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October 2026 · v0 (preprint, not peer reviewed) · rev. 6b3aae5+219





Abstract. For thirty years, closed source has promised that secrecy protects value. I argue that the promise has failed, for science first. I read the case through Robert M. Pirsig's *Zen and the Art of Motorcycle Maintenance* (1974), the book this work stands on: the number of rational hypotheses that can explain a phenomenon is infinite, so science never closes, and a method that only its owner can open is one that nobody else can test. Three trends from 2024 to 2026 make the case. Large language models now decompile binaries into re-executable source. Black-box queries extract parts of production models, and distillation copies capabilities at industrial scale. The largest 2026 trade-secret dispute in artificial intelligence, Apple's suit against OpenAI, turns on people, laptops and recruiting, not on code that stayed secret. Against this background I revisit Linus Torvalds' Linux and Git as evidence that openness scales, and my closed undergraduate thesis as evidence of what secrecy costs a researcher. I review obfuscation, including my own, and why it cannot hide code, and I contrast it with John Nash's 1955 idea of security from computational cost: signed commits, keys held in hardware and two-factor authentication protect what secrecy cannot. I then argue that the ethical answer to universal reverse engineering is professional citation: persistent identifiers, machine-readable citation metadata and agents that borrow, cite and propagate credit by default. I describe Paper Fabric, an open, self-improving pipeline in the Ouroboric style that produced this preprint, and 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 had left the Communist movement published their reasons in a book called *The God That Failed*. The title of this paper borrows the phrase for a smaller and less tragic faith: the belief that keeping source code, models and methods secret is what keeps them valuable. I call that belief *closed source* in a broad sense. It covers proprietary software, unpublished research code, model weights behind an API and the scripts behind a thesis that only its author has seen.

The faith has two articles. The first says that secrecy protects the owner: what nobody can read, nobody can copy. The second says that secrecy is costless to everyone else: the world loses little when one lab, one company or one student keeps their work to themselves. I argue that both articles failed, and that the failure became obvious between 2024 and 2026.

The first article failed because reverse engineering stopped being rare. Reverse engineering has always been lawful and common (Samuelson and Scotchmer 2002). What changed is its cost. Language models now turn compiled binaries into source 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 a frontier model's behaviour into a competitor's (Anthropic 2026; Reuters 2026). And when one of the most valuable

companies in the world sued one of the most valuable laboratories over trade secrets in July 2026, its complaint was about people, laptops and recruiting, 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. A result that cannot be inspected cannot be reproduced (Ince, Hatton, and Graham-Cumming 2012), and code that cannot be found cannot be credited (Smith et al. 2016). A whole economy now runs on code that someone chose to open: one estimate puts the cost of rebuilding the open source that firms use at about 8.8 trillion US dollars (Manuel Hoffmann, Nagle, and Zhou 2024).

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

[**ANALYSIS**] If the hypotheses have no end, science has no end, and no single laboratory can test them all. A method that only its owner can open is a method that nobody else can test. That is why I hold that openness comes first for science, before it comes for business.

If anyone can reverse engineer anything, the useful question is no longer *how do I hide my work?* It is *how do I make sure my work is credited when it is used?* My answer is professional citation: persistent identifiers, machine-readable metadata and, increasingly, agents that cite by default. That answer has a practical side, so I also describe the open pipeline that produced this paper.

The paper is organised as follows. Section 2 revisits Linux and Git as evidence that openness scales. 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 gathers five lines of evidence that closed source is dead as a protection. Section 6 turns to obfuscation, John Nash's letters on cryptography and how to secure source code in 2026. Section 7 argues for professional citation and states rules for agents. Section 8 describes Paper Fabric. Section 9 lists my related work, and Section 10 states the limits of my argument.

A note on method. This is a position paper, written in the *research* style of the Ouroboric documentation standard: every factual claim carries a citation, and passages of interpretation are marked [**ANALYSIS**]. It is a v0 preprint and has not been peer reviewed.

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

On 25 August 1991, a student at the University of Helsinki posted to the Usenet group `comp.os.minix` 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, and asked what people would like to see in it (Torvalds 1991). Version 0.01 followed on 17 September 1991. In early 1992 Torvalds moved the kernel to the GNU General Public License, and version 0.95, in March 1992, was the first release under it (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 to make the point; the reader's own week almost certainly passed through several Linux machines.

Torvalds' first achievement was not only technical. Eric Raymond's account of Linux development, *The Cathedral and the Bazaar*, identified the social method behind it: release early, release often, treat users as co-developers, and let many eyes find bugs that few would (Raymond 1998). Raymond contrasted this *bazaar* with the *cathedral* of carefully guarded,

centrally planned development. Linux showed that the bazaar could build the hardest kind of software there is.

Torvalds' second achievement came from a failure of closed source. From 2002 the kernel used BitKeeper, a proprietary version-control system offered at no cost. In April 2005 that free access ended. No free system met the project's needs, so Torvalds wrote his own. He began on 3 April 2005; on 7 April Git was managing its own source code; the first multi-branch merge followed on 18 April; and on 26 July he handed maintenance to Junio Hamano (Wikipedia contributors 2026a). The first commit in the Git repository, still readable today, is titled "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 had accepted a closed tool for convenience. When the owner withdrew it, the community was exposed; when Torvalds replaced it with an open one, the replacement became the substrate of nearly all software development, including the repositories, pull requests and release tags through which this paper is built and archived. Git also embodies, in its data model, the ethic I argue for in Section 7: every commit names its author and points, by cryptographic hash, to the exact history it builds on. Git is a citation graph for code.

I praise Torvalds here without reservation for those two gifts and for three decades of maintaining the first. I also note what the praise does not require: that one person's technical style be copied, or that open source be free of conflict. The claim is narrower and stronger. Openness, at the largest scale ever attempted, worked.

3 A closed-source confession: Chemo-RX

This section is written in the first person by Matheus Hoffmann. Passages marked [**AUTHOR ANECDOTE**] are placeholders for the author's own recollections. They must be written by the author before this paper leaves v0; nothing in them may be invented by an agent.

In 2018 I started an undergraduate thesis at the Escola Politécnica of the Universidade Federal do Rio de Janeiro, in association with the LESS laboratory of the Biomedical Engineering Programme. Its alias was Chemo-RX. The goal was to compare statistical methods for predicting adverse reactions to chemotherapy in women treated for breast cancer, using data from Brazil's National Cancer Institute (INCA). The reactions in scope were asthenia, myalgia, arthralgia, mucositis, abdominal pain and diarrhoea, and 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 holds a README, a proposal in OpenDocument format and an R Markdown data summary. The README has three headings that were never filled in: "Proposal EN link", "Final project EN link" and "Thesis full content". Under the last one it says only "contact me". The proposal's to-do list still includes "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] Two kinds of closedness are mixed in that repository, and the distinction matters for the rest of this paper. The patient data could not be published, and should not have been: clinical records are personal data, and their release is governed by ethics approval and data agreements, which the to-do list itself acknowledges. The *method* could have been published: the statistical code, the comparison protocol, the variable definitions and the negative results. The FAIR principles make exactly this distinction. Data and software should be findable and accessible *under well-defined conditions*, which may include restricted access, and their metadata should remain public even when the data are not (Wilkinson et al. 2016; Barker et al. 2022). Closing the method along with the data protected no patient. It only made the work uncitable.

[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 *reverse engineering* (REA in my project's shorthand, for reverse engineering, assisted) for any process that recovers how an artefact works from the artefact itself, without access to its design documents: decompiling a binary, inferring a model from its outputs, or rebuilding a product from its parts. This section reviews four developments. The first is a lawsuit, and its facts are allegations; I report them as such.

4.1 Apple v. OpenAI: secrets walk out on two legs

On 10 July 2026 Apple sued OpenAI, the device company io that OpenAI had acquired, and two former Apple employees in 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 a colleague's Apple-issued computer; OpenAI's chief hardware officer, a former Apple executive, asked job candidates to bring Apple parts, design files and prototypes to interviews; and new hires received guidance on avoiding 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 it moved to dismiss, arguing that Apple let employees use personal cloud accounts for work, did not revoke access properly, and had not identified which secrets were taken (Perez 2026b). At the time of writing, the motion to dismiss and a motion for a preliminary injunction were pending, and no court had ruled on the merits (Bridge Counsels 2026; Perez 2026b).

[ANALYSIS] Whatever the outcome, the dispute is instructive for what it is *not* about. It is not about source code that an adversary failed to read because it was secret. Every alleged path runs through people: their accounts, their devices, their recruiting conversations and their memories. Secrecy, in this account, was a property of an organisation's processes, and processes leak. Note also that OpenAI's defence, as reported, is partly that Apple's own

practices did not treat the information as secret. A secret that must be actively guarded at every exit interview is expensive; the cost of the guard is part of the price of closed source.

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

4.2 Distillation: copying capability through the front door

Knowledge distillation, training a smaller model to imitate a larger one's outputs, is a standard and legitimate technique (Hinton, Vinyals, and Dean 2015). In 2026 it also became the centre of an accusation of industrial copying. In February, according to Reuters, OpenAI told the US House Select Committee on Strategic Competition between the United States and the Chinese Communist Party that accounts associated with DeepSeek employees had developed methods to circumvent OpenAI's access restrictions and "obtain outputs for distillation in programmatic ways" (Reuters 2026). DeepSeek did not respond to Reuters. Ten days later Anthropic reported "industrial-scale campaigns by three AI laboratories", DeepSeek, Moonshot AI and MiniMax, using about 24,000 fraudulent accounts and more than 16 million exchanges with its Claude models (Anthropic 2026).

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

4.3 Extraction: reading the model through its answers

Academic work made the same point with more precision and less drama. In 2016 Tramèr and colleagues showed that machine-learning models served through prediction APIs can be stolen by querying them (Tramèr et al. 2016). In 2024 Carlini and colleagues went further against 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, and confirmed their hidden dimensions for the first time (Carlini et al. 2024). The paper, published at ICML 2024, closes with defences and mitigations: the response of a research community to discovered leakage is to publish it, so that it can be fixed.

4.4 Decompilation: the end of the opaque binary

Reverse engineering of compiled software has long been routine and, for purposes such as interoperability, lawful (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). Language models then lowered the remaining barrier, which was skill. LLM4Decompile, a family of open models trained for decompilation, produces source 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] Put these four developments together and the cost curve of reverse engineering has collapsed along every axis: for binaries (decompilation), for models (extraction and distillation) and for organisations (people moving between employers). I call this the *REA*

explosion. It does not make every artefact transparent overnight, and it does not make copying lawful where the law forbids it. It does make secrecy a weak and expensive defence, and it makes the question of credit, who did what first and who built on whom, 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 keeping an artefact unreadable keeps its value safe, and that this protection is worth what it costs science. Five references, from cryptography, law, machine learning, software engineering and economics, support that 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 the system "must not require secrecy and can be stolen by the enemy without causing trouble" (Kerckhoffs 1883). Modern cryptography is built on the principle: the algorithms are public, and only keys are secret. A design whose security depends on the design staying hidden is considered broken in advance.

2. The law has long accepted reverse engineering. Samuelson and Scotchmer open their survey with the observation that reverse engineering "has a long history as an accepted practice" (Samuelson and Scotchmer 2002). Trade-secret law protects against improper means, such as theft and breach of confidence, not against someone who studies a product they lawfully obtained. Closed source therefore never had a legal guarantee against being understood. It had a practical one: reverse engineering was slow and expensive.

3. That practical guarantee is gone for models. Extraction attacks on prediction APIs (Tramèr et al. 2016) matured into attacks on production language models (Carlini et al. 2024), and distillation at scale is now alleged by the providers themselves (Anthropic 2026; Reuters 2026).

4. It is going for binaries. Open decompilation models, built on an open reverse-engineering framework released by a signals-intelligence agency (National Security Agency 2019), recover executable source from compiled code (Tan et al. 2024).

5. Meanwhile, 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). The economic weight of the alternative is now measurable: the open source that firms use would cost an estimated 4.15 billion US dollars to write once, and about 8.8 trillion US dollars if each firm had to replace it on its own (Manuel Hoffmann, Nagle, and Zhou 2024).

[**ANALYSIS**] Lines 1 to 4 say that secrecy no longer protects; line 5 says that it never came 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 choosing the second.

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

If secrecy fails as a protection, what is left for code that must stay private, and for the public code that others depend on? This section looks at one tempting answer, obfuscation, and at the answer that cryptography gives instead.

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

In early 1955 John Nash, the author of the equilibrium concept for non-cooperative games (Nash 1950), wrote to the US National Security Agency to propose an enciphering machine (Nash and National Security Agency 1955). 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 rest on the cost of computation rather than on hiding the design. Rivest, Shamir and their students read the letters as an anticipation of complexity theory and modern cryptography (Rivest, Shamir, et al. 2012). Nash's own machine fared less well. The NSA judged that it "affords only limited security" (Nash and National Security Agency 1955), and when Ronald Rivest, Adi Shamir and their students analysed it in 2012 they found a polynomial-time chosen-plaintext attack and a subexponential known-plaintext attack, and concluded that Nash was "a much better game theorist than a cryptographer" (Rivest, Shamir, et al. 2012).

[**ANALYSIS**] Both halves of the story support Kerckhoffs. Nash's principle, security from the work an attacker must do, is the one modern cryptography kept. His design, once published, was judged and broken by others. That is how cryptography is supposed to work: the method is open, and only the key is secret.

6.2 Obfuscation: what it buys and what it cannot

Obfuscation transforms a program so that it does the same thing but is harder to read. One of the I built a small, honest example. *nashkell*, a Haskell pipeline that runs OCR on Nash's declassified letters and asks a language model to annotate and cross-check them, stores its source in Git with identifiers translated into Latin (Matheus Hoffmann 2026a). A Git `clean` filter translates English identifiers to Latin on commit, and a `smudge` filter translates them back on checkout, through a fixed dictionary; anyone who clones without the filter sees, and GitHub shows, the Latin form.

The project's own documentation lists the limits, and they are general (Matheus Hoffmann 2026a):

- **Structure survives.** Only top-level names change. Control flow, types and local variables stay intact and inferable, so the result "raises, not eliminates, reading difficulty" and is "not a security boundary".
- **The key is the dictionary.** The mapping is a bijection stored next to the tool. Whoever has it, or can guess enough of it, undoes the transformation.
- **The stored form does not build.** File paths are not renamed, so the Latin modules no longer match their paths. Obfuscation costs the owner too.
- **It is a heuristic.** The filter is line-based, not a parser; it does not handle block comments or multi-line strings, and new names must be added to the dictionary by hand.

Theory says these limits are not a matter of effort. 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 they could not compute from input-output access alone. The authors built a family of programs that can be reverse engineered from any equivalent program, although black-box access reveals nothing of their code (Barak et al. 2012). The result leaves room for weaker notions of obfuscation, but not for a general method that makes source code secret. **[ANALYSIS]** And against the language models of Section 4, which already recover source from binaries, renaming 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 to stop protecting the *readability* of code and to protect what cryptography can protect: that code is what its authors wrote, and that only they could have written it. In 2026 that means four layers, each with public, mature tools.

1. **Sign commits and tags.** Git signs tags and commits with GnuPG, and `git merge --verify-signatures` refuses history without 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, FIDO security keys back the `ecdsa-sk` and `ed25519-sk` key types; the private key never leaves the token, and the token must be present when the key is used (OpenSSH project 2020). A stolen laptop no longer means a stolen signing key.
3. **Require two-factor authentication on the forge.** GitHub committed in 2022 to require every user who contributes code on GitHub.com to enable two-factor authentication by the end of 2023 (Hanley 2022). A person who takes over an account can push code in its owner's name.
4. **Make signatures verifiable by everyone.** Sigstore lets developers sign with ephemeral keys bound to an existing identity through OpenID Connect, and records signatures in transparency logs, which removes most of the burden and risk of key management (Newman, Meyers, and Torres-Arias 2022).

Hash functions matter too. In 2017 Stevens and colleagues produced the first collision for full SHA-1 (Stevens et al. 2017), the hash on which Git's object names were built. Git has offered a SHA-256 object format since version 2.29, first marked experimental (The Git project 2020).

What stays secret under this model is small and explicit: private keys, credentials, and data that is personal or under agreement, as in Section 3. Those deserve encryption and access control. The source code itself is better published, signed and cited.

[**ANALYSIS**] In Nash's own vocabulary, this is a change of strategy in a game whose payoffs have 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, while the payoff of credited, verifiable publication does not. I do not claim to have 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, the remaining protection is the norm that whoever uses it says so. That norm is old in science; it is young for software, and it is barely formed for agents.

7.1 Citation for people

The FORCE11 *Software citation principles* set out what it means to cite software properly: software is a legitimate product of research and should be cited as such; citations should give credit, be unique and persistent, be accessible, and be specific to the version used (Smith et al. 2016). The FAIR principles ask that research outputs be findable, accessible, interoperable and reusable (Wilkinson et al. 2016), and FAIR4RS adapts them to research software (Barker et al. 2022).

In practice this needs three things in every repository. The first is a persistent identifier: a DOI minted by an archive such as Zenodo, with 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 citation can be generated rather than typed: `CITATION.cff` for GitHub's "Cite this repository", and `.zenodo.json` for the archive. The third is a release process in which every published version gets its own identifier, so that a reader can cite exactly what they used.

This repository follows that recipe; Section 8 gives the details. So does the related work in Section 9, each item of which has a concept DOI.

7.2 Citation for agents: borrow, cite, propagate

Coding and research agents now write a large share of new code and text, including the drafting of this paper (see the AI-use statement). Agents read far more than any person and copy fluently. If they do not cite, the REA explosion becomes a laundering machine: code and ideas go in with authors and come out without them. I therefore propose three rules for agents, which Paper Fabric's `AGENTS.md` makes binding for any agent that works in its repositories.

1. **Borrow only what the licence allows.** Before reusing code, text, figures or data, read the licence and record it. Reuse that the licence does not permit is not borrowing. When in doubt, describe the idea and cite it instead of copying its expression.
2. **Cite at the point of use, with a persistent identifier.** Every borrowed element gets a citation where it is used, not only in a final list. Prefer a DOI; fall back to a URL with a commit hash or archive snapshot. Never invent a reference: an agent that cannot fetch and check a source must mark the claim as unverified.

3. **Propagate credit downstream.** When an agent's output is itself published, the citations travel with it: in the bibliography, in `CITATION.cff` references, and in the Zenodo record's related identifiers (`cites`, `isDerivedFrom`, `isSupplementTo`). Credit that stops at the first hop is lost at the second.

[**ANALYSIS**] The rules ask nothing exotic. They are what a careful graduate student does. The difference is scale and verifiability: a person who forgets a citation forgets one; an agent that forgets forgets thousands, and an agent that invents references poisons the record. Rule 2's prohibition on invented references is therefore enforced mechanically in Paper Fabric, by 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. It is designed to be reused for any paper, and it follows the philosophy of Ouroboric, Bruno Valério's project, whose charter describes "open, self-improving factories" that produce work, measure it and feed what they learn back into themselves (Ouroboric DEV 2026). The Ouroboric charter lists a research factory "for scientific research and papers" as a possible future direction, explicitly not a commitment (Ouroboric DEV 2026). Paper Fabric is an independent experiment in that direction.

8.1 Principles

Paper Fabric takes four of Ouroboric's principles directly; the ideas are Valério's. *Humans steer, agents build*: I approve the direction and every claim that rests on my own experience; agents draft, check and build. *Checks, not promises*: citations are verified against the source, and the build fails on a missing citation. *Read before you borrow*: licences decide what may be reused, and credit is given for what is taken. *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 Sandy Maguire's books (Maguire 2019): prose is written in Markdown, one file per section, and a `justfile` runs pandoc with a custom LaTeX template and a BibTeX bibliography, then XeLaTeX, and stamps the PDF with the Git revision it was built from. A Nix flake pins every tool to one version, through the `paper` lab of picasso (Matheus Hoffmann 2026b). Pandoc's built-in citeproc resolves citations, and the build runs with warnings treated as errors, so an unresolved citation key stops the build.

Each paper lives in its own directory, `papers/<slug>/`, with a `metadata.md`, numbered prose files and a `refs.bib`. The repository-level citation metadata (`.zenodo.json`, from which `CITATION.cff` is generated) is borrowed, 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, attaches it to a GitHub release and, only when a maintainer supplies a Zenodo token, uploads *only the PDF* to a Zenodo draft deposit. Publishing the deposit, which mints a permanent DOI, stays a human decision.

8.3 The loop

[**ANALYSIS**] What makes the pipeline ouroboric rather than merely automated is that each paper feeds the next. The reference log, the readiness assessment and the exported agent configuration are kept in the repository. The next paper starts from them, and the journal records what went wrong. This paper's own known defects are listed in the repository's readiness assessment, which is part of the same pull request that introduced it.

9 Related work by the same author

The argument of this paper is also a description of how I have worked since the closed thesis of Section 3. Each project below is open source, archived on Zenodo with a concept DOI, and carries machine-readable citation metadata.

- **WW3 GPU Lab** (Matheus Hoffmann 2026c): a book in progress on agentic coding and agentic research, built around porting kernels of the WAVEWATCH III spectral wave model to GPUs with coding agents, under a bit-for-bit parity harness against the Fortran reference.
- **marola** (Matheus Hoffmann et al. 2026): an open, non-profit platform for sea conditions and bathing-water quality at Brazilian beaches, built on public data. I am among its creators.
- **picasso** (Matheus Hoffmann 2026b): self-contained Nix flakes that pin the toolchains used by the projects above, including a pandoc and TeX Live publishing pipeline. picasso is also a use case of a self-improving system: when a project needs a tool, the need goes back into picasso as a new lab or a new tool, and every project that uses picasso receives it on its next update. The `paper` and `gleam-website` labs, which build this paper and its website, came into picasso that way.
- **nashkell** (Matheus Hoffmann 2026a): a Haskell pipeline that reads the declassified Nash–NSA letters with OCR and a language model, and stores its identifiers in Latin through a Git filter; Section 6 discusses it as an example of obfuscation and its limits.

One influence is not my work. The ouroboric ideas in this paper are Bruno Valério's: Ouroboric (Ouroboric DEV 2026), open factories that improve themselves, for software and companies. Its documentation standard, writing styles and readiness checklist shaped Paper Fabric.

[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 others on the themes of this paper, see Raymond on the social method of open source (Raymond 1998), Samuelson and Scotchmer on the law and economics of reverse engineering (Samuelson and Scotchmer 2002), Smith, Katz and Niemeyer on software citation (Smith et al. 2016) and Barker and colleagues on FAIR research software (Barker et al. 2022).

10 Limitations

I state the limits of my argument plainly, in line with the Ouroboric value of honesty about results.

- **Position, not measurement.** This paper argues from cited evidence; it reports no new experiment. Its claims about the cost of reverse engineering rest on the studies cited, not on my own measurements.
- **Allegations, not findings.** The Apple v. OpenAI dispute and the distillation accusations are claims by interested parties. No court or independent body had ruled on them at the time of writing.
- **Secondary sources for 2026 events.** I relied on news reports and one law-firm summary for the 2026 events. The court docket was not retrieved directly, and secondary sources disagree on the judge's 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 concerns 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 weigh Section 9 accordingly.
- **Not yet peer reviewed.** This is a v0 preprint.

11 Conclusion

Closed source asked researchers to trust that secrecy would protect their work. Between 2024 and 2026 that trust failed in public: binaries decompiled by language models, production models read through their own answers, capability copied through the front door, and the biggest trade-secret fight in the field fought over laptops and job interviews. Linux and Git showed long ago that the alternative works at scale.

What remains is a question of manners, made urgent by machines. If anyone can reverse engineer anything, the honest course is to publish the method, mint an identifier, and cite whatever you use, and to require the same of the agents that now do much of the reading and writing. 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 names Mr. Nobody. I, Matheus Hoffmann, wrote this paper in the first person, and the citation metadata names me.

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

Use of AI. The structure, prose drafts, reference checks, build tooling and repository configuration of this v0 were produced by a Claude coding agent (Anthropic) working in a Claude project, under my direction. Every reference was fetched and checked by the agent at the time of writing; the checks are recorded in the repository's reference log. The agent configuration is published in the repository's `.claude/` directory. I am responsible for the content.

Competing interests. I maintain the open-source projects cited in Section 9. Anthropic, whose report is cited in Section 4, 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.