

DeepSynth: Scaling Neural Program Synthesis with Distribution-based Search

Théo Matricon¹, Nathanaël Fijalkow^{1, 2}, Guillaume Lagarde¹, and Kevin Ellis³

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](https://doi.org/10.21105/joss.04151)

Software

- [Review](#) ↗
- [Repository](#) ↗
- [Archive](#) ↗

Editor: [Yuan Tang](#) ↗

Reviewers:

- [@njuaplusplus](#)
- [@Kraks](#)

Submitted: 25 January 2022

Published: 10 February 2022

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

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. \(2021\)](#). The main purpose was to evaluate and compare different search algorithms and their integration with neural predictions. Beyond the publication [Fijalkow et al. \(2021\)](#), 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.

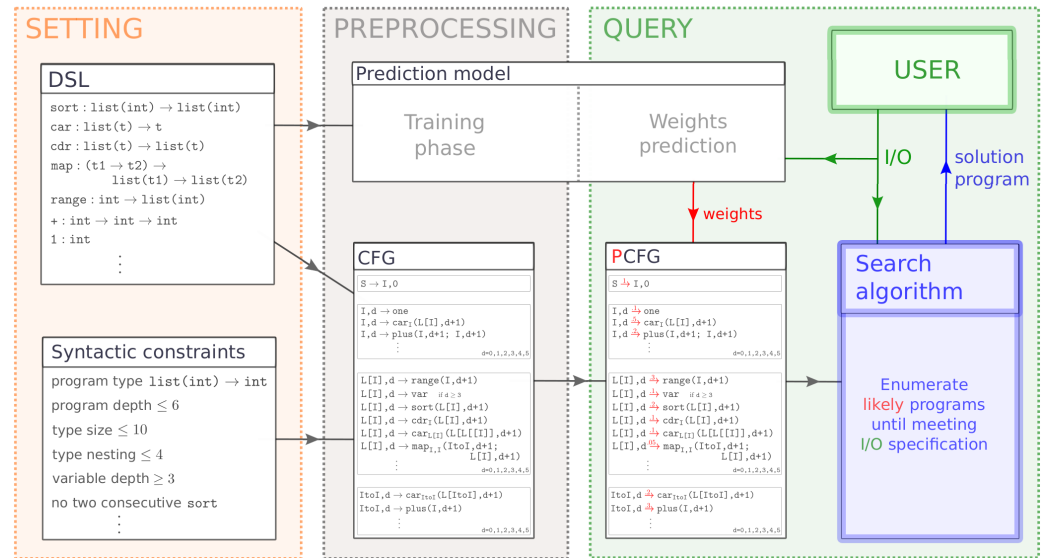


Figure 1: Pipeline for neural predictions for syntax guided program synthesis.

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 program space.

We refer to the [readme](#) for example uses of DeepSynth. The full technical details are described in [Fijalkow et al. \(2021\)](#).

State of the field

Programming by example has been intensively investigated in the past years both in academia and in industry, in particular thanks to the success of FlashFill in Microsoft Excel, allowing users to synthesize spreadsheet programs by giving examples. The recent release of the GitHub Copilot ([GitHub, 2021](#)), powered by the Codex big language model from OpenAI, shows the wide applicability of the framework: Copilot is presented as your AI pair programmer, meaning that it assists developers by autocompleting pieces of code in an interactive fashion.

Features

The package allows to:

- 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

- Fijalkow, N., Lagarde, G., Matricon, T., Ellis, K., Ohlmann, P., & Potta, A. (2021). *Scaling neural program synthesis with distribution-based search*. <http://arxiv.org/abs/2110.12485>
- GitHub. (2021). *GitHub copilot*. <https://copilot.github.com/>
- 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>

DRAFT