Closed Source: The God that Failed

Reverse engineering, espionage and the ethics of citation in the age of coding agents

Mr. Nobody

October 2026 · v0, ASD-STE100 text (preprint, not peer reviewed) · rev. 6b3aae5+219





Abstract. For thirty years, closed source promised that secrecy protects value. I argue that this promise failed, and that it failed for science first. I examine the case with the book that this book stands on: Robert M. Pirsig's *Zen and the Art of Motorcycle Maintenance* (1974). Pirsig wrote that the number of rational hypotheses that can explain a phenomenon is infinite. Thus science does not end, and only the owner of a closed method can test it. Three trends from 2024 to 2026 support this argument. Large language models now decompile binaries into re-executable source code. Black-box queries extract parts of production models. Distillation copies capabilities at industrial scale.

Apple's lawsuit against OpenAI is the largest 2026 trade-secret dispute in artificial intelligence. It is about people, laptops and recruitment, not about code that stayed secret. With this background, I examine Linus Torvalds' Linux and Git again as evidence that openness works at a large scale. I also examine my closed undergraduate thesis as evidence of the cost of secrecy to a researcher.

I review obfuscation, which includes an example of my own, and I show why obfuscation cannot hide code. I compare it with the idea of John Nash in 1955: security that comes from the cost of computation. Signed commits, keys in hardware and two-factor authentication protect what secrecy cannot protect.

Then I argue that professional citation is the ethical answer to universal reverse engineering. It has these parts: persistent identifiers, machine-readable citation metadata, and agents that borrow, cite and propagate credit by default. I describe Paper Fabric, an open pipeline in the Ouroboric style that improves itself and that produced this preprint. I also list its own known limitations.

Keywords: open source; reverse engineering; AI-assisted reverse engineering; model distillation; trade secrets; software citation; research software; FAIR; coding agents.

1 Introduction

In 1949, six writers who left the Communist movement published their reasons in a book with the title *The God That Failed*. The title of this paper borrows the phrase for a smaller and less tragic faith. This faith is the belief that secrecy is the thing that keeps source code, models and methods valuable. I call that belief *closed source* in a broad sense. It includes closed-source software, unpublished research code, model weights behind an API, and the scripts of a thesis that only its author saw.

The faith has two articles. The first article says that secrecy protects the owner: if nobody can read a thing, nobody can copy it. The second article says that secrecy costs nothing to everyone else. When one laboratory, one company or one student keeps their work secret, the world loses little. I argue that both articles failed. I also argue that this failure became clear between 2024 and 2026.

The first article failed because reverse engineering is no longer rare. Reverse engineering was always lawful and common (Samuelson and Scotchmer 2002). The change is in its cost.

Language models now change compiled binaries into source code that compiles and passes tests (Tan et al. 2024). A few thousand API queries recover parts of a production language model (Carlini et al. 2024). Tens of thousands of accounts can distil the behaviour of a frontier model into the model of a competitor (Anthropic 2026; Reuters 2026). In July 2026, one of the most valuable companies in the world sued one of the most valuable laboratories about trade secrets. Its complaint was about people, laptops and recruitment, not about source code that stayed safely secret (Perez 2026a; Bridge Counsels 2026).

The second article failed because the costs of secrecy fall on science. If nobody can examine a result, nobody can reproduce it (Ince, Hatton, and Graham-Cumming 2012). If nobody can find code, nobody can credit it (Smith et al. 2016). A whole economy now operates on code that someone chose to make open. One estimate says that the open source that firms use would cost about 8.8 trillion US dollars to build again (Manuel Hoffmann, Nagle, and Zhou 2024).

Robert M. Pirsig wrote the book that this book stands on. In *Zen and the Art of Motorcycle Maintenance*, he gives the problem of science in one sentence: "The number of rational hypotheses that can explain any given phenomenon is infinite" (Pirsig 1974, 107).

[ANALYSIS] If the number of hypotheses has no end, science has no end. No single laboratory can test all the hypotheses. Only its owner can open a closed method, thus no other person can test it. For this reason, I think that openness is necessary for science first, and for business after.

If anyone can reverse engineer anything, the useful question is no longer *how do I hide my work*? The useful question is now *how do I make sure that people credit my work when they use it*? My answer is professional citation. It has these parts: persistent identifiers, machine-readable metadata and, more and more, agents that cite by default. This answer has a practical side. Thus I also describe the open pipeline that produced this paper.

The structure of this paper is as follows. Section 2 examines Linux and Git again as evidence that openness works at a large scale. Section 3 is a confession about my closed undergraduate thesis. Section 4 reviews science and espionage in 2026, and the explosion of AI-assisted reverse engineering.

Section 5 gives five lines of evidence that closed source is dead as a protection. Section 6 examines obfuscation, John Nash's letters on cryptography, and how to secure source code in 2026. Section 7 argues for professional citation and gives rules for agents. Section 8 describes Paper Fabric. Section 9 lists my related work. Section 10 gives the limits of my argument.

This paragraph is about my method. This is a position paper. It uses the *research* style of the Ouroboric documentation standard. In this style, each factual claim has a citation, and the mark **[ANALYSIS]** shows each passage of interpretation. It is a v0 preprint, and it is not peer reviewed.

2 The bazaar that worked: Linus Torvalds, Linux and Git

On 25 August 1991, a student at the University of Helsinki sent a message to the Usenet group `comp.os.minix`. He wrote that he was "doing a (free) operating system (just a hobby, won't be big and professional like gnu)" for 386 and 486 AT clones. He asked what people wanted to see in it (Torvalds 1991). Version 0.01 followed on 17 September 1991. In early 1992, Torvalds changed the licence of the kernel to the GNU General Public License. Version 0.95, in March 1992, was the first release under this licence (Wikipedia contributors 2026b).

The hobby became the most widely deployed operating-system kernel in history. It runs on servers, phones, routers, cars and the supercomputers on which much of computational science depends. I do not need a market-share figure for this point. Almost certainly, the reader's own week went through several Linux machines.

The first achievement of Torvalds was not only technical. *The Cathedral and the Bazaar*, the account by Eric Raymond of Linux development, identified the social method behind it. The method is to release early, release often, treat users as co-developers, and let many eyes find bugs that few eyes would find (Raymond 1998). Raymond compared this *bazaar* with the *cathedral*, in which development is carefully guarded and centrally planned. Linux showed that the bazaar could build the hardest kind of software that exists.

The second achievement of Torvalds came from a failure of closed source. From 2002, the kernel used BitKeeper, a closed-source version-control system that was available at no cost. In April 2005, that free access ended. No free system met the needs of the project, so Torvalds wrote his own. He started on 3 April 2005, and on 7 April Git managed its own source code. The first multi-branch merge followed on 18 April, and on 26 July he gave maintenance to Junio Hamano (Wikipedia contributors 2026a).

The first commit in the Git repository is still readable today. Its title is "Initial revision of"git", the information manager from hell" (Torvalds 2005).

[ANALYSIS] The BitKeeper episode is a small parable for this paper. The kernel community accepted a closed tool for convenience, and was exposed when the owner withdrew it. When Torvalds replaced it with an open tool, the replacement became the base of almost all software development. This base includes the repositories, pull requests and release tags through which this paper is built and archived. Git's data model has the ethic that Section 7 argues for: each commit names its author and points by cryptographic hash to its exact history. Git is a citation graph for code.

I praise Torvalds here without reservation for these two gifts, and for three decades of maintenance of the first gift. I also note what this praise does not require. It does not require that others copy the technical style of one person, or that open source is free of conflict. The claim is narrower and stronger. At the largest scale that anyone ever tried, openness worked.

3 A closed-source confession: Chemo-RX

Matheus Hoffmann writes this section in the first person. The passages with the mark **[AUTHOR ANECDOTE]** are placeholders for the author's own memories. The author must write them before this paper leaves v0. An agent must not invent any part of them.

In 2018, I started an undergraduate thesis with the alias Chemo-RX at the Escola Politécnica of the Universidade Federal do Rio de Janeiro. The thesis was in association with the LESS laboratory of the Biomedical Engineering Programme. The goal was to compare statistical methods that predict adverse reactions to chemotherapy in women who received treatment for breast cancer. The thesis used data from Brazil's National Cancer Institute (INCA). The reactions in scope were asthenia, myalgia, arthralgia, mucositis, abdominal pain and diarrhoea. The predictors included comorbidities, demographic variables and single-nucleotide polymorphisms (Matheus Hoffmann 2018).

The public trace of that work is a GitHub repository with eight commits between 23 March and 25 April 2018 (Matheus Hoffmann 2018). It contains a README, a proposal in OpenDocument format and a data summary in R Markdown. The README has three headings that nobody ever completed: "Proposal EN link", "Final project EN link" and "Thesis full

content". Under the last heading, the only text is "contact me". The to-do list of the proposal still contains "Add INCA formal license/ethic regiment".

[AUTHOR ANECDOTE: how the project started, who supervised it, and what the clinical question meant to the people involved.]

[AUTHOR ANECDOTE: why the thesis text and code stayed closed. Was it a supervisor's instruction, uncertainty about the data agreement, unfinished work, or simply never getting round to it?]

[AUTHOR ANECDOTE: what was lost. For example, results that could not be reused, collaborators who could not build on the work, or the experience of being unable to show the work later.]

[ANALYSIS] That repository mixes two kinds of closedness, and the distinction is important for the rest of this paper. The patient data could not be published, and it was correct not to publish them. Clinical records are personal data, and ethics approval and data agreements control their release, as the to-do list itself shows. But it was possible to publish the *method*: the statistical code, the comparison protocol, the variable definitions and the negative results.

[ANALYSIS] The FAIR principles make this exact distinction. Data and software should be findable and accessible *under well-defined conditions*, and these conditions can include restricted access. Their metadata should stay public even when the data are not public (Wilkinson et al. 2016; Barker et al. 2022). To close the method together with the data protected no patient. It only made the work impossible to cite.

[AUTHOR ANECDOTE: what the author has done differently since, leading into the open-source projects in Section 9.]

4 Science and espionage in 2026: the reverse-engineering explosion

I use the term *reverse engineering* for any process that recovers how an artefact works from the artefact itself, without access to its design documents. My project uses the shorthand REA for reverse engineering, assisted. Examples are to decompile a binary, to infer a model from its outputs, or to build a product again from its parts. This section reviews four developments. The first development is a lawsuit, and its facts are allegations. I report them as allegations.

4.1 Apple v. OpenAI: secrets leave on two legs

On 10 July 2026, Apple sued OpenAI, the device company io that OpenAI acquired before, and two former Apple employees. The court is the United States District Court for the Northern District of California (San Jose). The claims are trade-secret misappropriation under the Defend Trade Secrets Act of 2016 and, against the two individuals, breach of contract (Bridge Counsels 2026; Perez 2026a).

According to the complaint as reported, a former Apple engineer accessed Apple systems through an authentication bug, from the Apple-issued computer of a colleague. The complaint says that OpenAI's chief hardware officer, a former Apple executive, asked job candidates to bring Apple parts, design files and prototypes to interviews. The complaint also says that new employees received guidance about how to avoid Apple's exit process. Apple also alleges that more than 400 former Apple employees now work at OpenAI (Perez 2026a).

OpenAI said publicly that it has "no interest in other companies' trade secrets" (Perez 2026a). In August, OpenAI moved to dismiss. It argued that Apple let employees use personal cloud accounts for work, did not revoke access correctly, and did not identify which secrets were taken (Perez 2026b). When I wrote this paper, the motion to dismiss and a motion for a preliminary injunction were pending, and no court ruled on the merits (Bridge Counsels 2026; Perez 2026b).

[**ANALYSIS**] Whatever the result, the dispute is useful because it is *not* about source code that an adversary could not read because it was secret. Each alleged path goes through people: their accounts, their devices, their recruitment conversations and their memories. In this account, secrecy was a property of the processes of an organisation, and processes leak. Also, as reported, part of OpenAI's defence is that Apple's own practices did not treat the information as secret. A secret that must have an active guard at each exit interview is expensive. The cost of the guard is part of the price of closed source.

Verification note. I could not fetch the docket directly. Secondary sources give the case number as 5:26-cv-07078, with two judge suffixes that do not agree (VKD and EJD). They give the caption both as *Apple Inc. v. OpenAI* and as *Apple Inc. v. Liu* (Bridge Counsels 2026). The number must be checked against PACER before this paper leaves v0.

4.2 Distillation: capability copied through the front door

Knowledge distillation trains a smaller model to imitate the outputs of a larger model. It is a standard and legitimate technique (Hinton, Vinyals, and Dean 2015). In 2026, it also became the centre of an accusation of industrial copying.

According to Reuters, OpenAI made a statement in February to a committee of the US House. The committee is the US House Select Committee on Strategic Competition between the United States and the Chinese Communist Party. OpenAI said that accounts associated with DeepSeek employees developed methods to bypass OpenAI's access restrictions and "obtain outputs for distillation in programmatic ways" (Reuters 2026). DeepSeek did not reply to Reuters. Ten days later, Anthropic reported "industrial-scale campaigns by three AI laboratories": DeepSeek, Moonshot AI and MiniMax. According to Anthropic, these campaigns used about 24,000 fraudulent accounts and more than 16 million exchanges with Anthropic's Claude models (Anthropic 2026).

[**ANALYSIS**] These are accusations by interested parties, and I do not judge them. Their relevance here is structural. A model served behind an API is closed source in the strongest sense: its weights never leave the servers of the provider. But each answer from the model is a sample of its behaviour. Enough samples teach a student model much of what the teacher model knows. The interface that sells the model is the interface that leaks it.

4.3 Extraction: a model read through its answers

Academic work made the same point with more precision and less drama. In 2016, Tramèr and colleagues showed that it is possible to steal machine-learning models served through prediction APIs, with queries to these models (Tramèr et al. 2016).

In 2024, Carlini and colleagues went further, with an attack on production language models. Their attack recovered the entire embedding projection matrix of OpenAI's Ada and Babbage models, for under twenty US dollars in queries. The attack also confirmed the hidden dimensions of these models for the first time (Carlini et al. 2024). The paper, published at ICML 2024, ends with defences and mitigations. When a research community discovers a leakage, its response is to publish it, so that it can be fixed.

4.4 Decompilation: the end of the opaque binary

Reverse engineering of compiled software is a routine practice with a long history, and it is lawful for purposes such as interoperability (Samuelson and Scotchmer 2002). In 2019, the US National Security Agency released its own reverse-engineering framework, Ghidra, as open source under the Apache-2.0 licence (National Security Agency 2019). Then language models lowered the last barrier, which was skill. LLM4Decompile is a family of open models trained for decompilation. It produces source code that recompiles and passes the original tests. Its end-to-end models more than double the re-executability rate of GPT-4o and Ghidra on the HumanEval and ExeBench benchmarks (Tan et al. 2024).

[ANALYSIS] If I put these four developments together, I see that the cost curve of reverse engineering collapsed on all axes. The axes are binaries (decompilation), models (extraction and distillation) and organisations (people who move between employers). I call this the *REA explosion*. It does not make each artefact transparent overnight, and it does not make copies lawful where the law forbids them. But it does make secrecy a weak and expensive defence. It also makes the question of credit (who did what first, and who used whose work) more important than the question of access.

5 Why closed source is dead: five lines of evidence

"Dead" is a strong word, and I should say what I mean. Closed-source software will be sold for a long time. What is dead is closed source *as a protection*: the idea that if an artefact stays unreadable, its value stays safe. It is also the idea that this protection is worth its cost to science. Five references from cryptography, law, machine learning, software engineering and economics support this claim. I chose them for credibility: each is peer reviewed, a primary legal or historical source, or a widely cited working paper from a research university.

Table 1: Five references on why secrecy fails as a protection.

#	Reference	Field	What it shows
1	Kerckhoffs (1883)	cryptography	A system must stay secure when the enemy has it; only the key may be secret.
2	Samuelson and Scotchmer (2002)	law and economics	Reverse engineering is an old, lawful and economically useful practice.
3	Carlini et al. (2024)	machine learning	Parts of a production model behind an API can be extracted for under 20 US dollars.
4	Tan et al. (2024)	software engineering	Language models decompile binaries into source that recompiles and passes tests.
5	Ince, Hatton, and Graham-Cumming (2012)	science policy	Results computed by unpublished code cannot be properly checked or reproduced.

1. Security by obscurity was refuted in 1883. Auguste Kerckhoffs' second principle for military ciphers says that the system "must not require secrecy and can be stolen by the enemy without causing trouble" (Kerckhoffs 1883). Modern cryptography has this principle as its base: the algorithms are public, and only the keys are secret. A design is considered broken in advance if its security needs the design to stay hidden.

2. The law accepted reverse engineering long ago, and accepts it now. Samuelson and Scotchmer start their survey with the observation that reverse engineering "has a long

history as an accepted practice" (Samuelson and Scotchmer 2002). Trade-secret law gives protection against improper means, such as theft and breach of confidence. It does not give protection against a person who studies a product that they lawfully obtained. Thus closed source never had a legal guarantee that nobody would understand it. It had a practical guarantee: reverse engineering was slow and expensive.

3. That practical guarantee is gone for models. Extraction attacks on prediction APIs (Tramèr et al. 2016) developed into attacks on production language models (Carlini et al. 2024). Now the providers themselves allege distillation at scale (Anthropic 2026; Reuters 2026).

4. For binaries, it starts to disappear. Open decompilation models recover executable source from compiled code (Tan et al. 2024). These models use an open reverse-engineering framework that a signals-intelligence agency released (National Security Agency 2019).

5. At the same time, secrecy costs science. Ince and colleagues argued in *Nature* that, with few exceptions, "anything less than the release of source programs is intolerable for results that depend on computation" (Ince, Hatton, and Graham-Cumming 2012). Now it is possible to measure the economic weight of the alternative. The open source that firms use would cost an estimated 4.15 billion US dollars to write once. If each firm had to replace it alone, the cost would be about 8.8 trillion US dollars (Manuel Hoffmann, Nagle, and Zhou 2024).

[**ANALYSIS**] Lines 1 to 4 say that secrecy no longer protects. Line 5 says that secrecy was never free. Together they leave a researcher with a choice that is no longer between *hidden* and *exposed*. It is between *exposed and uncredited* and *exposed and credited*. The next section is about the choice of the second option.

6 Obfuscation, Nash's letters and how to secure source code in 2026

If secrecy fails as a protection, what remains for code that must stay private? And what remains for the public code that other people depend on? This section examines one attractive answer, obfuscation. It also examines the different answer that cryptography gives.

6.1 Nash's letters: security from computation, not concealment

In early 1955, John Nash wrote to the US National Security Agency to propose an enciphering machine (Nash and National Security Agency 1955). Nash is the author of the equilibrium concept for non-cooperative games (Nash 1950). The NSA declassified the correspondence in 2012 (Branwen 2012). In it, Nash argued that, for sufficiently complex ciphers, "the mean key computation length increases exponentially with the length of the key" (Branwen 2012). The idea is that security should come from the cost of computation, not from a hidden design. Rivest, Shamir and their students read the letters as an anticipation of complexity theory and modern cryptography (Rivest, Shamir, et al. 2012).

The machine of Nash himself had less success. The NSA decided that it "affords only limited security" (Nash and National Security Agency 1955). In 2012, Ronald Rivest, Adi Shamir and their students analysed the machine. They found a polynomial-time chosen-plaintext attack and a subexponential known-plaintext attack. They concluded that Nash was "a much better game theorist than a cryptographer" (Rivest, Shamir, et al. 2012).

[**ANALYSIS**] Both parts of the story support Kerckhoffs. Modern cryptography kept the principle of Nash: security that comes from the work that an attacker must do. After Nash's design was published, other people judged it and broke it. That is how cryptography should work: the method is open, and only the key is secret.

6.2 Obfuscation: what it gives and what it cannot give

Obfuscation changes a program so that it does the same thing but is more difficult to read. One of I made a small, honest example. *nashkell* is a Haskell pipeline that applies OCR to the declassified letters of Nash. It asks a language model to annotate and cross-check the letters. It stores its source code in Git with identifiers translated into Latin (Matheus Hoffmann 2026a).

On commit, a Git `clean` filter translates English identifiers to Latin. On checkout, a `smudge` filter translates them to English again. Both filters use a fixed dictionary. If a person clones the repository without the filter, that person sees the Latin form. GitHub also shows the Latin form.

The documentation of the project lists the limits, and these limits are general (Matheus Hoffmann 2026a):

- **Structure survives.** Only top-level names change. Control flow, types and local variables stay intact, and a reader can infer them. Thus the result "raises, not eliminates, reading difficulty" and is "not a security boundary".
- **The key is the dictionary.** The mapping is a bijection, and it is stored next to the tool. A person who has the mapping, or can guess enough of it, can reverse the transformation.
- **The stored form does not build.** The filter does not rename file paths, so the Latin modules no longer agree with their paths. Obfuscation also costs the owner.
- **It is a heuristic.** The filter operates line by line and is not a parser. It does not process block comments or multi-line strings. A person must add new names to the dictionary manually.

Theory shows that more effort cannot remove these limits. Barak and colleagues proved that a general "virtual black box" obfuscator cannot exist. Such an obfuscator would let no one compute from the obfuscated program anything that they could not compute from input-output access alone. The authors made a family of programs that a person can reverse engineer from any equivalent program. But black-box access to these programs shows nothing of their code (Barak et al. 2012). The result leaves space for weaker types of obfuscation, but not for a general method that makes source code secret.

[**ANALYSIS**] The language models of Section 4 already recover source code from binaries. Against these models, the translation of identifiers into Latin is a speed bump.

6.3 What to secure instead: integrity, identity and keys

[**ANALYSIS**] The lesson of Nash and of obfuscation is this: do not protect the *readability* of code. Protect what cryptography can protect: that code is what its authors wrote, and that only they could write it. In 2026, this means four layers, and each layer has public, mature tools.

1. **Sign commits and tags.** Git signs tags and commits with GnuPG. The command `git merge --verify-signatures` refuses history that does not have trusted signatures

(Chacon and Straub 2014). Since Git 2.34, SSH keys can sign as well as GnuPG keys (The Git project 2021).

2. **Keep signing keys in hardware.** Since OpenSSH 8.2, the `ecdsa-sk` and `ed25519-sk` key types use FIDO security keys. The private key never leaves the token, and the token must be present when the key is used (OpenSSH project 2020). Thus a person who steals a laptop no longer gets the signing key.
3. **Require two-factor authentication on the forge.** In 2022, GitHub made a commitment about two-factor authentication. By the end of 2023, each user who contributes code on GitHub.com must enable it (Hanley 2022). A person who takes control of an account can push code in the name of its owner.
4. **Make signatures verifiable by everyone.** Sigstore lets developers sign with ephemeral keys that are bound to an existing identity through OpenID Connect. Sigstore records signatures in transparency logs. This removes most of the burden and risk of key management (Newman, Meyers, and Torres-Arias 2022).

Hash functions are also important. In 2017, Stevens and colleagues produced the first collision for full SHA-1 (Stevens et al. 2017). Git used SHA-1 as the hash for its object names. Git version 2.29 added a SHA-256 object format, and Git marked this format as experimental at first (The Git project 2020).

In this model, only a small and explicit set of things stays secret. These things are private keys, credentials, and data that is personal or under agreement, as in Section 3. These things should have encryption and access control. For the source code itself, it is better to publish it, sign it and cite it.

[**ANALYSIS**] In the vocabulary of Nash himself, this is a change of strategy in a game whose payoffs moved. When reverse engineering was expensive, secrecy could be a sensible strategy for an author. When reverse engineering is cheap, the payoff of secrecy falls towards zero. But the payoff of a publication that others can check and credit does not fall. I do not claim that I computed an equilibrium (Nash 1950). I claim only that the costs on which the old strategy rested are gone.

7 The ethical answer: professional citation

If anyone can read your work, one protection remains: the norm that each person who uses the work says so. That norm is old in science. It is new for software, and for agents it is almost not formed.

7.1 Citation for people

The FORCE11 *Software citation principles* define a correct citation of software. Software is a valid product of research, and people should cite it as a product of research. A citation should give credit, be unique and persistent, and be accessible. It should also be specific to the version that the person used (Smith et al. 2016). The FAIR principles ask that research outputs are findable, accessible, interoperable and reusable (Wilkinson et al. 2016). FAIR4RS adapts these principles to research software (Barker et al. 2022).

In practice, each repository needs three things for this. The first is a persistent identifier: a DOI that an archive such as Zenodo issues. The archive gives a *concept* DOI that always resolves to the latest version, and a *version* DOI for each release. The second is machine-readable metadata, so that a tool can generate a citation and a person does not type it. The metadata files are `CITATION.cff` for GitHub's "Cite this repository", and `.zenodo.json` for

the archive. The third is a release process that gives each published version its own identifier, so that a reader can cite exactly what they used.

This repository uses this method, and Section 8 gives the details. Each item of the related work in Section 9 also uses this method and has a concept DOI.

7.2 Citation for agents: borrow, cite, propagate

Coding and research agents now write a large part of new code and text, which includes the drafts of this paper (see the AI-use statement). Agents read much more than any person, and they copy easily. If agents do not cite, the REA explosion becomes a machine that launders work. Code and ideas enter with their authors and exit without them. Thus, I propose three rules for agents. Paper Fabric's `AGENTS.md` makes these rules mandatory for each agent that works in its repositories.

1. **Borrow only what the licence allows.** Before you reuse code, text, figures or data, read the licence. Then record the licence. If the licence does not permit the reuse, it is not borrowing. If you are not sure, describe the idea and cite it. Do not copy its expression.
2. **Cite at the point of use, with a persistent identifier.** Give each borrowed element a citation at the point where you use it. A citation only in a final list is not sufficient. If a DOI exists, use the DOI. If no DOI exists, use a URL with a commit hash or an archive snapshot. Never invent a reference. If you cannot fetch and check a source, mark the claim as unverified.
3. **Propagate credit downstream.** If your output is published, keep its citations with it. Put the citations in the bibliography and in the `CITATION.cff` references. Also put them in the related identifiers of the Zenodo record (`cites`, `isDerivedFrom`, `isSupplementTo`). If credit stops at the first hop, it is lost at the second hop.

[**ANALYSIS**] The rules do not ask for an unusual thing: a careful graduate student does the same. The differences are scale and verifiability. A person who forgets a citation forgets one, but an agent that forgets forgets thousands. An agent that invents references poisons the record. Thus, Paper Fabric enforces the rule 2 prohibition of invented references mechanically, with a reference log that records how each entry was checked.

8 Paper Fabric: an ouroboric factory for papers

Paper Fabric is the open pipeline that produced this preprint. Its design permits reuse for any paper, and it follows the philosophy of Ouroboric, the project of Bruno Valério. The Ouroboric charter describes "open, self-improving factories" that produce work, measure it and apply what they learn to themselves (Ouroboric DEV 2026). The charter lists a research factory "for scientific research and papers" as a possible future direction. The charter explicitly states that this direction is not a commitment (Ouroboric DEV 2026). Paper Fabric is an independent experiment in that direction.

8.1 Principles

Paper Fabric uses four principles of Ouroboric directly. These ideas are from Valério. *Humans steer, agents build*: I approve the direction and every claim that is based on my own experience, and agents draft, check and build. *Checks, not promises*: citations are checked

against the source, and the build fails if a citation is missing. *Read before you borrow*: licences decide what may be reused, and each thing that is taken gets credit. *Learn in public*: the pipeline, the prompts and the agent configuration that produced a paper are published with it.

8.2 Pipeline

The toolchain follows the books of Sandy Maguire (Maguire 2019). The prose is in Markdown, with one file for each section. A `justfile` runs pandoc with a custom LaTeX template and a BibTeX bibliography, and then runs XeLaTeX. The `justfile` then stamps the PDF with the Git revision that the build used. A Nix flake sets each tool to one version, through the `paper` lab of picasso (Matheus Hoffmann 2026b). Pandoc's built-in citeproc resolves the citations. The build treats warnings as errors, so an unresolved citation key stops the build.

Each paper is in its own directory, `papers/<slug>/`, with a `metadata.md`, numbered prose files and a `refs.bib`. The citation metadata for the repository is in `.zenodo.json`, and `CITATION.cff` is generated from that file. I borrowed this metadata, with credit, from my earlier projects (Matheus Hoffmann 2026b; Matheus Hoffmann et al. 2026).

A release is a Git tag. For v0, the release workflow builds the PDF and attaches it to a GitHub release. If a maintainer supplies a Zenodo token, and only then, the workflow uploads *only the PDF* to a draft deposit on Zenodo. The publication of the deposit mints a permanent DOI, and this step stays a human decision.

8.3 The loop

[**ANALYSIS**] The pipeline is ouroboric, not only automated, because each paper gives input to the next paper. The repository keeps the reference log, the readiness assessment and the exported agent configuration. The next paper starts from these files. The journal records the problems. The readiness assessment of the repository lists the known defects of this paper. This assessment is part of the same pull request that introduced the paper.

9 Related work by the same author

The argument of this paper also describes how I worked from the time of the closed thesis of Section 3 until now. Each project below is open source and has an archive on Zenodo with a concept DOI. Each project also has machine-readable citation metadata.

- **WW3 GPU Lab** (Matheus Hoffmann 2026c): a book, not yet complete, about agentic coding and agentic research. Its subject is the port of kernels of the WAVEWATCH III spectral wave model to GPUs with coding agents. A bit-for-bit parity harness compares the port with the Fortran reference.
- **marola** (Matheus Hoffmann et al. 2026): an open, non-profit platform for sea conditions and bathing-water quality at Brazilian beaches. It uses public data. I am one of its creators.
- **picasso** (Matheus Hoffmann 2026b): self-contained Nix flakes that pin the toolchains of the projects above. These toolchains include a publication pipeline with pandoc and TeX Live. picasso is also a use case of a system that improves itself. When a project needs a tool, the need goes back into picasso as a new lab or a new tool. Then each project that uses picasso gets the tool when it updates. The `paper` and `gleam-website` labs, which build this paper and its website, came into picasso in this way.

- **nashkell** (Matheus Hoffmann 2026a): a Haskell pipeline that reads the declassified Nash–NSA letters with OCR and a language model. It stores its identifiers in Latin through a Git filter. Section 6 examines it as an example of obfuscation and of the limits of obfuscation.

One influence is not my work. The ouroboric ideas in this paper are from Bruno Valério. His project Ouroboric (Ouroboric DEV 2026) is about open factories that improve themselves, for software and companies. Its documentation standard, its writing styles and its readiness checklist gave Paper Fabric its form.

[AUTHOR ANECDOTE: why the author has worked in the open for years, in the author's own words. For example, what changed after Chemo-RX, and what open work has returned to the author.]

For prior work by other authors on the subjects of this paper, refer to these four sources. Raymond wrote about the social method of open source (Raymond 1998). Samuelson and Scotchmer wrote about the law and economics of reverse engineering (Samuelson and Scotchmer 2002). Smith, Katz and Niemeyer wrote about software citation (Smith et al. 2016). Barker and colleagues wrote about FAIR research software (Barker et al. 2022).

10 Limitations

I state the limits of my argument clearly. This agrees with the Ouroboric value of honesty about results.

- **Position, not measurement.** This paper uses cited evidence for its argument. It does not report a new experiment. Its claims about the cost of reverse engineering come from the cited studies, not from my own measurements.
- **Allegations, not findings.** The Apple v. OpenAI dispute and the distillation accusations are claims by interested parties. When I wrote this paper, there was no decision about them from a court or an independent body.
- **Secondary sources for 2026 events.** For the 2026 events, my sources are news reports and one summary from a law firm. I did not fetch the court docket directly. Secondary sources do not agree about the judge suffix of the case number.
- **Not all closedness is wrong.** Patient data, personal data, security-sensitive details and material under embargo have good reasons to stay closed. My argument is about methods, code and models whose secrecy protects only their owner.
- **Author's interest.** I build open-source tools and cite my own projects. Readers should consider this interest when they evaluate Section 9.
- **Not yet peer reviewed.** This is a v0 preprint.

11 Conclusion

Closed source asked researchers to trust secrecy to protect their work. Between 2024 and 2026, that trust failed in public. Language models decompiled binaries, and production models were read through their own answers. Capability was copied through the front door. The biggest trade-secret dispute in the field was about laptops and job interviews. Linux and Git showed long ago that the alternative works at scale.

What remains is a question of manners, and machines make it urgent. If anyone can reverse engineer anything, the honest course is this: publish the method, mint an identifier, and

cite all that you use. The course also includes a requirement: the agents that now read and write much of the text must do the same. Paper Fabric is my attempt to make that course the default for my own work. Its source, its agent configuration and its known defects are published with this paper.

Statements

Authorship. The cover gives the name Mr. Nobody. I, Matheus Hoffmann, wrote this paper in the first person. The citation metadata gives my name.

Acknowledgements. The ouroboric ideas in this paper are from Bruno Valério (Ouroboric DEV 2026).

Use of AI. A Claude coding agent (Anthropic) produced the structure, prose drafts, reference checks, build tools and repository configuration of this v0. The agent worked in a Claude project, under my direction. The agent fetched and checked each reference when the paper was written. The reference log of the repository records these checks. The agent configuration is published in the `.claude/` directory of the repository. I am responsible for the content.

Competing interests. I maintain the open-source projects that Section 9 cites. Section 4 cites a report by Anthropic. Anthropic also makes the model that drafted this paper. I declare no other competing interests.

Availability. Source, build pipeline and agent configuration: https://github.com/h0ffmann/closed-source. Licence: text CC BY 4.0; build scripts AGPL-3.0-only; LaTeX template MIT.

References

Anthropic. 2026. “Detecting and Preventing Distillation Attacks.” https://www.anthropic.com/news/detecting-and-preventing-distillation-attacks.

Barak, Boaz, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. 2012. “On the (Im)possibility of Obfuscating Programs.” *Journal of the ACM* 59 (2): 1–48. https://doi.org/10.1145/2160158.2160159.

Barker, Michelle, Neil P. Chue Hong, Daniel S. Katz, et al. 2022. “Introducing the FAIR Principles for Research Software.” *Scientific Data* 9: 622. https://doi.org/10.1038/s41597-022-01710-x.

Branwen, Gwern. 2012. “John Nash’s 1955 Letters to the NSA: Transcription.” https://gwern.net/doc/cs/cryptography/nash/1955-nash.

Bridge Counsels. 2026. “Inside the Apple Inc. v. OpenAI Dispute: Trade Secrets, Contracts, and AI Innovation.” https://bridgecounsels.com/inside-the-apple-inc-v-openai-dispute-trade-secrets-contracts-and-ai-innovation/.

Carlini, Nicholas, Daniel Paleka, Krishnamurthy Dj Dvijotham, Thomas Steinke, Jonathan Hayase, A. Feder Cooper, Katherine Lee, et al. 2024. “Stealing Part of a Production Language Model.” In *Proceedings of the 41st International Conference on Machine Learning*, 235:5680–5705. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v235/carlini24a.html.

Chacon, Scott, and Ben Straub. 2014. “Git Tools: Signing Your Work.” Pro Git, 2nd edition, Apress. https://git-scm.com/book/en/v2/Git-Tools-Signing-Your-Work.

Hanley, Mike. 2022. “Software Security Starts with the Developer: Securing Developer Accounts with 2FA.” The GitHub Blog. https://github.blog/2022-05-04-software-security-starts-with-the-developer-securing-developer-accounts-with-2fa/.

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. 2015. “Distilling the Knowledge in a Neural Network.” https://arxiv.org/abs/1503.02531.

Hoffmann, Manuel, Frank Nagle, and Yanuo Zhou. 2024. "The Value of Open Source Software." Strategy Unit Working Paper 24-038. Harvard Business School. https://papers.ssrn.com/abstract=4693148.

Hoffmann, Matheus. 2018. "Chemo-RX: Statistical Methods to Predict Adverse Reactions to Chemotherapy for Breast Cancer." GitHub repository. https://github.com/h0ffmann/chemo-rx.

Hoffmann, Matheus et al. 2026. "Marola: An Open, Non-Profit Platform for Sea Conditions and Bathing-Water Quality at Brazilian Beaches, Built on Public Data." Zenodo. https://doi.org/10.5281/zenodo.23224155.

Hoffmann, Matheus. 2026a. "Nashkell: An Algebraic-Effects Pipeline for OCR and LLM-Checked Annotation of the Nash–NSA Letters." GitHub repository. https://doi.org/10.5281/zenodo.23356739.

———. 2026b. "Picasso: Self-Contained Nix Flakes for Scientific and Agentic Toolchains, and the NixOS Workstation They Run On." Zenodo. https://doi.org/10.5281/zenodo.23311208.

———. 2026c. "WW3 GPU Lab: Agentic Coding and Agentic Research on a Bit-for-Bit GPU Port of WAVEWATCH III." Zenodo. https://doi.org/10.5281/zenodo.23221351.

Ince, Darrel C., Leslie Hatton, and John Graham-Cumming. 2012. "The Case for Open Computer Programs." *Nature* 482 (7386): 485–88. https://doi.org/10.1038/nature10836.

Kerckhoffs, Auguste. 1883. "La cryptographie militaire." *Journal des sciences militaires* IX: 5–38, 161–91. https://www.petitcolas.net/kerckhoffs/.

Maguire, Sandy. 2019. "Thinking with Types: Book Sources and Build." GitHub repository isovector/thinking-with-types. https://github.com/isovector/thinking-with-types.

Nash, John F. 1950. "Equilibrium Points in n-Person Games." *Proceedings of the National Academy of Sciences* 36 (1): 48–49. https://doi.org/10.1073/pnas.36.1.48.

Nash, John F., and National Security Agency. 1955. "Correspondence Regarding Cryptography Between John Nash and the National Security Agency." https://media.defense.gov/2021/Jul/12/2002760581/-1/-1/0/NASH_LETTERS1.PDF.

National Security Agency. 2019. "Ghidra: A Software Reverse Engineering Framework." Source code, Apache-2.0. https://github.com/NationalSecurityAgency/ghidra.

Newman, Zachary, John Speed Meyers, and Santiago Torres-Arias. 2022. "Sigstore: Software Signing for Everybody." In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2353–67. https://doi.org/10.1145/3548606.3560596.

OpenSSH project. 2020. "OpenSSH 8.2 Release Notes." https://www.openssh.org/txt/release-8.2.

Ouroboric DEV. 2026. "Ouroboric: Open Factories That Improve Themselves." GitHub repository ouroboric-dev/ouroboric. https://github.com/ouroboric-dev/ouroboric.

Perez, Sarah. 2026a. "The Wildest Allegations in Apple's Trade Secrets Lawsuit Against OpenAI." TechCrunch. https://techcrunch.com/2026/07/13/the-wildest-allegations-in-apples-trade-secrets-lawsuit-against-openai/.

———. 2026b. "OpenAI Says Apple's Own Security Practices Undermine Its Trade Secrets Case." TechCrunch. https://techcrunch.com/2026/08/06/openai-says-apples-own-security-practices-undermine-its-trade-secrets-case/.

Pirsig, Robert M. 1974. *Zen and the Art of Motorcycle Maintenance: An Inquiry into Values*. New York: William Morrow. https://openlibrary.org/isbn/0688002307.

Raymond, Eric S. 1998. "The Cathedral and the Bazaar." *First Monday* 3 (2). https://doi.org/10.5210/fm.v3i2.578.

Reuters. 2026. "OpenAI Claims China's DeepSeek Trained Its AI by Distilling U.S. Models, Memo Shows." Republished by Korea JoongAng Daily. https://www.koreajoongangdaily.com/world/openai-claims-chinas-deepseek-trained-its-ai-by-distilling-us-models-memo-shows/12461508.

Rivest, Ronald L., Adi Shamir, et al. 2012. "Analysis of the Cryptosystem Proposed by John Nash." Rump session talk, EUROCRYPT 2012. https://www.iacr.org/conferences/eurocrypt2012/Rump/nash.pdf.

Samuelson, Pamela, and Suzanne Scotchmer. 2002. "The Law and Economics of Reverse Engineering." *Yale Law Journal* 111 (May): 1575. https://www.yalelawjournal.org/article/the-law-and-economics-of-reverse-engineering.

Smith, Arfon M., Daniel S. Katz, Kyle E. Niemeyer, and FORCE11 Software Citation Working Group. 2016. "Software Citation Principles." *PeerJ Computer Science* 2: e86. https://doi.org/10.7717/peerj-cs.86.

Stevens, Marc, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. 2017. "The First Collision for Full SHA-1." In *Advances in Cryptology – CRYPTO 2017*. Vol. 10401. Lecture

Notes in Computer Science. Springer. https://doi.org/10.1007/978-3-319-63688-7_19.

Tan, Hanzhuo, Qi Luo, Jing Li, and Yuqun Zhang. 2024. "LLM4Decompile: Decompiling Binary Code with Large Language Models." In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.emnlp-main.203.

The Git project. 2020. "Git 2.29 Release Notes." https://github.com/git/git/blob/master/Documentation/RelNotes/2.29.0.adoc.

———. 2021. "Git 2.34 Release Notes." https://github.com/git/git/blob/master/Documentation/RelNotes/2.34.0.adoc.

Torvalds, Linus. 1991. "What Would You Like to See Most in Minix?" Usenet post to comp.os.minix. https://groups.google.com/g/comp.os.minix/c/dlNtH7RRrGA/m/SwRavCzVE7gJ.

———. 2005. "Initial Revision of "Git", the Information Manager from Hell." Git commit e83c5163316f89bfbde7d9ab23ca2e25604af290. https://github.com/git/git/commit/e83c5163316f89bfbde7d9ab23ca2e25604af290.

Tramèr, Florian, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. "Stealing Machine Learning Models via Prediction APIs." In *25th USENIX Security Symposium (USENIX Security 16)*, 601–18. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer.

Wikipedia contributors. 2026a. "Git." Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Git.

———. 2026b. "History of Linux." Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/History_of_Linux.

Wilkinson, Mark D., Michel Dumontier, IJsbrand Jan Aalbersberg, et al. 2016. "The FAIR Guiding Principles for Scientific Data Management and Stewardship." *Scientific Data* 3: 160018. https://doi.org/10.1038/sdata.2016.18.