Do Machine Learning Models Produce TypeScript Types That Type Check? (Artifact)

Ming-Ho Yee
Northeastern University, Boston, MA, USA

Arjun Guha
Northeastern University, Boston, MA, USA
Roblox Research, San Mateo, CA, USA

Abstract

Type migration is the process of adding types to untyped code to gain assurance at compile time. TypeScript and other gradual type systems facilitate type migration by allowing programmers to start with imprecise types and gradually strengthen them. However, adding types is a manual effort and several migrations on large, industry codebases have been reported to have taken several years. In the research community, there has been significant interest in using machine learning to automate TypeScript type migration. Existing machine learning models report a high degree of accuracy in predicting individual TypeScript type annotations. However, in this paper we argue that accuracy can be misleading, and we should address a different question: can an automatic type migration tool produce code that passes the TypeScript type checker?

We present TypeWeaver, a TypeScript type migration tool that can be used with an arbitrary type prediction model. We evaluate TypeWeaver with three models from the literature: DeepTyper, a recurrent neural network; LambdaNet, a graph neural network; and InCoder, a general-purpose, multi-language transformer that supports fill-in-the-middle tasks. Our tool automates several steps that are necessary for using a type prediction model, including (1) importing types for a project's dependencies; (2) migrating JavaScript modules to TypeScript notation; (3) inserting predicted type annotations into the program to produce TypeScript when needed; and (4) rejecting non-type predictions when needed.

We evaluate TypeWeaver on a dataset of 513 JavaScript packages, including packages that have never been typed before. With the best type prediction model, we find that only 21% of packages type check, but more encouragingly, 69% of files type check successfully.

2012 ACM Subject Classification Software and its engineering → Source code generation; General and reference → Evaluation; Theory of computation → Type structures

Keywords and phrases Type migration, deep learning

Digital Object Identifier 10.4230/DARTS.9.2.5

Funding This work is partially supported by the National Science Foundation grant CCF-2102291.

Acknowledgements We thank Northeastern Research Computing and the New England Research Cloud for providing computing resources; and Donald Pinckney and the anonymous reviewers for their feedback.

Related Article Ming-Ho Yee and Arjun Guha, "Do Machine Learning Models Produce TypeScript Types That Type Check?", in 37th European Conference on Object-Oriented Programming (ECOOP 2023), LIPIcs, Vol. 263, pp. 37:1–37:28, 2023. `https://doi.org/10.4230/LIPIcs.ECOOP.2023.37`

Related Conference 37th European Conference on Object-Oriented Programming (ECOOP 2023), July 17–21, 2023, Seattle, Washington, United States

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2023 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

This artifact is *functional.* All figures and tables in the paper (with the exceptions of Figures 1–3 and 13–16, which are examples) are generated from the experiments. They can be regenerated from the CSV data, and should match the paper. It is also possible to run all experiments from scratch: in this case, the figures and tables *will not exactly match the paper*, because the machine learning models are non-deterministic and may return different results.

This artifact is *reusable.* The artifact can be used to migrate your own JavaScript project to TypeScript, using the infrastructure in `./data/playground/`. New models can be added to the artifact, though the process is time consuming, as models typically expose different interfaces and thus require custom adapters. New datasets can also be added to the artifact.

For full instructions, refer to the `README.md` and `ECOOP_AE_Submission_Document.md` files.

2 Content

The artifact package includes:
- Benchmarks (source code of NPM packages)
- Results (log files, TypeScript source code, CSV files)
- Code (Python, TypeScript, and R)

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at: `https://doi.org/10.5281/zenodo.7662708` and `https://github.com/nuprl/TypeWeaver/tree/ecoop2023-artifact`.

4 Tested platforms

The artifact has been tested on Ubuntu Linux, and requires Python +3.6 and the `tqdm` package. All other software dependencies are managed by Open Container Initiative [4] images; OCI implementations include Podman [3] and Docker [2]. The GPU experiments require a GPU with at least 14 GB of VRAM.

5 License

The artifact is available under a Creative Commons Attribution 4.0 International license (CC BY 4.0) [1].

6 MD5 sum of the artifact

415b9ed685645c24f95d500d099c4c08

7 Size of the artifact

666.6 MB

References

1 Creative Commons. Attribution 4.0 International (CC BY 4.0). https://creativecommons.org/licenses/by/4.0/. Accessed: 2023-05-25.

2 Docker Inc. Docker. https://www.docker.com/. Accessed: 2023-05-25.

3 Podman. Podman. https://podman.io/. Accessed: 2023-05-25.

4 The Linux Foundation. Open Container Initiative. https://opencontainers.org/. Accessed: 2023-05-25.