Why more software development needs to go to the machines

28 October 2019

Justin Gottschlich of Intel Labs leads a team of machine programming researchers. Their aim is to automate software development to reduce coding errors and address a shortage of trained expert programmers. Credit: Walden Kirsch/Intel Corporation

Our expert: Justin Gottschlich leads the Machine Programming Research (MPR) team in the Systems and Software Research Lab. Justin's newly-formed research group focuses on the pioneering promise of machine programming, which is a fusion of machine learning, formal methods, programming languages, compilers and computer systems.

His simple explanation of machine programming: MPR uses forms of machine learning and other automatic methods to create software capable of creating its own software. It's called machine programming and is fundamentally about automating software development and maintenance. When fully realized, machine programming will enable everyone to express their creativity and develop their own software without writing a single line of code.

Machine programming's promise: In today's technological landscape, software is integrated into almost everything we do. It controls many aspects of our mobile devices—laptops, tablets, phones. It connects us to the internet and drives our social media feeds. It virtualizes our data centers and makes our homes more intelligent. But developing and maintaining software is a time-consuming and error-prone process, Justin says. "I believe we can create a society where everyone can create software, but machines will handle the 'programming' part," he says. "Thus, 'machine programming.'"

A shortage of human programmers: A core problem for Intel and other leading tech companies, according to Justin, is that they are running low on senior developers—a shortage that crimps the amount of programming across all industries. According to code.org, there are 500,000 open programming positions available in the U.S. alone—compared with an annual crop of 50,000 graduating computer science majors. A similar shortage can be found across the European Union. In the programming jobs market, Justin says, at best only 10% of the people filling those jobs have the computer science training to become top-level advanced developers. With today's heterogeneous hardware—CPUs, GPUs, FPGAs, ASICs, neuromorphic and, soon, quantum chips—it will become difficult, perhaps impossible, to find developers who can correctly, efficiently, and securely program across all of that hardware.

Now is the time: Machine programming is a fusion of different fields. It uses automatic programming technique, from precise (e.g., formal program synthesis) to probabilistic (e.g., differentiable programming) methods. It also uses and learns from everything we've built in hardware and software to date. Researchers have dabbled in machine programming since the 1950s, Justin says. "But today is different. We're at an inflection point with new machine learning algorithms, new and improved hardware, and rich and dense

programming data. These are the three essential ingredients that we believe enable machine programming." One example is illustrated by recent genetic algorithm (GA) research from Justin's team, which illustrates how the fitness function of a genetic algorithm—a complicated machine learning heuristic developed by expert programmers—can be automated. Justin says this work likely wouldn't have been possible just a few years ago.

Refusing to accept bugs: Almost all large-scale software today (e.g., operating systems, browsers, social media platforms) includes accuracy, performance or security bugs. "Our latest NeurIPS '19 paper provides early evidence that certain types of bugs that have historically evaded even expert programmer detection can be automatically detected with machine programming, requiring zero human intervention," Justin says. "The next step is to automatically fix them."

From 500,000 lines of code to 500: Justin points to a well-known example of machine programming's benefits. Google Translate, a service that automatically translates among languages, was built by engineers who hand-coded around 500,000 lines using classical programming techniques. With the advent of machine programming, Google rewrote its code, partially using differentiable programming (one small slice of the overall machine programming pie). That rewrite decreased the code base from 500,000 lines to 500 lines, a 1,000x reduction. "Not only did the code size shrink by 1,000 times," Justin says, "the accuracy of the system actually improved—it's incredible."

More programming jobs, not fewer: Machine programming will not eliminate jobs, Justin contends, but instead create them—possibly millions of them. The more menial aspects of programming will be automated, he says, which is the goal. With machine programming, he adds, "our blue-sky vision is so long as you can express your ideas (as we call it—intention) in some way that the machine can recognize—be it natural language, visual diagrams or gestures—machine programming builds a path for you to create your own software." To begin building these advanced machine programming systems, Justin says, we'll rely heavily on a community of programmers and scientists—those who can work across platforms, machine learning and formal techniques, heterogeneous hardware, and many programming languages. Justin and team outline their future vision of machine programming in a paper jointly published with MIT researchers, "The Three Pillars of Machine Programming."

More information: Justin Gottschlich et al. The three pillars of machine programming, *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages - MAPL 2018* (2018). DOI: 10.1145/3211346.3211355

Provided by Intel

APA citation: Why more software development needs to go to the machines (2019, October 28) retrieved 19 September 2021 from https://techxplore.com/news/2019-10-software-machines.html