Can Data Mining Help to Survive the Annual Compiler Upgrade?

Gunnar Kudrjavets
Amazon Web Services*
Seattle, WA, USA
gunnarku@amazon.com

Aditya Kumar
Google
Mountain View, CA, USA
appujee@google.com

Piotr Przymus
Nicolaus Copernicus University
Toruń, Poland
piotr.przymus@mat.umk.pl

Abstract

Modern compilers provide improved diagnostics, performance, and security. The industry lacks the data and tools to estimate the cost to upgrade a compiler toolchain for complex projects. A knowledge base mined from defect databases, mailing lists, experience reports, commits, and grey literature will improve the planning process.

ACM Reference Format:
Gunnar Kudrjavets, Aditya Kumar, and Piotr Przymus. 2026. Can Data Mining Help to Survive the Annual Compiler Upgrade?. In . ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Background and motivation

Software projects need to periodically upgrade compiler toolchains to make use of (a) improvements in programming languages (enhanced error detection, new features, library updates), (b) performance benefits (code generation, CPU-specific optimizations), and (c) meet the requirements from the dependencies already upgraded [10].

Estimating the cost of modernizing a specific project is complex because compiler upgrades are a multilayered problem. Compilers do not show all compilation errors at once, syntax errors must be fixed before linker errors, and code generation performance can only be measured after the previous steps have succeeded.

For multimillion line projects, each phase can take weeks, months, or years of engineering effort. There is no clear way to calculate the cost of upgrading a project P from a compiler toolchain version N to version $N+M$. The state-of-the-art estimation technique looks at the collection of release notes and similar past experiences, resulting in a *guesstimate*. Another problematic factor is the highly repetitive and unattractive nature of this work. Empirically, engineers prefer to work on new features versus fix linker errors.

As a result, complex software projects, such as operating system kernels, require maintaining support for compilers that are a decade old. For example, the Linux kernel changed a required minimal GNU Compiler Collection (GCC) version only in April 2025 from GCC 5.1.0 (released in April 2015) to GCC 8.1.0 (released in May 2018) [1]. Although newer compiler versions can still be used to build the kernel, this constraint means that the kernel code cannot rely on modern language features or optimizations introduced after the minimal GCC version. That limits the ability of kernel engineers to use newer C standards or compiler-assisted safety checks.

*Conducting research is not related to Gunnar Kudrjavets' role at Amazon Web Services, Inc. All opinions and statements in this paper are the author's own.

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 Availability of data

The most popular open-source compiler toolchains are Clang and GCC. Clang has a six-month release cycle [6]. The GCC releases follow an annual cadence [3]. A list of notable changes, porting guidelines, and release notes accompany each release of Clang and GCC. Developers and users also share relevant information in the Discord groups, mailing lists, and Bugzilla defect databases. The history of changes in source code is tracked in Git or Subversion. In addition, the compiler test suites expose the defects that affect a specific compiler version [2].

3 Industry's needs

The high velocity of compiler updates is an important topic for warehouse-scale applications that serve billions of users. Empirical findings from Google, Meta and Uber show the importance of using the latest compiler toolchains to improve performance (e.g., applying link-time or profile-guided optimizations), reduce operating costs, and decrease energy consumption [4, 5, 8, 11].

To efficiently plan a compiler toolchain upgrade, engineers need to answer the following questions.

(1) What are the common problems encountered when upgrading software from the toolchain version N to version $N+M$? For example, migration from GCC 14.3.0 to GCC 15.2.0? A similar but more complex scenario is to migrate to a different toolchain altogether or to support multiple toolchains.
(2) How many resources (computing power, people, time) are needed to be allocated for upgrading a particular project? Are the estimates from the past reliable, given the AI-induced increase in the code velocity during the past 1–2 years?
(3) In the spirit of "using AI for everything," can we use AI agents and LLMs for toolchain upgrades? Given the growing corpus of research on automated code migration, can we reduce this problem to a subset of code migration [7, 9, 12]?

A small number of organizations with access to the necessary computing power have the means to use LLMs to assist with compiler updates. However, that approach does not solve the problem for numerous open-source software projects that are mainly maintained by volunteers and supported by donations.

We postulate that the collection of data sources enumerated in Section 2 contains an unstructured data set that, when categorized and publicized, can be used to (a) help engineers find the answers faster, (b) train models targeted for solving similar problems, and (c) raise awareness of the developer time and labor costs associated with toolchain upgrades.

References

[1] Arnd Bergmann. 2025. *kbuild: require gcc-8 and binutils-2.30.* Retrieved November 14, 2025 from https://github.com/torvalds/linux/commit/118c40b7

[2] Junjie Chen, Jibesh Patra, Michael Pradel, Yingfei Xiong, Hongyu Zhang, Dan Hao, and Lu Zhang. 2020. A Survey of Compiler Testing. *ACM Comput. Surv.* 53, 1, Article 4 (Feb. 2020), 36 pages. doi:10.1145/3363562

[3] Free Software Foundation, Inc. 2025. *GCC Releases.* Retrieved November 14, 2025 from https://gcc.gnu.org/releases.html

[4] Wenlei He, Hongtao Yu, Lei Wang, and Taewook Oh. 2024. Revamping Sampling-Based PGO with Context-Sensitivity and Pseudo-instrumentation. In *Proceedings of the 2024 IEEE/ACM International Symposium on Code Generation and Optimization* (Edinburgh, United Kingdom) *(CGO '24)*. IEEE Press, 322–333. doi:10.1109/CGO57630.2024.10444807

[5] Kyungwoo Lee, Ellis Hoag, and Nikolai Tillmann. 2022. Efficient profile-guided size optimization for native mobile applications. In *Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction* (Seoul, South Korea) *(CC 2022)*. Association for Computing Machinery, New York, NY, USA, 243–253. doi:10.1145/3497776.3517764

[6] LLVM Project. 2025. *LLVM Download Page.* Retrieved November 14, 2025 from https://releases.llvm.org/

[7] Behrooz Omidvar Tehrani, Ishaani M, and Anmol Anubhai. 2024. Evaluating Human-AI Partnership for LLM-based Code Migration. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI EA '24)*. Association for Computing Machinery, New York, NY, USA, Article 133, 8 pages. doi:10.1145/3613905.3650896

[8] Han Shen, Krzysztof Pszeniczny, Rahman Lavaee, Snehasish Kumar, Sriraman Tallam, and Xinliang David Li. 2023. Propeller: A Profile Guided, Relinking Optimizer for Warehouse-Scale Applications. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 617–631. doi:10.1145/3575693.3575727

[9] Ty Smith. 2025. *Large Scale Changes with AI—Migrating millions of lines of Java to Kotlin at Uber.* Uber Technologies, Inc. Retrieved November 14, 2025 from https://www.youtube.com/watch?v=K2PN03AepC0

[10] Jialiang Tan, Shuyin Jiao, Milind Chabbi, and Xu Liu. 2020. What every scientific programmer should know about compiler optimizations?. In *Proceedings of the 34th ACM International Conference on Supercomputing* (Barcelona, Spain) *(ICS '20)*. Association for Computing Machinery, New York, NY, USA, Article 42, 12 pages. doi:10.1145/3392717.3392754

[11] Chris Zhang, Yufan Xu, Milind Chabbi, and Shauvik Roy Choudhary. 2025. *Automating Efficiency of Go programs with Profile-Guided Optimizations.* Uber Technologies, Inc. Retrieved November 14, 2025 from https://www.uber.com/blog/automating-efficiency-of-go-programs-with-pgo/

[12] Celal Ziftci, Stoyan Nikolov, Anna Sjövall, Bo Kim, Daniele Codecasa, and Max Kim. 2025. Migrating Code At Scale With LLMs At Google. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering* (Clarion Hotel Trondheim, Trondheim, Norway) *(FSE Companion '25)*. Association for Computing Machinery, New York, NY, USA, 162–173. doi:10.1145/3696630.3728542