LLNL-TR-822365

LAWRENCE LIVERMORE NATIONAL LABORATORY

Picking an Open Source License at LLNL: Guidance and Recommendations from the Computing Directorate

T. Gamblin

May 24, 2021

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Picking an Open Source License at LLNL

Guidance and Recommendations from the Computing Directorate

Todd Gamblin[*]
tgamblin@llnl.gov

May 24, 2021

1 Introduction

Science and innovation are at the core of LLNL's mission, and open source software is a vehicle by which LLNL innovations reach the scientific community, the nation, and the world. It also serves as a vehicle for building collaborations, publicizing our capabilities, and attracting new employees. Over half of the software produced by LLNL is open source, and open source licenses are the legal tools that allow our software to be used widely. Choosing an appropriate license can be difficult, but it can be even more difficult to *change* licenses after your software has amassed a significant number of contributors, so it is important to understand the potentially subtle implications of your license choice. This document provides a set of recommendations for the busy and uninitiated. They are not set in stone (there are always exceptions). Our goal is to provide default choices that align best with LLNL's mission, along with a high-level rationale.

1.1 Why release software as open source?

Our goals with open source are **broad adoption**, **community building**, **collaboration**, and **contribution**. These four attributes are indicators of external impact, and they help us make the case for continued funding. Most LLNL software is supported by taxpayer-funded programs, and we aim to make our software easily available to taxpayers (individuals and companies) as a public good. As the software becomes more popular and attracts more users, communities can form around projects and help make the work more sustainable. Not releasing software as open source can isolate us from the broader scientific community, increasing the likelihood of redundancy and decreasing our ability to foster meaningful collaborations over time. Not all software can be released as open source, e.g., some is export-contolled, and some sponsors will not give approval. But, our goal with most scientific projects is to ensure that they can be sustained into the future, either through community contributions, continued program funding, or industry adoption. The memo titled *Policy Guidance - OSS License Release of Software Developed with ASC and OASCR Funding* [8] outlines why DOE (ASC and ASCR) value and encourage releasing lab HPC software as open source.

1.2 Summary of recommendations

With these goals in mind, we recommend when releasing LLNL-developed software as open source that you **choose a permissive license, preferably one with an explicit patent license**. Permissive licenses allow anyone to use and distribute your code with few conditions. They are the *most compatible* licenses: they can be used in nearly any other software product. They lower barriers to collaboration, and companies increasingly require that employees obtain special approval to use software licensed under the less permissive, or *copyleft* licenses (see section 3.2 for a definition). These less-permissive licenses can be an obstacle for potential users, and can impede or prevent collaborations with industry. Building a broader user base around a permissive license frequently attracts more contributors than requiring contributions from a limited set of users willing to accept a non-permissive license.

Modern open source licenses include an explicit patent license in addition to a copyright license. This provides assurance to users that they are not infringing on any patents held by contributors, and it assures us that *we* are not infringing on patents that apply to others' contributions. Large companies (particularly in the tech industry) strongly prefer to have this assurance, as it further lowers the risks of adoption.

In the absence of other considerations, we recommend that you use:

```
Apache-2.0 WITH LLVM-exception
```

This is a permissive license used by the LLVM project. It has been scrutinized extensively by lawyers from major open source foundations, is maximally compatible (including with `GPL-2.0`), and it does not impose large compliance burdens on users of your code. It contains an explicit patent license. Read on for the rationale behind this choice.

[*]The author thanks Rob Neely, Ian Lee, Vanessa Sochat, and Bruce Hendrickson for the discussions that led to the writing of this position paper.

2 Background

Open source licensing is a complex topic and a full treatment of it is beyond the scope of this document. This is a short intro for those unfamiliar with licensing and Intellectual Property (IP) law.

2.1 Why do I need a license?

Open source licenses enable free distribution of software by removing some defaults laid out in contract and copyright law. Under U.S. copyright law, authors by default retain all rights to distribute and use their software. For users to copy and modify software, a *copyright license* must be granted. Typically, the original author retains copyright (i.e., ownership) of their work, but the license enables others to make use of it.

Similarly, under U.S. contract law, there are *implied warranties* with nearly any product. These come with liabilities – if a "reasonable consumer" might expect software to work in a particular scenario, the consumer could sue if it failed in some damaging way, *even if the software is free*. Nearly all open source licenses include a waiver of these types of liabilities so that the author does not incur risks by making it available.

As mentioned, our goal with open source is to distribute software widely. When you release software as open source at LLNL, the lab *must* release your software publicly. Be sure this is your goal before choosing an open source license.

2.2 Who owns my code?

Whoever owns the copyright controls how the code can be copied and distributed. For most open source projects, *copyright is retained by contributors*. At LLNL, Lawrence Livermore National Security (LLNS), LLC is the copyright holder. So, LLNS retains the copyright to code you write on the job, and contributors (or their organizations) retain the copyright for theirs. When you add a license to your code, contributions are assumed to have contributed *under that license*, and this is what allows the contributed code to be distributed freely. Projects with many contributions are effectively owned by *all* of the contributors.

2.3 Can I relicense my code?

Relicensing code with many contributors requires approval from all of the contributors to date. On the one hand, this is good—it means that a popular open source project cannot change its license to something proprietary at the whim of any one organization. On the other hand, it means that projects that *want* to relicense may need to go through a long relicensing process to accomplish that, and they may not succeed. Most open source licenses are *irrevocable*, so already released versions of the code will still be available under the original license. The new license will apply to new contributions (and new versions of the code).

2.4 What do open source licenses cover?

Open source licenses govern *intellectual property (IP)*. There are four common types of IP: copyright, patents, trademarks, and trade secrets. We ignore trade secrets, which are not open. The rest are:

- **Copyright** is IP that gives its owner the right to modify and reproduce a creative work. It controls who can copy and modify code. Open source licenses are based primarily on copyright law.
- **Patents** give their owners the right to exclude others from making, using or selling an invention for a period of time. Unlike copyright, which covers a specific creative work (the code), a patent covers the high level ideas. Software can infringe a patent without exactly copying any code.
- **Trademarks** are registered names, logos, or signs used to promote a product. Some open source licenses (`Apache-2.0`) explicitly forbid the use of project trademarks – this usually means that if you fork the project, you cannot call your version by the same name or use the logos of the original project.

Most modern open source licenses include a patent license as well as a copyright license, mainly because the murkiness of patent clauses in older licenses left the door open for lawsuits. It may be tempting to think that patent licenses are not needed if the *original* authors of the software have no patents on it. However, the patent license applies to *all contributors*, and it protects the original authors by forcing external contributors to provide a license for any patents that others might infringe.

	Permissive	**Copyleft**		
		Weak	*Strong*	*Network*
Patent License	`Apache-2.0`	`LGPL-3.0`, `MPL-2.0`	`GPL-3.0`	`AGPL-3.0`
No Patent License	`MIT`, `ISC`, `BSD-2-Clause`, `BSD-3-Clause`	`LGPL-2.1`, `LGPL-2.0`	`GPL-2.0`	`AGPL-2.0`

Figure 1: The major open source licenses, categorized by permissiveness and patent licenses.

3 Types of Licenses

There are many different open source licenses, but we strongly recommend using a widely used and accepted license. There are only a handful of these, and they are useful because they are well understood. When you use a well known license, users do not need to read and understand the details of a new license to understand what they are allowed to do with your software.

A useful mental model for open source licenses is the 2×4 matrix shown in Figure 1. The differences are explained in the following sections.

3.1 Permissive Licenses

Permissive licenses place few or no requirements on the user. They allow software to be copied, modified, combined with commercial software, and redistributed under more restrictive licenses. The main (and sometimes only) requirements are that the license and original copyright notice be included with the software.

Permissive licenses became steadily more popular during the 2010's, and they are now the most popular type of license. A survey conducted by WhiteSource, and open source security and license compliance management service, shows that in 2012, only 41% of open source projects used permissive licenses, but by 2020, permissive licenses were used by 76% of open source projects (Figure 2a). A 2015 study conducted by GitHub [1], showed similar results – the permissive `MIT` and `Apache-2.0` licenses were the most widely used across GitHub's entire site (Figure 2d). In 2020, `Apache-2.0` finally overtook `MIT` for the top spot (Figure 2b) after increasing in popularity steadily for many years (Figure 2c).

3.2 Copyleft licenses

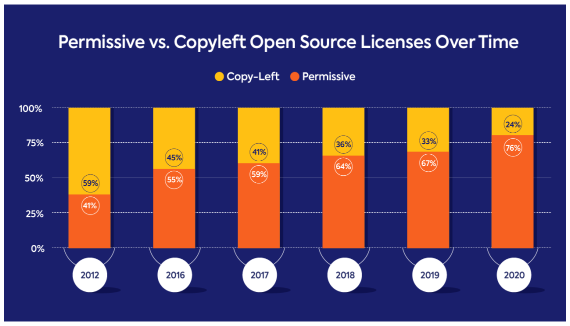
So-called copyleft licenses are more restrictive than permissive licenses. Copyleft licenses aim to ensure that software users can always modify and redistribute modified copies of the original software. They do this by requiring source code to be made available for any derived work they produce.

There are several types of copyleft licenses; the main distinctions between them are how they define *derived work* and how they determine when source code should be made available to consumers of the software.

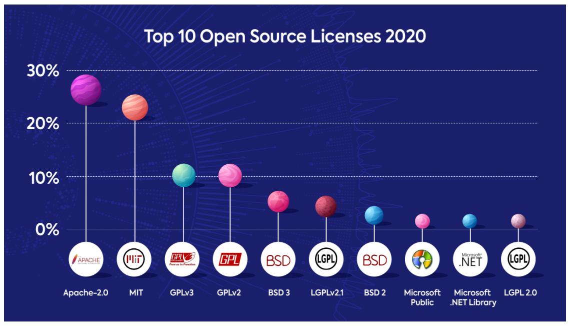
3.2.1 Strong Copyleft

The original strong copyleft license was the GNU General Public License (GPL), and the GPL defines derived work as *any* software that extends GPL-licensed work or combines with it to form a "larger program". This means that if you use a GPL library or include GPL code when you distribute your program, your entire code *must* be distributed under the GPL. That is, you must make source available for it. For this reason, people say that the GPL is a "viral" license, as any GPL product affects all parts of a large program.

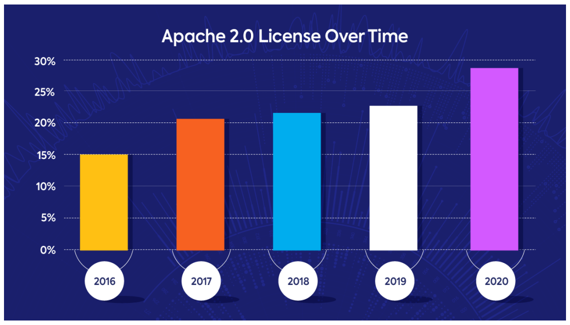
For many proprietary packages, the GPL is a non-starter, so GPL is frequently used to *prevent* users from incorporating software into proprietary products. This typically reduces businesses' willingness to rely on GPL software – or at least on GPL libraries that link with their main product. Because of this, other researchers may be uninterested in using or developing your software, and the community may gravitate towards a more open alernative (even if it is currently technically inferior).



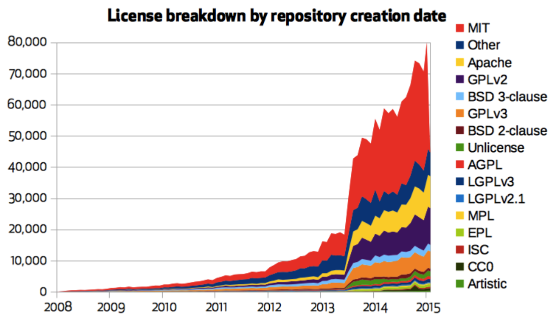
(a) Rise of permissive licenses in the 2010's [5]



(b) Top 10 open source licenses in 2020 [5]



(c) Rise of `Apache-2.0` [5]



(d) Licenses over time on GitHub, 2015 [1]

Figure 2: Prevalence of major open source licenses over time on GitHub

3.2.2 Weak Copyleft

Weak copyleft is similar to strong copyleft, but it can be incorporated into proprietary code. In weak copyleft licenses, derived work is defined as modifications to *the work itself*, not other works that are combined with it. Weak copyleft is designed for libraries – it requires that changes to the library be made available, but dependent applications and libraries need not release their code. The Lesser GPL (LGPL) is the best known weak copyleft license. It was created for `glibc`, the GNU C library, so that proprietary software could be distributed with Linux distributions without violating the GPL.

Weak copyleft licenses like Lesser GPL (LGPL) can still be an obstacle to industry use, because compliance is extremely difficult when static linking is required. An application that depends on a *dynamically* linked LGPL library need not be distributed under the LGPL license, but if the same application *statically* links to an LGPL library, it must also be licensed as LGPL. This is because of the strict requirement for end user freedom—the LGPL says that the end user must always be able to produce a new version of the complete product that uses a modified version of the LGPL portion. With static linking, it is nontrivial to swap out *just* the LGPL code, so typically the entire product must be LGPL licensed. These complexities lead industry customers, particularly in HPC and the embedded world where static linking is very common, to disallow LGPL just as they disallow GPL.

3.2.3 Network Copyleft

Network copyleft licenses like the Affero GPL (AGPL) are equivalent to the GPL save for one change – how the source distribution requirement is triggered. Rather than require that source be distributed to any recipients of the software *binaries*, AGPL requires that source be distributed to anyone *who uses the software as a service over a network*. AGPL was intended to get around a loophole in the original GPL – namely that software was less frequently distributed in binary form, and was more frequently used over a network. This type of use case was not anticipated when the GPL was written, and AGPL is intended as a GPL update for the software-as-a-service (SaaS) era.

Note that AGPL does *not* redefine derived work to cover network use of the software. In this sense it is more like LGPL than GPL: *only* the source for the AGPL network service must be made available, *not* other programs that connect to it remotely. AGPL was intended to create a GPL-like copyleft ecosystem for the cloud, but it has so far not fulfilled that purpose. More often, it is used by small companies to force customers to buy a commercial license for their product.

3.3 Patent licenses and compatibility

As mentioned above, modern licenses contain a patent license in addition to a copyright license. Having both of these makes it clear that your software is unencumbered—i.e., that users will not infringe any contributor's IP. This reduces the risk of adoption for your users. Patent licenses have become a de-facto standard within the tech industry.

`Apache-2.0` was designed to allow large companies with large patent portfolios to work together on common infrastructure. `Apache-2.0`'s patent license deters contributors from suing each other. It contains a *retaliation clause* that says that you *lose* the patent license if you sue for patent infringement. Unfortunately, this clause was deemed *incompatible* with `GPL-2.0` (and therefore `LGPL-2.x` and `AGPL-2.0`) because it can take away some user freedom. This means that `*GPL-2` projects cannot have `Apache-2.0` dependencies, which means that `Apache-2.0` cannot be used as widely as we might like. Many ecosystems still use the `GPL-2.0` license (e.g., the Linux kernel), and being compatible with them is important for broad adoption. While the Free Software Foundation found `Apache-2.0` to be incompatible with the letter of `GPL-2.0`, they nonetheless thought that the patent clause was a good idea, and `GPL-3.0`, `LGPL-3.0`, and `AGPL-3.0` all contain similar patent clauses. `Apache-2.0` software can be used in projects with `*GPL-3.0` licenses.

3.4 Permissive licenses with patent licenses

`Apache-2.0`'s patent license has made it popular, but its unintended incompatibility with `GPL-2.0` can be a hindrance for software that needs to be used in many different ecosystems. There have been many attempts to come up with a widely compatible permissive license that also includes a patent clause.

The Rust project [7] was one of the first projects to solve this issue, in their case by *dual licensing* `Apache-2.0 OR MIT` [4]. The dual license works because contributors must contribute code under the terms of *both* licenses, which means that they provide a patent license under `Apache-2.0`. Users, however, can use either `Apache-2.0` or `MIT`, and `MIT` is `GPL-2.0`-compatible, so `GPL-2.0` projects can use the code under `MIT`. LLNL's Spack project adopted this approach based on Rust – with a very similar goal of maximum possible permissiveness [3]. LLNL's Hypre project followed suit [2]. The Python Cryptography project was able to achieve the same goal by dual licensing with `Apache-2.0 OR BSD-2-Clause`.

Today, there is a single license that achieves both of these goals: `Apache-2.0 WITH LLVM-exception`. As with others, the LLVM project wanted a `GPL-2.0`-compatible, patent-granting, permissive license. Lawyers from the LLVM project and the Free Software Foundation worked together to add exceptions to `Apache-2.0` that would allow it to be compatible with `GPL-2.0`. The new `Apache-2.0 WITH LLVM-exception` license adds minimal additional language to the widely accepted `Apache-2.0` license, and it has had extensive scrutiny from the stewards of both `Apache-2.0` and `GPL-2.0`. More on the exception can be found at the LLVM relicensing website [6].

4 Recommendations

4.1 The Argument for Permissive

While the requirement of less permissive licenses like `GPL-3.0` and `LGPL-3.0` discussed above may sound attractive to you as an open source author who wants to force modifications to come back to you in the form of contributions, there are three primary counterarguments to consider:

1. In the end, copyleft is largely unenforcable. Bad actors are likely not swayed by the existence of a license. If caught, they can likely just cease to use your software to come back into compliance.
2. Copyleft licenses do not obligate your users to contribute back to your project – only to distribute source code with any derived work. Contributions can be hard work, and users are motivated to contribute when they know that their changes will be *maintained* with the upstream project, i.e., when they can leverage the efforts of project maintainers. This incentive is the same regardless of license.
3. As discussed, copyleft likely limits the audience of users and thus the pool of potential contributors, and can incentivize the development of more open/permissive competing products.

Yes, someone may take your permissively licenced software and profit from it by including it in a commercial closed-source product. They may even do so without attribution. However, many users *will* acknowledge your software in their documentation (especially if asked) or publications *because* you have granted them license to do so. Permissive licenses grow the pie. This is likely to increase the impact of your project, which will ultimately increase your recognition.

4.2 So, which license should I use?

As mentioned in the Introduction, our recommendation is to choose a permissive license with a patent license. The best license we know of for this is **`Apache-2.0 WITH LLVM-exception`**. This license is compatible with all the major open source licenses, is permissive, contains a patent license, and has been closely scrutinized by lawyers from the LLVM project and the Free Software Foundation.

There are reasons not to abide by this recommendation, For example, you might be working in a community that strongly prefers a particular license. Or, if a critical dependency of your software uses a strong copyleft license like GPL, you may have to distribute your software under that license, as well.

If you must use another license, our recommendations are as follows:

1. **`Apache-2.0 WITH LLVM-exception`**: permissive, has patent license, broadly compatible, based on well established `Apache-2.0 license`, extensive lawyer scrutiny.
2. **`MIT`**: Permissive, simple. Broadly compatible. Has no explicit patent license. `ISC` is shorter, functionally equivalent, and less popular. `BSD-2-Clause` and `BSD-3-Clause` are pretty much equivalent, though there is only one MIT license (so less confusion).
3. **`LGPL-3.0`**: If you must use weak copyleft, use a copyleft license with a patent clause that is compatible with `Apache-2.0`, so that you can leverage the cloud software ecosystem (e.g., Kubernetes, Docker, most packages from Google, Amazon, etc.)
4. **`GPL-3.0`**: If you must use strong copyleft, use a copyleft license with a patent clause that is compatible with `Apache-2.0` (see above).

5 SPDX Identifiers

The license names used in this document (`Apache-2.0`, `Apache-2.0 WITH LLVM-exception`, `MIT`, `GPL-3.0`, `LGPL-3.0`, etc.) are Software Package Data Exchange (SPDX) identifiers. SPDX is an open standard for communicating software bill of material information, including licenses. A full list of license identifiers and links to full license text can be found at the SPDX website. [9]

References

[1] Ben Balter. Open source license usage on GitHub.com. `https://github.blog/2015-03-09-open-source-license-usage-on-github-com/`, March 9 2015.

[2] Rob Falgout. Hypre Relicense. `https://github.com/hypre-space/hypre/pull/31`, July 10 2019.

[3] Todd Gamblin. Relicense Spack from LGPL-2.1 to MIT/Apache-2.0. Spack GitHub Project. `https://github.com/spack/spack/issues/9144`, August 30 2018.

[4] Graydon Hoare. [rust-dev] Please read: Rust license changing (very slightly). Rust developer mailing list. `https://mail.mozilla.org/pipermail/rust-dev/2012-November/002664.html`, November 21 2012.

[5] Patricia Johnson. Open Source Licenses in 2021: Trends and Predictions. `https://www.whitesourcesoftware.com/resources/blog/open-source-licenses-trends-and-predictions`, January 28 2021.

[6] LLVM Foundation. LLVM Relicensing Effort. LLVM Foundation Website. `https://foundation.llvm.org/docs/relicensing/`, Last updated Feb 8 2021.

[7] Nicholas D Matsakis and Felix S Klock. The Rust Language. *ACM SIGAda Ada Letters*, 34(3):103–104, 2014.

[8] Department of Energy Memo. Policy Guidance - OSS License Release of Software Developed with ASC and OASCR Funding. `https://science.osti.gov/-/media/ascr/pdf/research/docs/Doe_lab_developed_software_policy.pdf`, 2003.

[9] SPDX Workgroup. SPDX License List. SPDX website `https://spdx.org/licenses/`, March 7 2021.