to the similarity, we plan on using this as a baseline to compare our approach against.

Finally, our idea of incorporating code-only data is inspired by similar work on leveraging monolingual data for machine translation [6][9]. It could be necessary to look into additional work within the machine translation literature on this topic if our initial methods do not achieve desired results.

5 Resources Needed

Our main resources will be Python and PyTorch (and associated modules such as `transformers`). We will make use of the CoNaLa dataset [14] to train and evaluate our models. We will likely also use the CodeSearchNet Corpus [2] and potentially also mine data from open source sites such as StackOverflow and Github to obtain additional data for our datastore. My current plan is to use AWS EC2 GPU instances to train models for this project, paid for with leftover AWS credits from other courses (as well as any provided by this course). However, it would probably be ideal to use CMU/SCS-affiliated GPU resources for this project so that funding is not an issue.

References

[1] Shirley Anugrah Hayati, Raphael Olivier, Pravalika Avvaru, Pengcheng Yin, Anthony Tomasic, and Graham Neubig. Retrieval-based neural code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 925–930. Association for Computational Linguistics, 2018.

[2] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

[3] Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Nearest neighbor machine translation. In *International Conference on Learning Representations (ICLR)*, 2021.

[4] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through Memorization: Nearest Neighbor Language Models. In *International Conference on Learning Representations (ICLR)*, 2020.

[5] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020.

[6] Benjamin Marie and Atsushi Fujita. Synthesizing monolingual data for neural machine translation, 2021.

[7] Sajad Norouzi, Keyi Tang, and Yanshuai Cao. Code generation from natural language with less prior knowledge and more monolingual data. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 776–785, Online, August 2021. Association for Computational Linguistics.

[8] Md Rizwan Parvez, Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Retrieval augmented code generation and summarization. In *EMNLP-Findings*, 2021.

[9] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany, August 2016. Association for Computational Linguistics.

[10] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021*, 2021.

[11] Frank F. Xu, Junxian He, Graham Neubig, and Vincent J. Hellendoorn. Capturing structural locality in non-parametric language models. 2021.

[12] Frank F. Xu, Zhengbao Jiang, Pengcheng Yin, and Graham Neubig. Incorporating external knowledge through pre-training for natural language to code generation. In *Annual Conference of the Association for Computational Linguistics*, 2020.

[13] Frank F. Xu, Bogdan Vasilescu, and Graham Neubig. In-ide code generation from natural language: Promise and challenges. *ACM Transactions on Software Engineering and Methodology*, 30, January 2021.

[14] Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. Learning to mine aligned code and natural language pairs from stack overflow. In *International Conference on Mining Software Repositories*, MSR, pages 476–486. ACM, 2018.

[15] Pengcheng Yin and Graham Neubig. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP) Demo Track*, 2018.