The Journal of Open Source Software

DeepSynth: Scaling Neural Program Synthesis with Distribution-based Search

Théo Matricon [1], **Nathanaël Fijalkow** [1,2], **Guillaume Lagarde**[1], **and Kevin Ellis**[3]

1 CNRS, LaBRI and Université de Bordeaux, France **2** The Alan Turing Institute of data science, United Kingdom **3** Cornell University, United States

DOI: 10.21105/joss.04151

Software

- Review
- Repository
- Archive

Editor: Arfon Smith

Reviewers:

- @njuaplusplus
- @bzz

Submitted: 25 January 2022
Published: 16 October 2022

Summary

Writing software is tedious, error-prone, and accessible only to a small share of the population – yet coding grows increasingly important as the digital world plays larger and larger roles in peoples' lives. Programming by example seeks to make programming more reliable and accessible by allowing non-technical users to specify programs only from pairs of input-output examples. `DeepSynth` is a general purpose programming by example tool, which combines predictions from neural networks with symbolic methods to generate programs.

Statement of need

`DeepSynth` was used for the experiments of the recently published Fijalkow et al. (2022). The main purpose was to evaluate and compare different search algorithms and their integration with neural predictions. Beyond the publication Fijalkow et al. (2022), the goal of `DeepSynth` is to serve as a reference implementation of existing neuro-symbolic methods for programming by example. Indeed, it includes different options for different parts of the pipeline, focussing on neural predictions and search algorithms.

How it works

`DeepSynth` is parameterised by a domain specific language (DSL), which is the programming language chosen for solving a program synthesis task. The user first specifies a DSL by giving a set of primitives together with their types and semantics, as well as semantic and syntactic constraints (such as program depth). The compilation phase constructs a Context Free Grammar (CFG) representing the set of programs.

The second ingredient is the prediction model. `DeepSynth` leverages PyTorch (Paszke et al., 2019): a neural network reads the examples and outputs predictions to guide the search towards likely programs. `DeepSynth` includes end to end procedures for training a neural network that makes predictions from examples. The predictions are given as distributions on the derivation rules of CFG.

Figure 1 illustrates the machine learning pipeline on a toy DSL describing integer list manipulating programs.

The Journal of Open Source Software

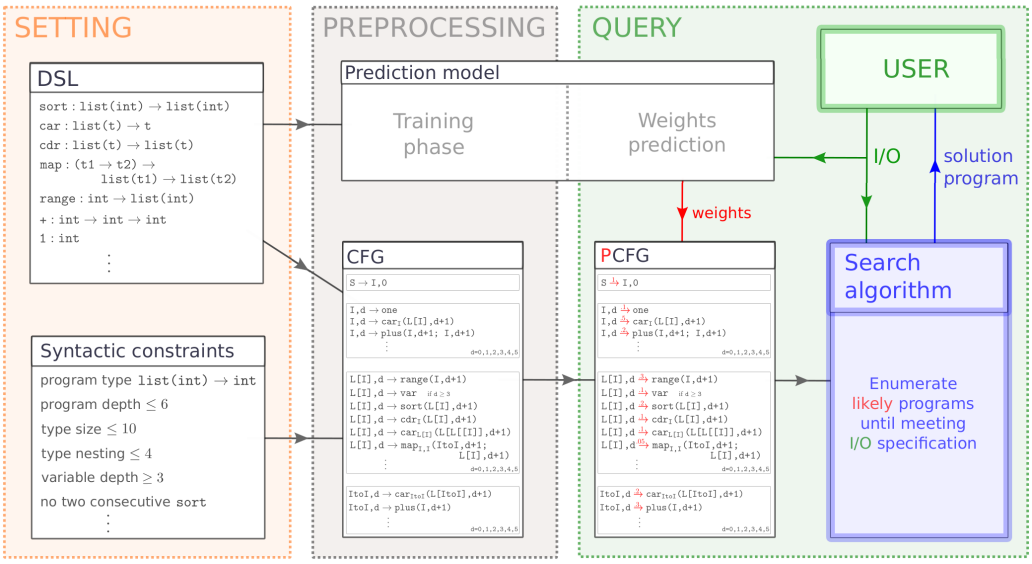

Figure 1: Pipeline for neural predictions for syntax guided program synthesis.

Once the DSL is specified and the prediction model trained, `DeepSynth` can be used as follows: the end user gives a few input-output examples, and `DeepSynth` searches for a program matching the examples. `DeepSynth` includes a number of predictions-guided search algorithms; the most efficient is `HeapSearch`. An optional parallelisation framework allows the user to split the search on multiple CPUs by partitioning the progam space.

We refer to the `readme` for example uses of `DeepSynth`. The full technical details are described in Fijalkow et al. (2022).

State of the field

Programming by example has been intensively investigated in the past years both in academia and in industry, spearheaded by the success of FlashFill in Microsoft Excel (see Gulwani (2011)), allowing users to synthesize spreadsheet programs by giving examples. FlashFill is now integrated into the larger and more ambitious project PROSE by Microsoft Research (see the website). PROSE is a general purpose program synthesis tool (see the GitHub repository). It is based on constraint programming: in addition to the DSL, the end-user needs to specify so-called witness functions, which are used for specifying a programming by example instance in a SAT or SMT solver.

Other tools have emerged recently exploiting the programming by example paradigm, for instance SmartFill for Google Sheets, and the TF-Coder for TensorFlow (see Shi et al. (2020)). Unlike PROSE, they are tailored to address a single domain: for SmartFill spreadsheet programs, and for TF-Coder tensorflow programs. A number of other prototype tools have been developed in the academic world: DeepCoder was the first leveraging deep learning techniques (see Balog et al. (2017)), PC-Coder (see Zohar & Wolf (2018)), and recently the general-purpose DreamCoder (see Ellis et al. (2021)).

The recent release of the GitHub Copilot (see GitHub (2021)), powered by the Codex large language model from OpenAI, shows the wide applicability of the program synthesis: Copilot is presented as `your AI pair programmer`, meaning that it assists developers by autocompleting pieces of code in an interactive fashion. The key difference is that specifications in Copilot are given in natural language, which may or may not include examples.

Features

This package has the following capabilities:

- Create a DSL from syntactic constraints and semantics functions;
- Transform this DSL into a Context Free Grammar (CFG);
- Transform this CFG into a Probabilistic CFG (PCFG);
- Sample programs from a PCFG;
- Enumerate programs in a PCFG with different algorithms including `HeapSearch`;
- A grammar splitter that enables to split the search into n independent searches, enabling parallel search scaling linearly with the number of CPUs;
- A neural network architecture to predict probabilities of a CFG given pairs of input-output examples and its automatic training procedure from a DSL that supports `int`, `bool` and `list` inputs.

References

Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., & Tarlow, D. (2017). DeepCoder: Learning to write programs. *International Conference on Learning Representations, ICLR*. https://openreview.net/forum?id=ByldLrqlx

Ellis, K., Wong, C., Nye, M. I., Sablé-Meyer, M., Morales, L., Hewitt, L. B., Cary, L., Solar-Lezama, A., & Tenenbaum, J. B. (2021). DreamCoder: Bootstrapping inductive program synthesis with wake-sleep library learning. *International Conference on Programming Language Design and Implementation, PLDI*. https://doi.org/10.1145/3453483.3454080

Fijalkow, N., Lagarde, G., Matricon, T., Ellis, K., Ohlmann, P., & Potta, A. N. (2022). Scaling neural program synthesis with distribution-based search. *Proceedings of the AAAI Conference on Artificial Intelligence*, *36*(6), 6623–6630. https://doi.org/10.1609/aaai.v36i6.20616

GitHub. (2021). *GitHub copilot*. https://copilot.github.com/

Gulwani, S. (2011). Automating string processing in spreadsheets using input-output examples. *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*. https://doi.org/10.1145/1926385.1926423

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Shi, K., Bieber, D., & Singh, R. (2020). TF-coder: Program synthesis for tensor manipulations. *CoRR*, *abs/2003.09040*. https://doi.org/10.1145/3517034

Zohar, A., & Wolf, L. (2018). Automatic program synthesis of long programs with a learned garbage collector. *Neural Information Processing Systems, NeurIPS*. https://proceedings.neurips.cc/paper/2018/hash/390e982518a50e280d8e2b535462ec1f-Abstract.html