



Contents

page 1: Unjust algorithms
page 3: Overcoming the hurdle of
“industry standard” software in
education technology
page 5: Old but not forgotten
page 7: The need for free software
education now
page 10: Verifying free software: The
basics

Unjust algorithms

By Zoë Kooyman
Executive Director

Developments in artificial intelligence (AI) injustices have rapidly taken a turn for the worse in recent years. Algorithmic decision-making systems are used more than ever by organizations, educational institutions, and governments looking for ways to increase understanding and make predictions. The Free Software Foundation (FSF) is working through this issue, and its many scenarios, to be able to say useful things about how this relates to software freedom. Our call for papers on Copilot was a first step in this direction (see: u.fsf.org/3i5).

Though complex, we are still talking about proprietary software systems which integrate AI. Often, they are algorithmic systems where only the inputs and outputs can be viewed. It is trained with a selection of base categories of information, after which information goes in and a verdict comes out — but what led to the conclusion is unknown. This makes AI systems less straightforwardly understandable, even by the people who wrote the code.

These systems (referred to as black box systems) can have the potential or intent to do good, but technology is not objective — and at the FSF, we believe that *all* software should be free. And when it comes to governments, they have the responsibility to demand for the



*FSF's Lemote
Yeelong, the
first fully free
software-
compatible
netbook
(see page 5).*

software they use to be free, and the public has a right to the software. The scale to which the increased use of artificial intelligence is affecting people's lives is immense, making this matter of computational sovereignty all the more urgent.

For regulators around the globe, the dangers involved in using AI haven't gone unnoticed either. In late 2018, both the United States and the European Union (EU) started working on obtaining guidance and forming regulations to deal with the proliferation and pervasive use of AI. In September 2021, Brazil's congress passed a bill that creates a legal framework for AI.

But the acknowledgment of the fast-paced integration of AI into our society without proper oversight didn't stop US tax agencies from recently trying to implement facial recognition for its systems. Worth exploring is the situation where 26,000 people were affected during the Dutch *toeslagenaffaire* (Dutch childcare benefits scandal) and 1,675 children were removed from their parents' custody during this scandal.

The Dutch tax office that regulates social benefits used an AI-based software system to automate the identification of errors and fraud. The system's training data blatantly violated privacy laws, and led to biased enforcement. It did so by flagging data points such as dual

nationality, low income, and "non-Western appearance" as big risk indicators for fraud. Because of people's blind trust in technology, this first system was used to then also teach another algorithm, this time affecting the childcare allowance unit.

The agencies, emboldened by the data provided by the system, ruthlessly penalized the families by withdrawing their aid and fining them tens of thousands — sometimes hundreds of thousands — of Euros. The debts ruined lives, and led to people losing their homes and their relationships. Some even lost custody over their children. The whole affair also led to the resignation of the entire Dutch cabinet. Citizens were discriminated against without their knowing, and they were unjustly denied knowledge as to why they were treated this way, nor given a chance to perform any research, to question the results of the system, or to defend themselves at any point of the process.

This is just one story of many that shows us what is at stake, and it shows the snowballing and disastrous effects of the lack of free software in government, while also revealing our lack of understanding of the consequences of using machine learning. In the EU, regulators have taken note of this scandal as a warning. European Commission executive vice-president, Margrethe

Vestager, said the *toeslagenaffaire* is “exactly what every government should be scared of.” In proposed legislation, they speak of adding checks and balances conducted by humans, and the European AI Act thus far proposes a restriction of the use of so-called “high-risk” AI systems and banning certain “unacceptable” uses that would protect people from such a scenario through what they call a “pyramid of risks.”

But it is not enough. Without the freedom to inspect the source code (which includes the software’s algorithms), we create a loophole where the organization ends up policing its own actions. When a system is nonfree, the argument for “enforcement” of regulations can never truly be made. What the draft legislation typically lacks is the central argument that software should be free (as in freedom). We do not yet have an elegant definition as to what elements must be shared along with the program when we are talking about machine learning, but we know we need to be holding up the GNU General Public License’s (GPL) definition of source code as the preferred form for modification, and the importance of installation information, as the guiding lights for charting the right path. In this case, had it been made public what software and what identifiers were used to teach the system how to

identify its victims, it may have been given less space to cause this much harm. Systems that play a major role in how our lives unfold should offer the possibility to be checked, and checked again.

You can make a difference as these laws and regulations are taking shape. Many governments have initiatives now that are open to public feedback. Get informed on where artificial intelligence proprietary software systems are used and on its dangers, and demand that your government deploys free software and protects its computational sovereignty: government software *must* be free software and *must* be readily available for inspection by its citizens.



Overcoming the hurdle of
“industry standard” in
education technology
By Michael McMahon
Web Developer

Educators who try to teach free software in education commonly run into an artificial barrier that usually sounds something like, “We cannot use that software because it is not industry standard.” This situation is a self-fulfilling prophecy. The institution expects proprietary software to be used in the workplace, the institution teaches students proprietary software, and the students end up using proprietary

software throughout their lives. The prophecy is fulfilled.

Quality free software tools are often perceived to be lacking in the fields of design, audio, and engineering. There is always room for improvement, but typically a free tool exists already.

For most 2D graphic design tasks, GNU Image Manipulation Program (GIMP), Krita, Inkscape, and ImageMagick will do the job. Most of the complaints regarding, for example, GIMP boil down to the fact that it does not work exactly the same as Adobe Photoshop — as in the menus, buttons, keyboard shortcuts, and selection interface are not exactly the same. However, these complaints are made from unreasonable expectations. If you know how to use one program, it is often possible to quickly adapt and learn the differences after watching a tutorial video. GIMP, Krita, Inkscape, and ImageMagick are worth the time investment to learn.

It is nearly always better to teach a process or approach than to teach to a specific piece of software because it teaches the concept instead of the motions. Moreover, just because something is perceived as industry standard does not mean that it ought to be, especially if it deprives students of their freedom. I have personally had success teaching multimedia creation processes using free software tools in

a well-known after-school program. I taught project-based learning through screen printing with vector graphics using Inkscape instead of Adobe Illustrator, photo editing using GIMP instead of Adobe Photoshop, drawing from a blank canvas with Krita instead of Adobe Photoshop, editing lots of photos in bulk on the command line with ImageMagick, video editing with Kdenlive instead of Final Cut Pro, and 3D modeling with Blender instead of AutoDesk Maya.

Professional video editing software is a particularly good example of why industry standards should be reevaluated from time to time as all nonlinear video editors work with approximately the same workflow. Of these, Blender specifically has broken the mold and become known as industry standard within the cinematic effects industry, all while respecting the user's




With free software, students are in control of their learning and can share their freedoms with others.

freedom.

Microsoft, Adobe, and other major companies offer their software to schools for free or at a reduced rate so that students will be unlikely to use anything else throughout their lives. Teachers often find this to be justifying, but they cannot share the software with the students at home or after they finish their education. Teaching with free software can help level the digital divide.

The beauty of free software is that every little piece or program that is swapped out improves the students' ability to study the environment that they use. Students are naturally inquisitive. When a student asks how something works, a teacher should be able to point to an answer, which is only possible if the software that they use is free as in freedom. With its focus on study and collaboration, free software is far more suited to the spirit of pedagogy than proprietary software, which causes dependence and inflicts abuse upon its users. Free software is the only kind of software that does so by allowing the student to run, copy, distribute, study, change, and improve the tools that they use.

Teaching with free software has been proven to work at Penn Manor School District in Pennsylvania, Kerala, India, as well as other places. It can work anywhere. Using free software in the classroom communicates the importance of

values key to a free society: sharing, social responsibility, and independence. If you find an instance where your school should be using free software instead of proprietary software, sign our *Give Students #UserFreedom* petition today (see: u.fsf.org/3m2)! 

Old but not forgotten

By Greg Farough
Campaigns Manager

Through a complicated series of events, I briefly ended up in a situation where my only working computer was one of the FSF's Lemote Yeeloongs, the first fully free software-compatible netbook ever released. Those who have been in the free software community for long enough will remember how coveted this machine was back in the "bad old days" of free software, when systems compatible with a free BIOS were even more rare than they are today. Even in 2008, just what it could do was severely limited by its single-core 800MHz processor and somewhat meager amount of memory. Videos online can also attest to its whopping three and a half minute boot time. All of this coupled with the Yeeloong's other quirks make it a good opportunity to reflect on the state of free software on older machines.

The *oldest* of old machines are in a tricky place when it comes to free software. Except in extremely rare circumstances, the days when the kernel Linux could fit on any kind of floppy disk are long behind us. Yet given that the “top of the line” when it comes to laptop freedom is a different machine released in 2008, the X200, and that most FSF staffers run Trisquel 10 on these machines, one would assume that the Yeeloong is similarly well-supported. Well, this isn’t the case. You see, the Yeeloong isn’t just an old machine. It’s an old machine on a very rare architecture: mipsel64.

As all GNU/Linux distributions and operating systems generally are limited in the number of CPU architectures they can target, this puts the Yeeloong at even more of a disadvantage when it comes to support. From the day of its release, it never supported nonfree operating systems like Windows, but now, even most free systems have left it behind. The Yeeloong I used ran an installation of Parabola GNU+Linux that is now too old to update, and while I sketched the outline of this article, support for the Yeeloong and related systems were dropped from OpenBSD, the only distribution of that operating system to have ever supported it. This leaves the Yeeloong in the lurch. Without significant manual intervention, it can no longer receive

security and software updates, and given its rarity, likely never will again.

This invites the question: what *can* the Yeeloong do nowadays? Recently, and for arguably good reasons, “retrocomputing” has come into the spotlight in certain sections of the free software community. Most of these experiments are centered around, or at least feature use of, the “small web” — old or alternative network protocols like Gopher, Gemini, or Spartan. Although it was something of a challenge to get a client compiled on the Yeeloong, it’s well suited to use cases like these. It’s no surprise that older protocols like Gopher pare a user’s browsing experience down to text, but Gemini, a protocol that is only just finalizing its written standard, is also a good way of keeping connected while avoiding what we’ve termed the JavaScript Trap (see: u.fsf.org/fb7). By design, Gemini pages are forbidden from executing programs on a user’s machine. Positioning itself as a middle point between a protocol as “lightweight” as Gopher and one as “heavy” as HTTP, Gemini takes the Web back to its roots in simple hypertext, albeit with some baked-in TLS and support for other media formats.

TLS? Therein lies the rub for these old machines. Even if the Yeeloong’s processor could support

the JavaScript transport behemoth known as the modern Web, without an up-to-date implementation of key security components like Pretty Good Privacy (PGP) or Transport Layer Security (TLS), it can't be trusted to escape any recent vulnerabilities that might compromise a user's privacy and security.

Too often, we're led to think that newer technology is intrinsically better than anything that came before it. If you're reading this *Bulletin*, you most likely know that this isn't the case. The advertisements put out by Apple and its ilk don't take the time to mention what advancements (or encroachments) it makes to your freedom or *ability* to do a particular task. If you splurge for that new computer, chances are that you'll be browsing the same Web sites, writing the same types of documents, and engaging with the same kinds of media as you did with the old one. Independent of the environmental or economic reasons why someone might stick with an older machine, there's an argument from simplicity: if the machine you're currently using does all that you need, why should you feel somehow "pressured" to buy a new one?

Naturally, paring down your computing environment doesn't need to involve purchasing a different set of hardware, even if that hardware's old and "outdated." Living the retro

experience can be as simple as choosing not to boot to X11 or Wayland, and getting as far as you can with your daily tasks on a shell like Bash running in the framebuffer. For every user who chooses to take this approach, there are at least ten laptops in varying states of repair languishing in the dumpster or electronics recycling shop. As most of my own machines are sourced this way, I should know! No matter what circumstances have brought you to that blinking cursor on that old machine, I hope you'll spare a thought to their upkeep, whether that's ensuring the program you develop is able to be compiled on old or rare architectures, or simply loading an old laptop up with GNU/Linux, and passing freedom on to a friend. 🐻

The need for free software education now

By Greta Goetz, PhD

Assistant Professor, University of Belgrade

Making the choice to use free software in the classroom helps us create learning environments that foreground questions of value. This can be illustrated by a passage in Confucius's *Analects* where the skilled mechanic is illustrated as sharpening their tools before they are able to do their work well. In the 21st century, and in the context of digital learning

environments, to sharpen tools would mean to know not just *how* the digital tool works but *what* the digital tool means with respect to the place of the human in the world. This will be explored by considering the five Ws: what free software education is, who it is for, when and where it takes place, and why our active role as digital makers, not just passive users, is central to the meaning of free software education.

What is free software education?

Most concretely, free software is a set of tools such as software packages and programming libraries, frameworks, and languages, which are released under a free software license. More abstractly, free software represents a set of values. These are the freedom to run, copy, distribute, study, change, and improve the software — the last of which is possible because it can be studied to begin with. Free software education teaches what the tool is not just in terms of its code but also its function: demonstrating what a human-centered tool looks like.

Free software teaches us the significance of the freedom to think and make informed decisions that champion human agency. It does so by offering myriad choices for personal or local customization and types of support. It allows us to question whether we really need to use locked-in software silos out of

“Convenience,” to cite an edifying poem by W. S. Merwin. It challenges us to see knowledge as something that is not passively downloaded but requires active engagement, discovery, and selection. To “sharpen our tools” is to learn something about learning and how it is embedded in social habit and institutional systems.

Who is the study of free software for?

Free software education is for everyone, not just programmers. As such, free software shares the learning values set out by educationalists such as Paolo Freire and John Dewey. Freire sought to create learning contexts where all students would feel motivated to generate contributions and not be overwhelmed by systemic power. He emphasized the importance of teaching students to make meaningful decisions rather than be carried away by the tides of the trends of the age. Dewey saw education as a cure for societal confusion and championed freedom of mind, rooted in freedom of action and experience in co-creating collectively valuable goods. Free software reminds us of our duty to think of the privilege (i.e. right, priority, law) of knowledge and to safeguard access to the encoded knowledge that informs and shapes our lives.

When is free software education?

Our journey up the freedom ladder (see: u.fsf.org/3f1) begins whenever we are ready to take

a step towards software freedom. Even if we know nothing about programming, we can still explore the value of free software. Some of these values have been considered above. Free software education, while centering on a set of values, also evolves with developments in computing.

Now is the time to think deeply about what free software means when artificial neural networks are being trained on data sets containing or referencing much of the transgenerational knowledge that we are born into, which is to say, the other forms of coding perpetuated through publication, training, education, and so forth. The question of free software is always current but is especially timely now that the largest artificial neural networks obscure sources while doing work, thus affecting access to knowledge.

Where does free software education take place?

The existence of free software invites us to build software starting from where we are, according to local needs. Tailoring tools does involve a time investment. However, this is an investment that teaches us about the principles behind the software and allows us more control over the data processed by the software. This means more deliberate decisions as to how software is configured in design that supports ongoing mutual

learning.

Free software education can take place in all learning environments, not just programming, and not just in the classrooms of any discipline. Learning to assemble digital tools in our lives such that they help us help each other continue to learn and share is an exercise in noticing what is worth being cared for and what is worth doing.

Why free software education?

The free software movement encourages us to find ways to leave digital traces on terms that respect collective understanding and contributory practices. Learning about free software gives experience in our generative potential and helps us learn to be makers, not just users, conscientiously assembling, if not also creating, our digital environments.

Like the Confucian mechanic, the person learning about free software becomes skilled in identifying, valuing, relating to, and championing the human place in a technical world. Understanding this can help a person extend this learning beyond the workplace to life in general.

Prompts for further conversation:

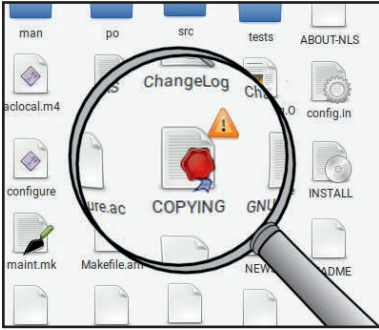
- 1) What does digital freedom mean to your own learning process?
- 2) How does free software teach the power of sharing?

Tag your posts as
#FreeSoftwareEducation! 🐼

Verifying free software: The basics

By Craig Topham

Copyright & Licensing Associate



So you found a program which you would like to use. Congratulations! Now what? How do you know if you can use this program in freedom? A little birdie told you it is free software, but how can you be sure your avian friend knows what they are talking about?

Joking aside, if you plan to use a program in freedom with the expectation you will be able to run, copy, distribute, study, change, and improve to it as you wish, you'll want to verify it is free software first. The easiest way to tell if you have a free program is to see if it is in the Free Software Directory. The Directory is a collection of over 16,000 free software programs which have been verified for freedom. If it is not in the Directory and you are new to the subject, this article will provide some basic tips in determining a program's license.

An interesting aspect of software licensing is that there is no single official way to properly license a program. The FSF certainly has its recommendations, and there are other groups with their own ideas as to how a program should be properly licensed. However, over the years (and as software freedom and licensing literacy has increased) some generally accepted methods have been established. This is not always the case, so if you have any doubts while trying to determine the licensing of a program, the best thing to do is to reach out to the maintainer of the program to clarify.

Once you have obtained a copy of a program's source code, the first step is to look for a file named README, which is the most common file name for the purpose of documenting the details of a program. It is typically found in the main directory of the source code. There are other names which might be used, such as RELEASE or RELEASE NOTES, but it is fair to assume that a file will be available with the program's details. A maintainer wishing for their program to be increasingly used and redistributed will provide all the details of not only the program's licensing, but also notes on dependencies, the project's history, contact information, a copyright notice and other information in the README file. It also might have

additional details about the licensing, including any additional permissions or restrictions determined by the copyright holder.

A copy of the license is also a file typically found in the main directory of the source code, but it may be stored in a sub-directory like `doc/` or `LICENSES/`. The title of the file may vary. Look for a name like `COPYING` or `LICENSE`, or it might be the name of the license itself like `GPL` or `APACHE`.

Once you have found the license, take a moment to examine it. It is important to make sure the license you have received has not been altered, because that could introduce legal issues and could even render the program nonfree. For your convenience, the FSF has a list of reviewed licenses with links to their corresponding license text so you can compare it with the license you received with the code. If it is an *entirely unaltered* GNU license, you are set — freedom secured! If it isn't a GNU license, the list also provides additional details of many licenses including GNU license compatibility. If you have any questions about the license you found, the licensing and compliance team, with the help of our dedicated and knowledgeable licensing volunteers, are available to answer your questions at licensing@fsf.org.

After successfully verifying the

program's license from the recommended steps above, there are some additional steps you should take. Check the licenses of dependencies, plug-ins, or other programs required to build and make use of the program. If we are lucky the maintainer of the program listed the dependencies in the README file. They may also be distributed with their source code, so look for directories titled “lib,” “resources,” “tools,” “plugins,” or “modules.” Essentially, you'll repeat the same steps in the other packages, starting with the README and LICENSE files. This can be a daunting task, but just be patient and tackle the list one project's source code at a time.

If you would like to learn more or practice examining a program's licensing, then join us at one of our weekly Free Software Directory where we do this collectively to review new packages to be added to the list. Meetings are held every Friday 12:00 to 15:00 EDT in `#fsf` on Libera Chat. All are welcome. 🐼

Get 10% off!

**Support the FSF by
purchasing FSF merchandise!**

**Visit shop.fsf.org and
enter discount code
SPRING2022, 6/15/22 - 9/1/22**



Donate to the FSF with Bitcoin:

15R987t9AoqRLxDkQ
evr2AVz8GSqvwMJvC

Copyright ©2022

Free Software Foundation, Inc.

The articles in this *Bulletin* are individually licensed under the Creative Commons Attribution-ShareAlike 4.0 International license.
<https://creativecommons.org/licenses/by-sa/4.0/>

Published twice yearly by the Free Software Foundation, 51 Franklin Street, 5th Floor, Boston, MA 02110-1335, (617) 542-5942
info@fsf.org

This *Bulletin* was produced using all free software, including Inkscape, Scribus, and GIMP.

IMAGE CREDITS

Page 1: Photo by Greg Farough

Copyright © 2022.

Page 4: Image by Michael McMahon

Copyright © 2022.

Page 10: Image by Craig Topham

Copyright © 2022.

All images Copyright © Free Software Foundation, Inc., licensed under a Creative Commons Attribution ShareAlike 4.0 International license.

How to Contribute

Associate Membership:

Become an associate member of the FSF. Members will receive a bootable 16GB USB card, email forwarding, and an account on the FSF's Jabber/XMPP server. Plus: participate in our members forum at forum.members.fsf.org! To sign up or get more information, visit member.fsf.org or write to membership@fsf.org.

Online: Make a donation at donate.fsf.org, or contact donate@fsf.org for more information on supporting the FSF.

Jobs: List your job offers on our jobs page: fsf.org/jobs.

Free Software Directory:

Browse and download from thousands of different free software projects:
directory.fsf.org.

Volunteer: To learn more, visit fsf.org/volunteer.

LibrePlanet: Find local groups in your area or start your own at libreplanet.org! And join us online for the yearly LibrePlanet conference next spring.

Free Software Supporter: Receive our monthly email newsletter: fsf.org/fss.