ComPile: A Large IR Dataset from Production Sources

Aiden Grossman[1]* **Ludger Paehler**[2] **Konstantinos Parasyris**[3] **Tal Ben-Nun**[3]
Jacob Hegna[4] **William Moses**[5] **Jose M Monsalve Diaz**[6] **Mircea Trofin**[7]
Johannes Doerfert[3]
[1]UC Davis [2]Technical University of Munich [3]Lawrence Livermore National Laboratory
[4]University of Minnesota [5]University of Illinois Urbana Champaign
[6] Argonne National Laboratory [7] Google, Inc.
`amgrossman@ucdavis.edu` `ludger.paehler@tum.de`
`{parasyris1,talbn,jdoerfert}@llnl.gov` `jacobhegna@gmail.com`
`wsmoses@illinois.edu` `jmonsalvediaz@anl.gov` `mtrofin@google.com`

Abstract

Code is increasingly becoming a core data modality of modern machine learning research impacting not only the way we write code with conversational agents like OpenAI's ChatGPT, Google's Bard, or Anthropic's Claude, the way we translate code from one language into another, but also the compiler infrastructure underlying the language. While modeling approaches may vary and representations differ, the targeted tasks often remain the same within the individual classes of models. Relying solely on the ability of modern models to extract information from unstructured code does not take advantage of 70 years of programming language and compiler development by not utilizing the structure inherent to programs in the data collection. This detracts from the performance of models working over a tokenized representation of input code and precludes the use of these models in the compiler itself. To work towards better intermediate representation (IR) based models, we fully utilize the LLVM compiler infrastructure, shared by a number of languages, to generate a 182B token dataset of LLVM IR. We generated this dataset from programming languages built on the shared LLVM infrastructure, including Rust, Swift, Julia, and C/C++, by hooking into LLVM code generation either through the language's package manager or the compiler directly to extract the dataset of intermediate representations from production grade programs. Our dataset shows great promise for large language model training, and machine-learned compiler components.

1 Introduction

In several pieces of previous work (8; 14), the transformative potential of machine learning was harnessed, machine-learned heuristic replacements developed, and in some cases (21) the *heuristics were upstreamed to the main LLVM codebase*, improving all code run through LLVM when the ML heuristics are enabled. Orthogonal to the replacement of heuristics with machine learning, a large number of people have explored the ordering of compiler passes (4; 10). While the learning of pass orderings was initially held back by the lack of easy-to-access, high-performance reinforcement learning environments to validate new reinforcement learning strategies, this has by now been addressed with the introduction of CompilerGym (4). In contrast, the learning of entirely new heuristics, optimization passes, and other compiler components with large language models (22; 3) to realize the transformative potential of this model class is held back partially by the lack of large datasets of high-quality code to train such models properly. Models are only trained on smaller

*Corresponding author

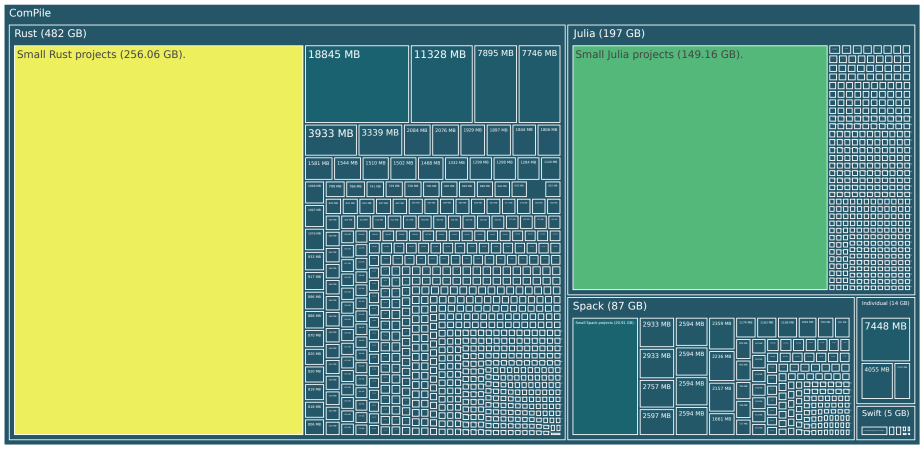

Figure 1: Size distribution of LLVM intermediate representation (IR) bitcode within ComPile before de-duplication within and among languages. Projects that we considered small and pooled had less than 100MB of bitcode.

datasets, such as Anghabench (5), Exebench (2), and HPCORPUS (11), or sometimes rely on synthetic benchmarks. Small datasets ultimately lead to smaller, worse-performing models (9).

1.1 Contributions

Focussing on the paradigm of taking a pre-trained basic building block, a “foundation model”, we pose the question *"What does a modern, large code training dataset for compilers actually look like?"* and construct a high-quality dataset of a similar scale to existing LLM datasets solely at the level of LLVM-IR. Within this context, we associate quality with the *usage* of code, with code being used more often being of higher quality for our purposes. Correctly being able to reason about very widespread code in production systems is incredibly important for compiler work. In the short term, we believe our dataset will enable the training of larger language models for compilers useful for an ever broader array of downstream tasks after fine-tuning, and in the long-term enable use-cases such as direct performance prediction to obtain a reliable runtime estimate without ever running a single line of code. To these goals, our work makes the following contributions:

- The introduction of a 2.4TB dataset of textual LLVM-IR from Rust, Julia, Swift, and C/C++ with {182B, 119B, 102B, 87B} tokens at a vocabulary size of {10k, 50k, 100k, 200k } respectively. The pre-deduplication size distribution is shown in Figure 1.
- Open-sourcing of our workflow and compiler tooling to construct massive-scale IR datasets.

2 Background

Building upon package ecosystems as sources of intermediate representation is ideal due to the large amount of packaged code and the abstraction over the build systems of individual projects. In addition, package ecosystems act as a filter. Only code that gets used in production systems will get packaged. The build system abstraction is due to a common build wrapper that builds recipes which often specify exact build steps, including an exact specification of dependencies. Modifying these build processes allows us to take advantage of this existing infrastructure. In this work, we choose to specifically focus on utilizing package managers that explicitly allow setting compiler flags, such as the from-source package manager Spack (6) that is focused on high-performance computing (HPC).

In addition to utilizing package managers, we also take advantage of several aspects of the LLVM compilation infrastructure (16), particularly the Clang C/C++ frontend and LLVM-IR, the intermediate representation LLVM uses. The full process of compilation, such as the one performed by Clang with LLVM during the compilation of C/C++, is composed of three main stages: the frontend, the middle-end, and the backend. A compiler frontend has the job of taking a piece of

Language	C	C++	Julia	Rust	Swift	Total
Size (Bitcode)	13 GB	81 GB	197 GB	482 GB	5 GB	778 GB
Size (Text)	61 GB	334 GB	1292 GB	1868 GB	22 GB	3577 GB
Dedup. Size (Bitcode)	8 GB	67 GB	130 GB	310 GB	4 GB	518 GB
Dedup. Size (Text)	34 GB	266 GB	856 GB	1221 GB	19 GB	2395 GB

Table 1: Amount of IR contained within ComPile in textual and bitcode form before and after deduplication.

source code, typically a single source file, sometimes called a translation unit, and generating a *module* of intermediate representation that can then be processed by a compiler middle-end, such as LLVM. A module typically contains multiple functions, referenced globals, and relevant metadata. Compiler intermediate representations, or IRs, are designed to sit between the source programming language and the compiler's output, assembly. They are typically designed to be source-language and target-agnostic. Within LLVM, the compiler middle-end operates over the IR produced by the frontend through a series of grouped operations called passes. A *pass* is designed to perform a specific task, such as removing dead code, simplifying the control flow graph, or combining instructions. After optimization, the compiler backend takes over, performing the necessary tasks to transform the (mostly) target-agnostic IR into target-specific machine code that can be executed on the target machine. The backend typically performs tasks such as instruction selection, instruction scheduling, and register allocation. We compose our dataset, *ComPile*, of LLVM-IR, as it gives a common framework across programming languages and target platforms. These properties and more make LLVM-IR a great modality for a compiler-centric dataset useful for compiler tasks such as program analysis, optimization, and code generation.

3 Dataset Construction

To construct the IR dataset [2], we use a set of curated sources from five different languages. focusing on code used in production systems. We include the majority of Spack (6), the Rust Crates Index, the Julia Package Index, the Swift Package Index, and several large single projects. Individual project sources are defined in `.json` files. While most projects are hosted in repositories on GitHub, we also added sources consisting of archived compressed source codes such as tarball files. The builders then ingest the information from the project on its build system, either through the manifest information, which contains the information on the building mechanism and commands, or through an ecosystem specific manifest processed by a script that is then processed into a complete package manifest. Next in the workflow is the LLVM-IR extraction. Extracting IR depends on the way the IR is presented in the source. A manifest that contains a list of LLVM bitcode modules extracted from the project is then created. Leaning into the shared LLVM compiler infrastructure, we are able to take advantage of existing LLVM tools and LLVM passes to obtain information about the LLVM-IR modules. After building, IR extraction, and deduplication, the dataset is then ready for downstream usage in analysis or training capacities. [3]

The aim of our IR extraction approach is to extract IR immediately after the frontend, before any LLVM optimization passes have run. To extract the bitcode into a structured corpus, we take advantage of the ml-compiler-opt tooling from MLGO (21) as it allows for the extraction of IR in a variety of cases. During IR extraction, we also collect some additional data, such as debug information, as it is represented in the IR. We specifically collect bitcode rather than textual IR as LLVM supports reading bitcode produced by older versions of LLVM but has no such support for textual IR, which is also easily produced by running `llvm-dis` over the collected corpus.

Training dataset deduplication can be important for the performance of several key model characteristics. (1; 12). To this end, We deduplicate the entire dataset presented in this paper at the module level by computing a combined hash of all global variables and functions, deduplicating based on a hashing implementation that only captures semantic details of the IR. We chose to deduplicate at the module level as this ensures the majority of the duplicate code is removed from the dataset while leaving all significant context within each module for performing module-level tasks.

[2]The entire dataset will be available on HuggingFace https://huggingface.co/datasets/llvm-ml/ComPile

[3]Scripts and builders to reproduce the entire dataset are available under the `llvm-ir-dataset-utils` subdirectory under https://zenodo.org/doi/10.5281/zenodo.10155760

Name of Dataset	Tokens	Size	Languages
The Stack (13)	-	2.9TB	358 Languages
ComPile (ours)	182B [1]	2.4TB	Rust, Swift, Julia, C/C++
Code Llama (20)	197B [2]	859GB	-
TransCoder (15)	163B	744GB	C++, Java, Python
AlphaCode (17)	-	715.1GB [4]	12 Languages
LLM for Compiler Opt. (3)	373M	1GB	C/C++

Table 2: Breakdown of Related Datasets.

4 Related Work

Most pretraining datasets for large language models (17; 13; 18) contain large swaths of code, scraping source code from hosting services like GitHub, and GitLab without taking the quality of the included code into account. Datasets of this type also do not guarantee that any of the code is compilable, and often contain auxiliary files such as documentation in Markdown. Complementary to these large pretraining-scale datasets, there exist a number of smaller, more focused datasets aimed at the fine-tuning of already pretrained large language models (23; 17; 19). These datasets are primarily collected through data extraction from coding competitions (17; 19), or the scraping of curated websites (23). This guarantees a higher level of quality in regards to buildability and structure for the included code, hence making them more optimal for fine-tuning. However, the data collection methodology implicitly introduces a lack of variety in the datasets, reducing model performance (7). For example, coding competititon datasets might include a couple thousand coding exercises which contain a great many solutions to the same exercises, but yet they are only solving the very same set of coding problems.

Additionally, there exist a number of domain-specific datasets (11; 2; 5). Often beginning with the web-scraping of large amounts of code, these approaches modify the resulting code in a number of ways. Examples include the modification of arbitrary source files to make them compilable (5) or executable (2). ComPile, while being able to fulfill similar dataset demands, offers a number of key advantages. The code in our dataset, by means of our dataset construction methodology, consists only of compilable code, using the same compilation toolchain as used for production deployments without changing semantics. Collecting IR before optimization allows for IR at any stage of the compilation pipeline to be easily generated. This allows ComPile to go significantly beyond the capabilities of previous compiler-targeted datasets.

5 Conclusion

In this work, we presented ComPile, a novel dataset of LLVM-IR collected from a number of package ecosystems consisting of large production-grade codebases. It is significantly larger than previous finetuning-focussed, and compiler-focussed code datasets, albeit smaller than large language model-focussed code pretraining datasets. ComPile's increased size in combination with its quality-focused construction methodology not only enables the systematic evaluation of previous work, but opens up entirely new avenues of research for IR-centric machine learning, and most specifically machine-learned compiler componentry for which the scale of this dataset paves the way to an entirely new generation of machine learning models for compilers.

6 Acknowledgements

This work was in parts prepared by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-855448).

We would like to thank Valentin Churavy, Todd Gamblin, Alec Scott, Harmen Stoppels, Massimiliano Culpo, Nikita Popov, and Arthur Eubanks for their assistance with understanding the relevant language-specific optimization pipelines and assistance with getting upstreamed patches through code review.

References

[1] ALLAMANIS, M. The adverse effects of code duplication in machine learning models of code. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (2019), pp. 143–153.

[2] ARMENGOL-ESTAPÉ, J., WOODRUFF, J., BRAUCKMANN, A., MAGALHÃES, J. W. D. S., AND O'BOYLE, M. F. P. Exebench: an ml-scale dataset of executable c functions. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming* (New York, NY, USA, Jun 2022), MAPS 2022, Association for Computing Machinery, p. 50–59.

[3] CUMMINS, C., SEEKER, V., GRUBISIC, D., ELHOUSHI, M., LIANG, Y., ROZIERE, B., GEHRING, J., GLOECKLE, F., HAZELWOOD, K., SYNNAEVE, G., AND LEATHER, H. Large language models for compiler optimization. arXiv:2309.07062 [cs].

[4] CUMMINS, C., WASTI, B., GUO, J., CUI, B., ANSEL, J., GOMEZ, S., JAIN, S., LIU, J., TEYTAUD, O., STEINER, B., ET AL. Compilergym: Robust, performant compiler optimization environments for ai research. In *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)* (2022), IEEE, pp. 92–105.

[5] DA SILVA, A. F., KIND, B. C., DE SOUZA MAGALHÃES, J. W., ROCHA, J. N., FERREIRA GUIMARÃES, B. C., AND QUINÃO PEREIRA, F. M. Anghabench: A suite with one million compilable c benchmarks for code-size reduction. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)* (Feb 2021), p. 378–390.

[6] GAMBLIN, T., LEGENDRE, M., COLLETTE, M. R., LEE, G. L., MOODY, A., DE SUPINSKI, B. R., AND FUTRAL, S. The spack package manager: bringing order to hpc software chaos. p. 1–12.

[7] GUO, Z. C., AND MOSES, W. S. Enabling transformers to understand low-level programs.

[8] HAJ-ALI, A., AHMED, N. K., WILLKE, T., SHAO, Y. S., ASANOVIC, K., AND STOICA, I. Neurovectorizer: end-to-end vectorization with deep reinforcement learning. In *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization* (New York, NY, USA, Feb 2020), CGO 2020, Association for Computing Machinery, p. 242–255.

[9] HOFFMANN, J., BORGEAUD, S., MENSCH, A., BUCHATSKAYA, E., CAI, T., RUTHERFORD, E., CASAS, D. D. L., HENDRICKS, L. A., WELBL, J., CLARK, A., ET AL. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556* (2022).

[10] HUANG, Q., HAJ-ALI, A., MOSES, W., XIANG, J., STOICA, I., ASANOVIC, K., AND WAWRZYNEK, J. Autophase: Juggling hls phase orderings in random forests with deep reinforcement learning. arXiv:2003.00671 [cs].

[11] KADOSH, T., HASABNIS, N., MATTSON, T., PINTER, Y., AND OREN, G. Quantifying openmp: Statistical insights into usage and adoption. *arXiv preprint arXiv:2308.08002* (2023).

[12] KANDPAL, N., WALLACE, E., AND RAFFEL, C. Deduplicating training data mitigates privacy risks in language models. In *International Conference on Machine Learning* (2022), PMLR, pp. 10697–10707.

[13] KOCETKOV, D., LI, R., ALLAL, L. B., LI, J., MOU, C., FERRANDIS, C. M., JERNITE, Y., MITCHELL, M., HUGHES, S., WOLF, T., ET AL. The stack: 3 tb of permissively licensed source code. *arXiv preprint arXiv:2211.15533* (2022).

[14] KULKARNI, S., CAVAZOS, J., WIMMER, C., AND SIMON, D. Automatic construction of inlining heuristics using machine learning. In *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)* (Feb 2013), p. 1–12.

[15] LACHAUX, M.-A., ROZIERE, B., CHANUSSOT, L., AND LAMPLE, G. Unsupervised translation of programming languages. *arXiv preprint arXiv:2006.03511* (2020).

[16] LATTNER, C., AND ADVE, V. Llvm: A compilation framework for lifelong program analysis & transformation. In *International symposium on code generation and optimization, 2004. CGO 2004.* (2004), IEEE, pp. 75–86.

[17] Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. Competition-level code generation with alphacode. *Science 378*, 6624 (2022), 1092–1097.

[18] Markovtsev, V., and Long, W. Public git archive: a big code dataset for all. In *Proceedings of the 15th International Conference on Mining Software Repositories* (2018), pp. 34–37.

[19] Puri, R., Kung, D. S., Janssen, G., Zhang, W., Domeniconi, G., Zolotov, V., Dolby, J., Chen, J., Choudhury, M., Decker, L., et al. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655* (2021).

[20] Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[21] Trofin, M., Qian, Y., Brevdo, E., Lin, Z., Choromanski, K., and Li, D. Mlgo: a machine learning guided compiler optimizations framework. *arXiv preprint arXiv:2101.04808* (2021).

[22] Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., and Chen, X. Large language models as optimizers. *arXiv preprint arXiv:2309.03409* (2023).

[23] Zhu, M., Jain, A., Suresh, K., Ravindran, R., Tipirneni, S., and Reddy, C. K. Xlcost: A benchmark dataset for cross-lingual code intelligence. *arXiv preprint arXiv:2206.08474* (2022).

ComPile: A Large IR Dataset from Production Sources

Aiden Grossman[1*] **Ludger Paehler**[2] **Konstantinos Parasyris**[3] **Tal Ben-Nun**[3]
Jacob Hegna[4] **William Moses**[5] **Jose M Monsalve Diaz**[6] **Mircea Trofin**[7]
Johannes Doerfert[3]

[1]UC Davis [2]Technical University of Munich [3]Lawrence Livermore National Laboratory
[4]University of Minnesota [5]University of Illinois Urbana Champaign
[6] Argonne National Laboratory [7] Google, Inc.

`amgrossman@ucdavis.edu` `ludger.paehler@tum.de`
`{parasyris1,talbn,jdoerfert}@llnl.gov` `jacobhegna@gmail.com`
`wsmoses@illinois.edu` `jmonsalvediaz@anl.gov` `mtrofin@google.com`

November 2023

1 Abstract

Code is increasingly becoming a core data modality of modern machine learning research impacting not only the way we write code with conversational agents like OpenAI's ChatGPT, Google's Bard, or Anthropic's Claude, the way we translate code from one language into another, but also the compiler infrastructure underlying the language. While modeling approaches may vary and representations differ, the targeted tasks often remain the same within the individual classes of models. Relying solely on the ability of modern models to extract information from unstructured code does not take advantage of 70 years of programming language and compiler development by not utilizing the structure inherent to programs in the data collection. This detracts from the performance of models working over a tokenized representation of input code and precludes the use of these models in the compiler itself. To work towards the first intermediate representation (IR) based models, we fully utilize the LLVM compiler infrastructure, shared by a number of languages, to generate a 182B token dataset of LLVM IR. We generated this dataset from programming languages built on the shared LLVM infrastructure, including Rust, Swift, Julia, and C/C++, by hooking into LLVM code generation either through the language's package manager or the compiler directly to extract the dataset of intermediate representations from production grade programs. Statistical analysis proves the utility of our dataset not only for large language model training, but also for the introspection into the code generation process itself with the dataset showing great promise for machine-learned compiler components.

2 Datasheet

Motivation

For what purpose was the dataset created? Was there a specific task in mind? Was there a specific gap that needed to be filled? Please provide a description.

The dataset was created to enable large scale analysis of existing compiler techniques and to provide a large, representative set of training data for the next generation of compiler-focused ML models.

Who created this dataset (e.g., which team, research group) and on behalf of which entity (e.g., company, institution, organization)?

The dataset was created by a cross-institutional collaboration including members from UC Davis, Technical University of Munich, Lawrence Livermore National Laboratory, University of Minnesota, University of Illinois at Urbana Champaign, Argonne National Laboratory, and Google. The primary entity funding the development of ComPile is Lawrence Livermore National Laboratory.

Who funded the creation of the dataset? If there is an

*Corresponding author

associated grant, please provide the name of the grantor and the grant name and number.

This work was in parts prepared by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Any other comments?

None.

Composition

What do the instances that comprise the dataset represent (e.g., documents, photos, people, countries)? Are there multiple types of instances (e.g., movies, users, and ratings; people and interactions between them; nodes and edges)? Please provide a description.

Each instance within the dataset represents a single LLVM bitcode module. Depending upon the source language, this could represent a single translation unit like in C/C++, a single target as in Rust, or a package as in Julia.

How many instances are there in total (of each type, if appropriate)?

ComPile contains approximately 670,000 modules in its current form. module count

Does the dataset contain all possible instances or is it a sample (not necessarily random) of instances from a larger set? If the dataset is a sample, then what is the larger set? Is the sample representative of the larger set (e.g., geographic coverage)? If so, please describe how this representativeness was validated/verified. If it is not representative of the larger set, please describe why not (e.g., to cover a more diverse range of instances, because instances were withheld or unavailable).

The dataset is a sample of all possible instances. The sample is designed to be representative of bitcode modules that are present in widely-used production applications, but we do not currently have results quantifying how representative ComPile is of this population.

What data does each instance consist of? "Raw" data (e.g., unprocessed text or images) or features? In either case, please provide a description.

Each instance consists of an LLVM bitcode module in binary form along with associated provenance information, including the project that it was sourced from and the license information associated with that project.

Is there a label or target associated with each instance? If so, please provide a description.

No, there is not a label or target associated with each instance in the distributed form of the dataset.

Is any information missing from individual instances? If so, please provide a description, explaining why this information is missing (e.g., because it was unavailable). This does not include intentionally removed information, but might include, e.g., redacted text.

No, no information is missing from individual instances. All instances contain a complete, valid LLVM module and all associated provenance informtion.

Are relationships between individual instances made explicit (e.g., users' movie ratings, social network links)? If so, please describe how these relationships are made explicit.

Some relationships are made explicit. The per-project provenance information is included directly in each instance, allowing instances to be grouped by their source. However, other relationships such as the targets that modules get linked into are not preserved.

Are there recommended data splits (e.g., training, development/validation, testing)? If so, please provide a description of these splits, explaining the rationale behind them.

There is no recommended data split. The best data split will be highly dependent upon the specific downstream application that the user is interested in.

Are there any errors, sources of noise, or redundancies in the dataset? If so, please provide a description.

Many of the modules will contain some of the same functions, but no entire module will be exactly the same. All of the modules are parseable using an up-to-date LLVM toolchain, but some of them might fail to run through the optimization pipeline or timeout when performing certain analyses due to bugs in various parts of the toolchain.

Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)? If it links to or relies on external resources, a) are there guarantees that they will exist, and remain constant, over time; b) are there official archival versions of the complete dataset (i.e., including the external resources as they existed at the time the dataset was created); c) are there any restrictions (e.g., licenses, fees) associated with any of the external resources that might apply to a future user? Please provide descriptions of all external resources and any restrictions associated with them, as well as links or other access points, as appropriate.

The dataset is entirely self-contained and does not rely on any external resources for its useability.

Does the dataset contain data that might be considered confidential (e.g., data that is protected by legal privilege or by doctor-patient confidentiality, data that includes the content of individuals non-public communications)? If so, please provide a description.

The data included in ComPile is obtained from publicly available sources and thus will not contain any confidential data that was not already broadly available. However, we cannot be certain that it does not contain any confidential information. Our dataset construction techniques should remove most cases of confidential information if they are present, like data in project repositories and comments, but does not eliminate all cases, like data embedded in code.

Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety? If so, please describe why.

We believe that it is unlikely that any representation of the IR contained within our dataset would have any of these properties.

Does the dataset relate to people? If not, you may skip the remaining questions in this section.

No.

Does the dataset identify any subpopulations (e.g., by age, gender)? If so, please describe how these subpopulations are identified and provide a description of their respective distributions within the dataset.

N/A.

Is it possible to identify individuals (i.e., one or more natural persons), either directly or indirectly (i.e., in combination with other data) from the dataset? If so, please describe how.

N/A.

Does the dataset contain data that might be considered sensitive in any way (e.g., data that reveals racial or ethnic origins, sexual orientations, religious beliefs, political opinions or union memberships, or locations; financial or health data; biometric or genetic data; forms of government identification, such as social security numbers; criminal history)? If so, please provide a description.

N/A.

Any other comments?

None.

Collection Process

How was the data associated with each instance acquired? Was the data directly observable (e.g., raw text, movie ratings), reported by subjects (e.g., survey responses), or indirectly inferred/derived from other data (e.g., part-of-speech tags, model-based guesses for age or language)? If data was reported by subjects or indirectly inferred/derived from other data, was the data validated/verified? If so, please describe how.

After building each piece of software included in the dataset, the data was directly available. The only information associated with each instance is provenance information, which was also readily collected. No labels or targets are included with each instance.

What mechanisms or procedures were used to collect the data (e.g., hardware apparatus or sensor, manual human curation, software program, software API)? How were these mechanisms or procedures validated?

We wrote a custom suite of tooling available at https://zenodo.org/doi/10.5281/zenodo.10155760. The software pulls lists of software from specific package indices, attempts to build all the specified software, and extracts LLVM-IR in bitcode form from all of the software that successfully built.

If the dataset is a sample from a larger set, what was the sampling strategy (e.g., deterministic, probabilistic with specific sampling probabilities)?

We included as much IR as possible from package indices where it was feasible to automatically build all the packages. There was a significant portion of builds that failed and we also omitted multiple package indices that we believed would be more difficult to automate collection from. In addition, there are many projects that are not included in package repositories that the techniques we used to collect this dataset would not be able to obtain.

Who was involved in the data collection process (e.g., students, crowdworkers, contractors) and how were they compensated (e.g., how much were crowdworkers paid)?

All collection was automated.

Over what timeframe was the data collected? Does this timeframe match the creation timeframe of the data associated with the instances (e.g., recent crawl of old news articles)? If not, please describe the timeframe in which the data associated with the instances was created.

The most recent collection of the dataset was performed in early November of 2023. This version of the dataset was collected with the most up to date versions of the package indices available at the time and the latest release or nightly version of language specific toolchains depending upon the specific language.

Were any ethical review processes conducted (e.g., by an institutional review board)? If so, please provide a description of these review processes, including the outcomes, as well as a link or other access point to any supporting documentation.

No.

Does the dataset relate to people? If not, you may skip the remaining questions in this section.

No.

Did you collect the data from the individuals in question directly, or obtain it via third parties or other sources (e.g., websites)?

N/A.

Were the individuals in question notified about the data collection? If so, please describe (or show with screenshots or other information) how notice was provided, and provide a link or other access point to, or otherwise reproduce, the exact language of the notification itself.

N/A.

Did the individuals in question consent to the collection and use of their data? If so, please describe (or show with screenshots or other information) how consent was requested and provided, and provide a link or other access point to, or otherwise reproduce, the exact language to which the individuals consented.

N/A.

If consent was obtained, were the consenting individuals provided with a mechanism to revoke their consent in the future or for certain uses? If so, please provide a description, as well as a link or other access point to the mechanism (if appropriate).

N/A.

Has an analysis of the potential impact of the dataset and its use on data subjects (e.g., a data protection impact analysis) been conducted? If so, please provide a description of this analysis, including the outcomes, as well as a link or other access point to any supporting documentation.

N/A.

Any other comments?

None.

Preprocessing/cleaning/labeling

Was any preprocessing/cleaning/labeling of the data done (e.g., discretization or bucketing, tokenization, part-of-speech tagging, SIFT feature extraction, removal of instances, processing of missing values)? If so, please provide a description. If not, you may skip the remainder of the questions in this section.

We performed a deduplication step based on LLVM's `StructuralHash` to remove duplicate modules from the dataset. In addition, we filtered by project to only include projects with permissive licensing.

Was the "raw" data saved in addition to the preprocessed/cleaned/labeled data (e.g., to support unanticipated future uses)? If so, please provide a link or other access point to the "raw" data.

The raw data was saved, but there are no plans to release it. There are license constraints imposed by the source projects that we need to abide by, restricting us from publishing a completely unfiltered version. In addition, while LLVM's `StructuralHash` is slightly lossy, almost all of the duplicates it identifies are accurate, and publishing a dataset that contains many duplicate modules provides little additional value.

Is the software used to preprocess/clean/label the instances available? If so, please provide a link or other access point.

Yes. All of the software used to process the dataset is available at (llvm-ir-dataset-utils link).

Any other comments?

None.

Uses

Has the dataset been used for any tasks already? If so, please provide a description.

We have used the dataset internally to perform some tasks, including training large language models, to good effect. The results for our usage here are currently unpublished.

Is there a repository that links to any or all papers or systems that use the dataset? If so, please provide a link or other access point.

No, there is not currently a repository that links to all users of the dataset.

What (other) tasks could the dataset be used for?

In addition to large-scale training of machine learning models, we also believe the dataset could be valuable for large-scale analyses of existing and classical compilation techniques.

Is there anything about the composition of the dataset or the way it was collected and preprocessed/cleaned/labeled that might impact future uses? For example, is there anything that a future user might need to know to avoid uses that could result in unfair treatment of individuals or groups (e.g., stereotyping, quality of service issues) or other undesirable harms (e.g., financial harms, legal risks) If so, please provide a description. Is there anything a future user could do to mitigate these undesirable harms?

The dataset was collected from a corpus of software at a specific point in time and with specific toolchain versions. There have been significant changes in the past like the migration to opaque pointers that significantly impact how the IR looks. Major, and even relatively minor changes in LLVM and the language frontends should be analyzed before using ComPile to ensure that the data is representative.

Are there tasks for which the dataset should not be used? If so, please provide a description.

The dataset is currently not representative of all languages that contain an LLVM frontend. For example, we do not include any IR from Fortran. Language specific tasks where the language is not represented in ComPile should not currently be performed using ComPile.

Any other comments?

None.

Distribution

Will the dataset be distributed to third parties outside of the entity (e.g., company, institution, organization) on behalf of which the dataset was created? If so, please provide a description.

Currently, we are pushing the dataset through an internal review process. We will upload it to HuggingFace after the review is complete where it will be available at https://huggingface.co/datasets/llvm-ml/ComPile. Sadly, we can not provide an exact timeframe for when the dataset will be publicly available.

How will the dataset will be distributed (e.g., tarball on website, API, GitHub) Does the dataset have a digital object identifier (DOI)?

The dataset will be available for download on the HuggingFace hub https://huggingface.co/datasets/llvm-ml/ComPile after review approval.

When will the dataset be distributed?

Late 2023.

Will the dataset be distributed under a copyright or other intellectual property (IP) license, and/or under applicable terms of use (ToU)? If so, please describe this license and/or ToU, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms or ToU, as well as any fees associated with these restrictions.

Users of the dataset will need to comply with the licenses of the individual projects that compose ComPile. The authors of ComPile do not impose any additional restrictions on users of the dataset.

Have any third parties imposed IP-based or other restrictions on the data associated with the instances? If so, please describe these restrictions, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms, as well as any fees associated with these restrictions.

Yes, there are restrictions based on the licenses that the source projects that compose ComPile use. These include attribution for distribution of the data in verbatim form. ComPile only includes projects that are licensed under the `MIT` license, the `Apache-2.0` license, the `BSD-3-Clause` license, and the `BSD-2-Clause` license. The exact terms for each license can be found on the OSI's website at https://opensource.org/licenses/.

Do any export controls or other regulatory restrictions apply to the dataset or to individual instances? If so, please describe these restrictions, and provide a link or other access point to, or otherwise reproduce, any supporting documentation.

No.

Any other comments?

None.

Maintenance

Who will be supporting/hosting/maintaining the dataset?

The authors will be continuing to maintain and support the dataset. It will be hosted on HuggingFace.

How can the owner/curator/manager of the dataset be contacted (e.g., email address)?

The authors of the dataset can be contacted utilizing the contacts listed in the list of authors above.

Is there an erratum? If so, please provide a link or other access point.

No.

Will the dataset be updated (e.g., to correct labeling errors, add new instances, delete instances)? If so, please describe how often, by whom, and how updates will be communicated to users (e.g., mailing list, GitHub)?

The dataset will be updated periodically to contain the latest versions of the packages currently included in the dataset, any new packages added to the package indices that the data is pulled from. More package indices might also be added in the future. Additionally, updates of the dataset will be built against the latest version of the toolchain available for each specific language to better represent the contemporary distribution of IR.

If the dataset relates to people, are there applicable limits on the retention of the data associated with the instances (e.g., were individuals in question told that their data would be retained for a fixed period of time and then deleted)? If so, please describe these limits and explain how they will be enforced.

The dataset does not relate to people.

Will older versions of the dataset continue to be supported/hosted/maintained? If so, please describe how. If not, please describe how its obsolescence will be communicated to users.

Yes, older versions of the dataset will be available on HuggingFace to enable comparative analysis over time.

If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so? If so, please provide a description. Will these contributions be validated/verified? If so, please describe how. If not, why not? Is there a process for communicating/distributing these contributions to other users? If so, please provide a description.

The tooling to construct the dataset is open source and available at https://zenodo.org/doi/10.5281/zenodo.10155760.

Any other comments?

None.