Abstract
Most research tools are publicly available but rarely used due to the difficulty of building them, which hinders the sharing of ideas. However, web browsers have recently become an excellent platform for giving portable demos. Researchers could enable their tools to run on a web browser by compiling them into JavaScript. Making tools accessible on the web would facilitate trying new tools without the overhead of configuring and building them.

1. Motivation
When Computer Scientists create tools in the course of their research, they often publicly release them. This is a useful practice, as it makes their findings reproducible, allows other researchers to evaluate and build on their ideas, and potentially enables real users to use a practical tool. Conferences have begun to encourage authors to submit artifacts to be evaluated along with their papers [1, 3, 8].

Unfortunately, even when such tools are publicly available, they are often difficult to build. Finding all the dependencies needed to build a program can require significant effort, and when a binary is available, it can be difficult to figure out how to use it. A recent conference that encouraged authors to submit artifacts stipulated that the setup should take less than 30 minutes [8], which is still a significant amount of time for reviewers to spend on a single paper.

Recently, the web has become a powerful application platform that makes it easy to run programs in a portable fashion. Some research projects have made online versions of their tools available, which has made it easy for others to try them out (e.g., [4, 9]).

Unfortunately, the current practice for making research tools available online requires that researchers set up and run their own servers. This can be a difficult and costly process that also imposes a limit on the number of concurrent users to reduce load.1 In addition, security is a problem with researchers running code on their own servers. Most research tools are not designed to be secure, so running them with arbitrary inputs from the web could be dangerous.

We propose facilitating researchers to compile their tools to JavaScript, which would allow anyone with a web browser to use them. This would allow researchers to make their tools usable online without running their own server or limiting the number of users. Since our approach compiles code to JavaScript that runs on the client, security is not a problem.

2. Approach
We propose building an infrastructure that makes it easy both to take an existing project and compile it to JavaScript and to put the JavaScript onto a webpage for users to visit.

2.1 Translation to JavaScript
We plan to leverage existing tools that translate programs into JavaScript. For example, Emscripten [18] compiles LLVM [17] bitcode to JavaScript. It contains wrappers around Clang (LLVM’s C/C++ frontend) that allow it to compile C/C++ code directly to JavaScript. Furthermore, interpreters for many languages, including Python and Ruby, have been compiled with Emscripten and could be used to run code written in those languages. Finally, separate projects for many other languages, including Scala, Haskell, and OCaml, aim to generate JavaScript code directly [2, 7]. As a last resort, projects could be run in a JavaScript PC emulator [11].

For this paper, we focus on Emscripten. It targets asm.js [12], a low-level subset of JavaScript that is designed to allow execution at close to native speeds. Many projects have been ported to JavaScript with Emscripten, including the Unreal Engine 3 [10], LaTeX [6], Lua, Python, JavaScript, and parts of LLVM and Emscripten themselves [2, 7]. It is thus reasonably robust, and given its emphasis on porting games to the web, performs reasonably well.

2.2 Other Issues to Consider
There are many issues to consider when translating research projects into JavaScript. We discuss some of them below.

Binaries Many projects rely on closed-source binaries. In theory, it might be possible to compile assembly code to...
we now briefly discuss the status of each project. We have analyzed the performance of our compiled versions of MiniSat, Lingeling, and Boolector on a few benchmarks. On average, the Emscripten-compiled versions are 2.4-5.5x slower than native, which we believe is sufficient for our purposes. Furthermore, to show that performance is likely to improve further with more time, we compiled and tested MiniSat with six-month-old versions of Emscripten and Firefox and found that it was 11.4x slower than native.

We have also analyzed the build times and file sizes of Emscripten-compiled projects. Compile times are 2-7x slower and file sizes are 0.2-3x larger.

### Libraries
Projects often use many libraries, each of which would have to be compiled with the above technique. This is not a theoretical concern, as we would like to be able to compile all code, but it is a practical one, as compiling one research project might involve compiling many libraries used as dependencies. We believe that a central repository that hosted all known ported libraries would help alleviate this problem.

### Performance
A number of the techniques mentioned above, such as emulating shared-memory multithreading and running code in dynamic languages by running their interpreters in JavaScript, will likely be much slower than running the code natively. Luckily, performance is not critical for our domain, as most research tools are not optimized and online demos do not usually need to be as fast as possible. Thus unlike the games that asm.js targets, we will likely prioritize compatibility and ease-of-use over performance.

#### References


