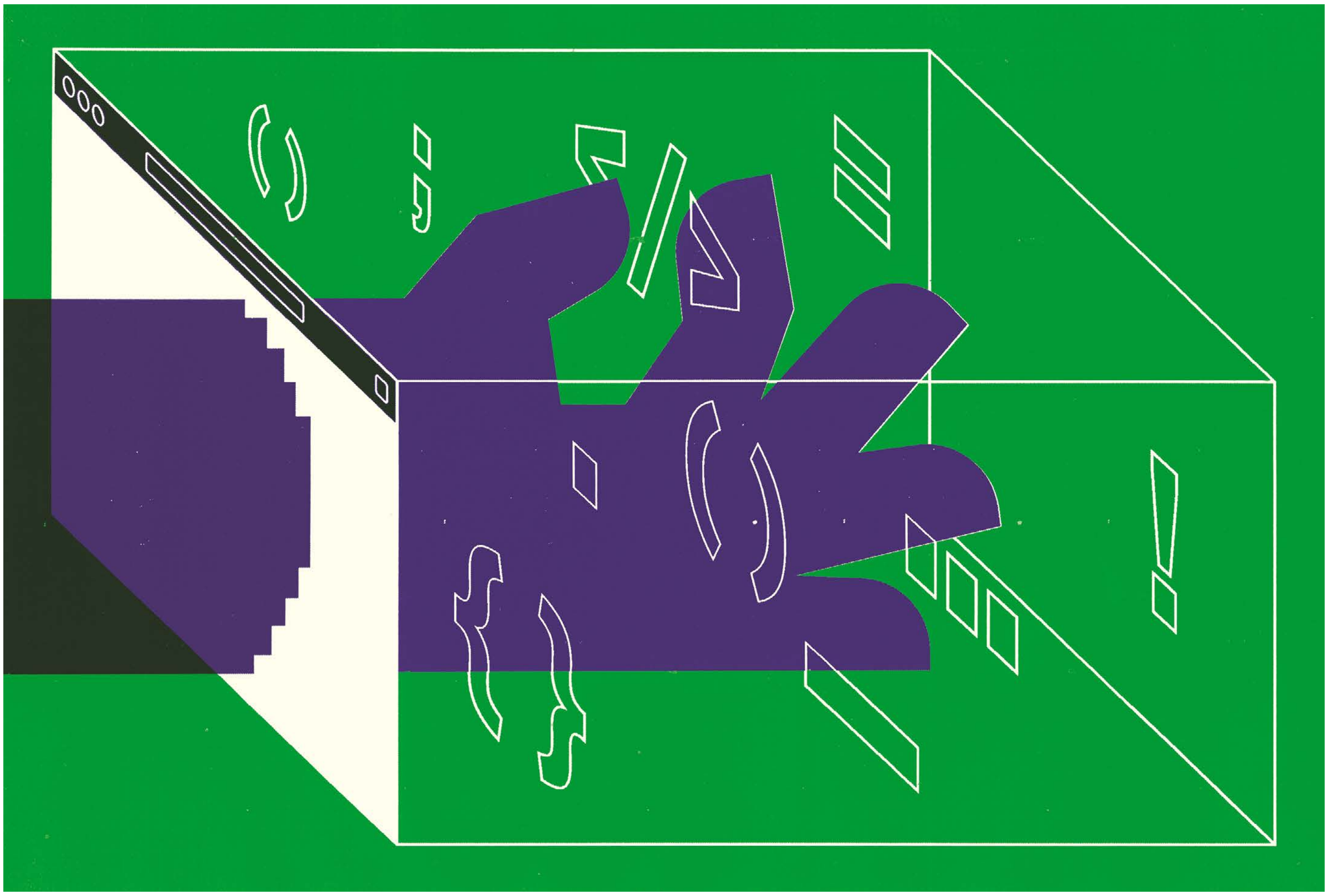

ILLUSTRATION: THE PROJECT TWINS

HOW WEBASSEMBLY IS CHANGING RESEARCH COMPUTING

Enabling code execution in the web browser, the multilanguage tool is powerful but complicated. **By Jeffrey M. Perkel**

In late 2021, midway through the COVID-19 pandemic, George Stagg was preparing to give exams to his mathematics and statistics students at the University of Newcastle, UK. Some would use laptops, others would opt for tablets or mobile phones. Not all of them could even use the programming language that was the subject of the test: the statistical language R. "We had no control, really, over what devices those students were using," says Stagg.

Stagg and his colleagues set up a server so that students could log in, input their code and automatically test it. But with 150 students trying to connect at the same time, the homegrown system ground to a halt. "Things were a little shaky," he recalls: "It was very, very slow."

Frustrated, Stagg spent the Christmas holidays devising a solution. R code runs in a piece of software called an interpreter. Instead of having students install the interpreter on their own computers, or execute their code on a remote server, he would have the interpreter run in the students' web browsers. To do that, Stagg used a tool that is rapidly gaining popularity in scientific computing: WebAssembly.

Code written in any of a few dozen languages, including C, C++ or Rust, can be compiled into the WebAssembly (or Wasm) instruction format, allowing it to run in a software-based environment inside a browser. No external servers are required. All modern browsers support WebAssembly, so code that works on one computer should produce the same result on any other. Best of all, no installation is needed, so scientists who are not authorized to install software – or lack the know-how or desire to do so – can use it.

WebAssembly allows developers to recycle their finely tuned code, so they don't have to rewrite it in the language of the web: JavaScript. Google Earth, a 3D representation of Earth from Google's parent company, Alphabet, is built on WebAssembly. So are the web version of Adobe Photoshop and the design tool Figma. Stagg, who is based in Newcastle but is now a senior software engineer at Posit, a software company in Boston, Massachusetts, solved his exam server issues by porting the R interpreter to WebAssembly in the webR package.

Daniel Ji, an undergraduate computer-science student in Niema Moshiri's laboratory at the University of California, San Diego, used WebAssembly to build browser interfaces for many of his group's epidemiological resources, including one that identifies evolutionary relationships between viral genomes[1]. Moshiri has used those tools to run analyses on smartphones, game systems and low-powered Chromebook laptops. "You might be able to have people run these tools without even needing a standard desktop or laptop computer," Moshiri says.

"They could actually maybe run it on some low-energy or portable device."

That being said, porting an application to WebAssembly can be a complicated process full of trial and error – and one that's right for only select applications.

Reusability and restrictions

Robert Aboukhalil's journey with WebAssembly began with an application that he created in 2017 for quality control of raw DNA-sequencing data. The necessary algorithms already existed in a tool called Seqtk, but they weren't written in JavaScript. So Aboukhalil, a software engineer at the Chan Zuckerberg Initiative in Redwood City, California, rewrote them – but his implementations were relatively slow. Retooling his application to use WebAssembly improved performance 20-fold. "It was awesome, because it gave me more features that I didn't have to write myself. And it happened to make the whole website a lot faster."

C and C++ code can be ported to WebAssembly using the free Emscripten compiler; Rust programmers can use 'wasm-pack', an add-on to Rust's package-manager and compilation utility, 'cargo'. Python and R code cannot be compiled into WebAssembly, but there are WebAssembly ports of their interpreters called Pyodide and webR, which can run scripting code in these languages.

Quarto, a publishing system that allows researchers to embed and execute R, Python and Javascript code in documents and slide decks, is compatible with WebAssembly, too, using the quarto-webr extension (see our example at go.nature.com/4c1ex). WebAssembly can also be used in Observable computational notebooks, which have uses in data science and visualization and run JavaScript natively. There's even a version of Jupyter, another computational-notebook platform, called JupyterLite that is built on WebAssembly.

Aboukhalil has ported more than 30 common computational-biology utilities to WebAssembly. His collection of 'recipes' – that is, code changes – that allow the underlying code to be compiled is available at biowasm.com. "Compiling things to WebAssembly, unfortunately, isn't straightforward," Aboukhalil explains. "You often have to modify the original code to get around things that WebAssembly doesn't support."

For instance, modern operating systems can handle 64-bit memory addresses. WebAssembly, however, is limited to 32 bits, and can access only 2^{32} bytes (4 gigabytes) of memory. Furthermore, it cannot directly access a computer's file system or its open network connections. And it's not multithreaded; many algorithms depend on this form of parallelization, which allows different parts of a computation to be performed simultaneously. "A lot of older code won't compile into WebAssembly, because it assumes that it can do things that can't be done," Stagg says.

Compounding these challenges, scientific software sits atop a tower of interconnected libraries, all of which must be ported to WebAssembly for the code to run. Jeroen Ooms, a software engineer in Utrecht, the Netherlands, has ported roughly 85% of the R-universe project's 23,000 open-source R libraries to WebAssembly. But only about half of those actually work, he says, because some underlying libraries have not yet been converted.

Then, there's the process of web development. Bioinformaticians don't typically write code in JavaScript, but it is needed to create the web pages in which those tools will run. They also have to manually handle tasks such as shuttling data between the two language systems and freeing any memory they use – tasks that are handled automatically in pure JavaScript.

"WebAssembly is a zero-install solution. They just hit the URL, and they're good to go."

As a result, WebAssembly is often used to build relatively simple tools or applied to computationally intensive pieces of larger web applications. As a postdoc, bioinformatician Luiz Irber, then at the University of California, Davis, used WebAssembly to make a Rust language tool called Branchwater broadly accessible. Branchwater converts sequence data into numerical representations called hashes, which are used to search databases of microbial DNA sequences. Rather than having users install a conversion tool or upload their data to remote servers, Irber's WebAssembly implementation allows researchers to convert their files locally.

Bioinformatician Aaron Lun and software engineer Jayaram Kancherla at Genentech in South San Francisco, California, used WebAssembly to implement kana, a browser-based analysis platform for single-cell RNA-sequencing data sets. The goal, Lun and Kancherla say, was to allow researchers to explore their data without a bioinformatician's help. About 200 users now use kana each month.

The porting process took "six months, maybe a year's worth of weekends", Lun says, and was complicated by the fact that they were starting from C++ libraries glued together with R code. But that was nothing compared with the challenge of crafting a smooth, friendly user experience. "I can see why web developers get paid so much," he laughs.

Powering up

Developers who need more computing power can supercharge their tools through a related project, WebGPU, which provides access to users' graphics cards.

Will Usher, a scientific-visualization engineer at the University of Utah in Salt Lake City, and his team used WebGPU and WebAssembly to implement a data-visualization algorithm called 'Marching Cubes', with which they manipulated terabyte-scale data sets in a browser[2]. Computer scientist Johanna Beyer's team at Harvard University in Cambridge, Massachusetts, created a visualization tool for gigabyte-sized whole-slide microscopy data, using an algorithm called 'Residency Octree'[3]. And developers at UK firm Oxford Nanopore Technologies built Bonito, a drag-and-drop basecalling tool that translates raw signals into nucleotide sequences, for the company's sequencing platform.

Chris Seymour, Oxford Nanopore's vice-president of platform development, says the company's aim was to make its tools accessible to scientists who lack the skills to install software or are barred from doing so. Installation can be "a barrier to entry for certain users", he explains. But WebAssembly is "a zero-install solution": "They just hit the URL, and they're good to go."

There are other benefits, too. Data do not have to be transferred to external servers, alleviating privacy concerns. And because the browser isolates the environment in which WebAssembly code can be executed, it is unlikely to harm the user's system.

Perhaps most importantly, WebAssembly allows researchers to explore software and data with minimal friction, thus enabling development of educational applications. Aboukhalil has created a series of tutorials at sandbox.bio, with which users can test-drive bioinformatics tools in an in-browser text console. Statistician Eric Nantz at pharmaceuticals company Eli Lilly in Indianapolis, Indiana, is part of a pilot project to use webR to share clinical-trial data with the US Food and Drug Administration – a process that would otherwise require each scientist to install custom computational dashboards. Using WebAssembly, he says, "will minimize, from the reviewer's perspective, many of the steps that they had to take to get the application running on their machines".

WebAssembly, says Niema, "bridges that gap that we have in bioinformatics, where bio people are the users, computer-science people are the developers, and how do we translate [between them]?"

Still, brace yourself for complications. "WebAssembly is a great technology, but it's also a niche technology," Aboukhalil says. "There's a small subset of applications where it makes sense to [use it], but when it does make sense it can be very powerful. It's just a matter of figuring out which use cases those are."

Jeffrey M. Perkel is technology editor at *Nature*.

1. Ji, D. Aboukhalil, R. & Moshiri, N. *et al. Bioinformatics* **40**, btae018 (2024).
2. Usher, W. & Pascucci, V. Preprint at arXiv https://doi.org/10.48550/arXiv.2009.03254 (2020).
3. Herzberger, L. *et al.* Preprint at arXiv https://doi.org/10.48550/arXiv.2309.04393 (2023).

Correction
This Technology feature erroneously stated that data are never transferred to an external server with WebAssembly. In fact, they can be; it is just not required. Also, it wrongly stated that WebAssembly is limited to 32-bit numbers. It can handle 64-bit numbers, but is limited to 32-bit memory addresses.

Corrected 14 March 2024