Código Fechado: O Deus que Falhou

Engenharia reversa, espionagem e a ética da citação na era dos agentes de programação

Mr. Nobody

Outubro de 2026 · v0, edição em português do Brasil (preprint, sem revisão por pares) · rev. 6b3aae5+219





Resumo. Por trinta anos, o código fechado prometeu que o segredo protege valor. Eu argumento que essa promessa falhou, e falhou primeiro para a ciência. Leio o caso com o livro em que esta obra se apoia, *Zen and the Art of Motorcycle Maintenance* (1974), de Robert M. Pirsig: o número de hipóteses racionais que podem explicar um fenômeno é infinito, então a ciência nunca se fecha, e um método que só o dono pode abrir é um método que mais ninguém pode testar. Três tendências de 2024 a 2026 sustentam o argumento. Grandes modelos de linguagem agora descompilam binários em código-fonte que volta a executar. Consultas de caixa-preta extraem partes de modelos em produção, e a destilação copia capacidades em escala industrial. A maior disputa de segredo industrial de 2026 em inteligência artificial, o processo da Apple contra a OpenAI, gira em torno de pessoas, notebooks e recrutamento, não de código que permaneceu secreto. Com esse pano de fundo, revisito o Linux e o Git de Linus Torvalds como evidência de que a abertura escala, e o meu trabalho de conclusão de curso fechado como evidência do que o segredo custa a um pesquisador. Examino a ofuscação, inclusive a minha, e por que ela não consegue esconder código, e a contraponho à ideia de John Nash, de 1955, de segurança pelo custo computacional: commits assinados, chaves guardadas em hardware e autenticação de dois fatores protegem o que o segredo não protege. Em seguida, argumento que a resposta ética à engenharia reversa universal é a citação profissional: identificadores persistentes, metadados de citação legíveis por máquina e agentes que pegam emprestado, citam e propagam o crédito por padrão. Descrevo o Paper Fabric, um pipeline aberto que melhora a si mesmo, no estilo Ouroboric, que produziu este preprint, e listo as suas próprias limitações conhecidas.

Palavras-chave: código aberto; engenharia reversa; engenharia reversa assistida por IA; destilação de modelos; segredos industriais; citação de software; software de pesquisa; FAIR; agentes de programação.

1 Introdução

Em 1949, seis escritores que haviam deixado o movimento comunista publicaram suas razões em um livro chamado *The God That Failed*. O título deste artigo toma a expressão emprestada para uma fé menor e menos trágica: a crença de que manter em segredo o código-fonte, os modelos e os métodos é o que os mantém valiosos. Chamo essa crença de *código fechado* em sentido amplo. Ela abrange o software proprietário, o código de pesquisa não publicado, os pesos de modelos por trás de uma API e os scripts por trás de uma tese que só o seu autor viu.

Essa fé tem dois artigos. O primeiro diz que o segredo protege o dono: o que ninguém consegue ler, ninguém consegue copiar. O segundo diz que o segredo não custa nada a mais ninguém: o mundo perde pouco quando um laboratório, uma empresa ou um estudante guarda o próprio trabalho para si. Eu argumento que os dois artigos fracassaram, e que o fracasso se tornou evidente entre 2024 e 2026.

O primeiro artigo fracassou porque a engenharia reversa deixou de ser rara. A engenharia reversa sempre foi lícita e comum (Samuelson e Scotchmer 2002). O que mudou foi o seu custo. Modelos de linguagem hoje transformam binários compilados em código-fonte que compila e passa nos testes (Tan et al. 2024). Alguns milhares de consultas a uma API recuperam partes de um modelo de linguagem em produção (Carlini et al. 2024). Dezenas de milhares de contas conseguem destilar o comportamento de um modelo de fronteira no de um concorrente (Anthropic 2026; Reuters 2026). E quando uma das empresas mais valiosas do mundo processou um dos laboratórios mais valiosos por segredos comerciais em julho de 2026, sua petição tratava de pessoas, laptops e recrutamento, e não de código-fonte que tivesse permanecido em segredo e a salvo (Perez 2026a; Bridge Counsels 2026).

O segundo artigo fracassou porque os custos do segredo recaem sobre a ciência. Um resultado que não pode ser inspecionado não pode ser reproduzido (Ince, Hatton, e Graham-Cumming 2012), e um código que não pode ser encontrado não pode ser creditado (Smith et al. 2016). Toda uma economia funciona hoje sobre código que alguém escolheu abrir: uma estimativa situa o custo de reconstruir o código aberto que as empresas usam em cerca de 8,8 trilhões de dólares americanos (Manuel Hoffmann, Nagle, e Zhou 2024).

Robert M. Pirsig escreveu o livro em que este se apoia. Em *Zen and the Art of Motorcycle Maintenance* (*Zen e a arte da manutenção de motocicletas*), ele resume o problema da ciência numa frase: "The number of rational hypotheses that can explain any given phenomenon is infinite" (em tradução livre: o número de hipóteses racionais que podem explicar qualquer fenômeno dado é infinito) (Pirsig 1974, 107).

[**ANÁLISE**] Se as hipóteses não têm fim, a ciência não tem fim, e nenhum laboratório sozinho consegue testá-las todas. Um método que só o dono pode abrir é um método que mais ninguém pode testar. Por isso sustento que a abertura vem primeiro para a ciência, antes de vir para os negócios.

Se qualquer pessoa pode fazer engenharia reversa de qualquer coisa, a pergunta útil deixa de ser *como escondo meu trabalho?* e passa a ser *como garanto que meu trabalho seja creditado quando for usado?* Minha resposta é a citação profissional: identificadores persistentes, metadados legíveis por máquina e, cada vez mais, agentes que citam por padrão. Essa resposta tem um lado prático, por isso descrevo também o pipeline aberto que produziu este artigo.

O artigo está organizado da seguinte forma. A Seção 2 revisita o Linux e o Git como evidência de que a abertura escala. A Seção 3 é uma confissão sobre o meu trabalho de conclusão de curso fechado. A Seção 4 examina a ciência e a espionagem em 2026 e a explosão da engenharia reversa assistida por IA. A Seção 5 reúne cinco linhas de evidência de que o código fechado está morto como forma de proteção. A Seção 6 trata da ofuscação, das cartas de John Nash sobre criptografia e de como proteger código-fonte em 2026. A Seção 7 defende a citação profissional e estabelece regras para agentes. A Seção 8 descreve o Paper Fabric. A Seção 9 lista meus trabalhos relacionados, e a Seção 10 expõe os limites do meu argumento.

Uma nota sobre o método. Este é um artigo de posição, escrito no estilo *research* do padrão de documentação Ouroboric: toda afirmação factual traz uma citação, e as passagens de interpretação vêm marcadas com [**ANÁLISE**]. É um preprint v0 e não passou por revisão por pares.

2 O bazar que deu certo: Linus Torvalds, o Linux e o Git

Em 25 de agosto de 1991, um estudante da Universidade de Helsinque publicou no grupo da Usenet `comp.os.minix` que estava "doing a (free) operating system (just a hobby, won't be big and professional like gnu)" (em tradução livre: fazendo um sistema operacional (livre) (só um hobby, não vai ser grande e profissional como o gnu)) para clones AT 386 e 486, e perguntou o que as pessoas gostariam de ver nele (Torvalds 1991). A versão 0.01 veio em 17 de setembro de 1991. No início de 1992, Torvalds passou o kernel para a GNU General Public License, e a versão 0.95, de março de 1992, foi o primeiro lançamento sob essa licença (Wikipedia contributors 2026b).

O hobby se tornou o kernel de sistema operacional mais amplamente implantado da história. Ele roda em servidores, telefones, roteadores, carros e nos supercomputadores dos quais depende boa parte da ciência computacional. Não preciso de um número de participação de mercado para defender o ponto; a semana do próprio leitor quase certamente passou por várias máquinas Linux.

A primeira conquista de Torvalds não foi apenas técnica. O relato de Eric Raymond sobre o desenvolvimento do Linux, *The Cathedral and the Bazaar*, identificou o método social por trás dele: lançar cedo, lançar com frequência, tratar os usuários como codesenvolvedores e deixar que muitos olhos encontrem os bugs que poucos encontrariam (Raymond 1998). Raymond contrapôs esse *bazar* à *catedral* do desenvolvimento cuidadosamente guardado e planejado de forma centralizada. O Linux mostrou que o bazar era capaz de construir o tipo mais difícil de software que existe.

A segunda conquista de Torvalds nasceu de um fracasso do código fechado. A partir de 2002, o kernel usou o BitKeeper, um sistema de controle de versão proprietário oferecido sem custo. Em abril de 2005, esse acesso gratuito acabou. Nenhum sistema livre atendia às necessidades do projeto, então Torvalds escreveu o seu próprio. Ele começou em 3 de abril de 2005; em 7 de abril, o Git já gerenciava o seu próprio código-fonte; o primeiro merge de múltiplos branches veio em 18 de abril; e em 26 de julho ele passou a manutenção a Junio Hamano (Wikipedia contributors 2026a). O primeiro commit do repositório do Git, ainda legível hoje, tem o título "Initial revision of"git", the information manager from hell" (em tradução livre: revisão inicial do git, o gerenciador de informação do inferno) (Torvalds 2005).

[**ANÁLISE**] O episódio do BitKeeper é uma pequena parábola para este artigo. A comunidade do kernel havia aceitado uma ferramenta fechada por conveniência. Quando o dono a retirou, a comunidade ficou exposta; quando Torvalds a substituiu por uma aberta, a substituta se tornou o substrato de quase todo o desenvolvimento de software, inclusive dos repositórios, pull requests e tags de lançamento por meio dos quais este artigo é construído e arquivado. O Git também encarna, em seu modelo de dados, a ética que defendo na Seção 7: todo commit nomeia o seu autor e aponta, por hash criptográfico, para a história exata sobre a qual se constrói. O Git é um grafo de citações para código.

Elogio Torvalds aqui sem reservas por esses dois presentes e por três décadas mantendo o primeiro. Observo também o que o elogio não exige: que o estilo técnico de uma pessoa seja copiado, ou que o código aberto seja livre de conflitos. A tese é mais estreita e mais forte. A abertura, na maior escala já tentada, deu certo.

3 Uma confissão de código fechado: Chemo-RX

Esta seção é escrita em primeira pessoa por Matheus Hoffmann. As passagens marcadas com [**RELATO DO AUTOR**] reservam lugar para as lembranças do próprio autor. Elas

devem ser escritas pelo autor antes que este artigo deixe a versão v0; nada nelas pode ser inventado por um agente.

Em 2018, comecei um trabalho de conclusão de curso de graduação na Escola Politécnica da Universidade Federal do Rio de Janeiro, em associação com o laboratório LESS do Programa de Engenharia Biomédica. Seu codinome era Chemo-RX. O objetivo era comparar métodos estatísticos para prever reações adversas à quimioterapia em mulheres tratadas de câncer de mama, usando dados do Instituto Nacional de Câncer (INCA). As reações consideradas eram astenia, mialgia, artralgia, mucosite, dor abdominal e diarreia, e os preditores incluíam comorbidades, variáveis demográficas e polimorfismos de nucleotídeo único (Matheus Hoffmann 2018).

O vestígio público desse trabalho é um repositório no GitHub com oito commits entre 23 de março e 25 de abril de 2018 (Matheus Hoffmann 2018). Ele contém um README, uma proposta em formato OpenDocument e um resumo de dados em R Markdown. O README tem três títulos que nunca foram preenchidos: "Proposal EN link", "Final project EN link" e "Thesis full content" (em tradução livre: link da proposta em inglês, link do projeto final em inglês e conteúdo completo da tese). Sob o último, diz apenas "contact me" (em tradução livre: entre em contato). A lista de pendências da proposta ainda inclui "Add INCA formal license/ethic regiment" (em tradução livre: adicionar a licença formal/regimento de ética do INCA).

[RELATO DO AUTOR: como o projeto começou, quem o orientou e o que a questão clínica significava para as pessoas envolvidas.]

[RELATO DO AUTOR: por que o texto e o código da tese ficaram fechados. Foi uma instrução do orientador, incerteza sobre o acordo de dados, trabalho inacabado ou simplesmente nunca ter chegado a fazê-lo?]

[RELATO DO AUTOR: o que se perdeu. Por exemplo, resultados que não puderam ser reutilizados, colaboradores que não puderam dar continuidade ao trabalho, ou a experiência de não conseguir mostrar o trabalho depois.]

[ANÁLISE] Dois tipos de fechamento se misturam naquele repositório, e a distinção importa para o restante deste artigo. Os dados das pacientes não podiam ser publicados, e não deveriam ter sido: prontuários clínicos são dados pessoais, e sua divulgação é regida por aprovação ética e por acordos de dados, o que a própria lista de pendências reconhece. O *método* poderia ter sido publicado: o código estatístico, o protocolo de comparação, as definições das variáveis e os resultados negativos. Os princípios FAIR fazem exatamente essa distinção. Dados e software devem ser localizáveis e acessíveis *sob condições bem definidas*, que podem incluir acesso restrito, e seus metadados devem permanecer públicos mesmo quando os dados não o são (Wilkinson et al. 2016; Barker et al. 2022). Fechar o método junto com os dados não protegeu nenhuma paciente. Apenas tornou o trabalho impossível de citar.

[RELATO DO AUTOR: o que o autor passou a fazer de diferente desde então, levando aos projetos de código aberto da Seção 9.]

4 Ciência e espionagem em 2026: a explosão da engenharia reversa

Uso *engenharia reversa* (REA, na abreviação do meu projeto, de *reverse engineering, assisted*) para qualquer processo que recupere o funcionamento de um artefato a partir do próprio artefato, sem acesso aos seus documentos de projeto: descompilar um binário, inferir um

modelo a partir de suas saídas ou reconstruir um produto a partir de suas peças. Esta seção revisa quatro acontecimentos. O primeiro é um processo judicial, e seus fatos são alegações; eu os relato como tais.

4.1 Apple v. OpenAI: segredos saem andando sobre duas pernas

Em 10 de julho de 2026, a Apple processou a OpenAI, a empresa de dispositivos io, que a OpenAI havia adquirido, e dois ex-funcionários da Apple no Tribunal Distrital dos Estados Unidos para o Distrito Norte da Califórnia (San Jose). As alegações são de apropriação indevida de segredo comercial, com base no Defend Trade Secrets Act de 2016, e, contra as duas pessoas físicas, de quebra de contrato (Bridge Counsels 2026; Perez 2026a). Segundo a petição inicial, tal como noticiada, um ex-engenheiro da Apple acessou sistemas da Apple por meio de uma falha de autenticação a partir do computador corporativo da Apple de um colega; o diretor de hardware da OpenAI, ex-executivo da Apple, pediu a candidatos a vagas que levassem peças, arquivos de projeto e protótipos da Apple às entrevistas; e novos contratados receberam orientações sobre como evitar o processo de desligamento da Apple. A Apple alega ainda que mais de 400 ex-funcionários da Apple trabalham hoje na OpenAI (Perez 2026a). A OpenAI declarou publicamente que não tem "no interest in other companies' trade secrets" (em tradução livre: nenhum interesse nos segredos comerciais de outras empresas) (Perez 2026a). Em agosto, pediu a extinção da ação, argumentando que a Apple permitia que funcionários usassem contas pessoais de nuvem para o trabalho, não revogou acessos de forma adequada e não havia identificado quais segredos teriam sido levados (Perez 2026b). No momento em que escrevo, o pedido de extinção e um pedido de liminar estavam pendentes, e nenhum tribunal havia decidido o mérito (Bridge Counsels 2026; Perez 2026b).

[**ANÁLISE**] Qualquer que seja o desfecho, a disputa é instrutiva por aquilo que ela *não* trata. Ela não trata de código-fonte que um adversário deixou de ler por ser secreto. Todo caminho alegado passa por pessoas: suas contas, seus dispositivos, suas conversas de recrutamento e suas memórias. O sigilo, neste relato, era uma propriedade dos processos de uma organização, e processos vazam. Note-se também que a defesa da OpenAI, tal como noticiada, sustenta em parte que as próprias práticas da Apple não tratavam a informação como secreta. Um segredo que precisa ser ativamente guardado em cada entrevista de desligamento é caro; o custo dessa guarda faz parte do preço do código fechado.

Nota de verificação. Não consegui consultar os autos diretamente. Fontes secundárias dão o número do processo como 5:26-cv-07078, com sufixos de juiz conflitantes (VKD e EJD), e o intitulam tanto *Apple Inc. v. OpenAI* quanto *Apple Inc. v. Liu* (Bridge Counsels 2026). O número precisa ser conferido no PACER antes que este artigo deixe a v0.

4.2 Destilação: copiando capacidade pela porta da frente

A destilação de conhecimento, isto é, treinar um modelo menor para imitar as saídas de um maior, é uma técnica padrão e legítima (Hinton, Vinyals, e Dean 2015). Em 2026, ela também se tornou o centro de uma acusação de cópia industrial. Em fevereiro, segundo a Reuters, a OpenAI informou ao House Select Committee on Strategic Competition between the United States and the Chinese Communist Party, da Câmara dos Representantes dos EUA, que contas associadas a funcionários da DeepSeek haviam desenvolvido métodos para contornar as restrições de acesso da OpenAI e "obtain outputs for distillation in programmatic ways" (em tradução livre: obter saídas para destilação de forma programática) (Reuters 2026). A DeepSeek não respondeu à Reuters. Dez dias depois, a Anthropic relatou "industrial-scale campaigns by three AI laboratories" (em tradução livre: campanhas em escala industrial de

três laboratórios de IA), DeepSeek, Moonshot AI e MiniMax, que usaram cerca de 24.000 contas fraudulentas e mais de 16 milhões de interações com seus modelos Claude (Anthropic 2026).

[**ANÁLISE**] Trata-se de acusações feitas por partes interessadas, e não me cabe julgá-las. Sua relevância aqui é estrutural. Um modelo servido por trás de uma API é código fechado no sentido mais forte: seus pesos nunca saem dos servidores do provedor. Ainda assim, cada resposta que ele dá é uma amostra do seu comportamento, e amostras suficientes ensinam a um modelo aluno boa parte do que o professor sabe. A interface que vende o modelo é a interface que o vaza.

4.3 Extração: lendo o modelo através de suas respostas

O trabalho acadêmico demonstrou o mesmo ponto com mais precisão e menos dramaticidade. Em 2016, Tramèr e colegas mostraram que modelos de aprendizado de máquina servidos por APIs de predição podem ser roubados por meio de consultas (Tramèr et al. 2016). Em 2024, Carlini e colegas foram além, contra modelos de linguagem em produção: seu ataque recuperou a matriz inteira de projeção de *embeddings* dos modelos Ada e Babbage da OpenAI, por menos de vinte dólares americanos em consultas, e confirmou pela primeira vez suas dimensões ocultas (Carlini et al. 2024). O artigo, publicado na ICML 2024, termina com defesas e mitigações: a resposta de uma comunidade de pesquisa a um vazamento descoberto é publicá-lo, para que possa ser corrigido.

4.4 Descompilação: o fim do binário opaco

A engenharia reversa de software compilado é há muito tempo rotineira e, para fins como a interoperabilidade, lícita (Samuelson e Scotchmer 2002). Em 2019, a National Security Agency dos EUA liberou seu próprio framework de engenharia reversa, o Ghidra, como código aberto sob a licença Apache-2.0 (National Security Agency 2019). Os modelos de linguagem então reduziram a barreira que restava, que era a habilidade. O LLM4Decompile, uma família de modelos abertos treinados para descompilação, produz código-fonte que recompila e passa nos testes originais; seus modelos de ponta a ponta mais que dobram a taxa de reexecutabilidade do GPT-4o e do Ghidra nos benchmarks HumanEval e ExeBench (Tan et al. 2024).

[**ANÁLISE**] Juntando esses quatro acontecimentos, a curva de custo da engenharia reversa desabou em todos os eixos: para binários (descompilação), para modelos (extração e destilação) e para organizações (pessoas que mudam de empregador). Chamo isso de *explosão da REA*. Ela não torna todo artefato transparente da noite para o dia, nem torna lícita a cópia onde a lei a proíbe. Ela torna, sim, o sigilo uma defesa fraca e cara, e torna a questão do crédito, quem fez o quê primeiro e quem construiu sobre o trabalho de quem, mais importante do que a questão do acesso.

5 Por que o código fechado está morto: cinco linhas de evidência

"Dead" (morto) é uma palavra forte, e devo dizer o que quero dizer com ela. Software de código fechado continuará sendo vendido por muito tempo. O que está morto é o código fechado *como proteção*: a ideia de que manter um artefato ilegível mantém seu valor seguro, e de que essa proteção vale o que custa à ciência. Cinco referências, da criptografia, do

direito, do aprendizado de máquina, da engenharia de software e da economia, sustentam essa afirmação. Escolhi-as pela credibilidade: cada uma é revisada por pares, uma fonte jurídica ou histórica primária, ou um *working paper* amplamente citado de uma universidade de pesquisa.

Table 1: Cinco referências sobre por que o sigilo falha como proteção.

#	Referência	Área	O que mostra
1	Kerckhoffs (1883)	criptografia	Um sistema deve continuar seguro quando o inimigo o tem; só a chave pode ser secreta.
2	Samuelson e Scotchmer (2002)	direito e economia	A engenharia reversa é uma prática antiga, lícita e economicamente útil.
3	Carlini et al. (2024)	aprendizado de máquina	Partes de um modelo em produção por trás de uma API podem ser extraídas por menos de 20 dólares americanos.
4	Tan et al. (2024)	engenharia de software	Modelos de linguagem descompilam binários em código-fonte que recompila e passa nos testes.
5	Ince, Hatton, e Graham-Cumming (2012)	política científica	Resultados calculados por código não publicado não podem ser devidamente verificados nem reproduzidos.

1. A segurança por obscuridade foi refutada em 1883. O segundo princípio de Auguste Kerckhoffs para cifras militares diz que o sistema "must not require secrecy and can be stolen by the enemy without causing trouble" (em tradução livre: não deve exigir sigilo e pode ser roubado pelo inimigo sem causar problemas) (Kerckhoffs 1883). A criptografia moderna se apoia nesse princípio: os algoritmos são públicos, e só as chaves são secretas. Um projeto cuja segurança depende de o próprio projeto permanecer oculto é considerado quebrado de antemão.

2. O direito aceita há muito tempo a engenharia reversa. Samuelson e Scotchmer abrem seu levantamento com a observação de que a engenharia reversa "has a long history as an accepted practice" (em tradução livre: tem uma longa história como prática aceita) (Samuelson e Scotchmer 2002). O direito de segredo comercial protege contra meios ilícitos, como furto e quebra de confidencialidade, e não contra alguém que estuda um produto que obteve licitamente. O código fechado, portanto, nunca teve uma garantia jurídica contra ser compreendido. Tinha uma garantia prática: a engenharia reversa era lenta e cara.

3. Essa garantia prática acabou para os modelos. Os ataques de extração a APIs de predição (Tramèr et al. 2016) amadureceram em ataques a modelos de linguagem em produção (Carlini et al. 2024), e a destilação em escala é agora alegada pelos próprios provedores (Anthropic 2026; Reuters 2026).

4. Ela está acabando para os binários. Modelos abertos de descompilação, construídos sobre um framework aberto de engenharia reversa liberado por uma agência de inteligência de sinais (National Security Agency 2019), recuperam código-fonte executável a partir de código compilado (Tan et al. 2024).

5. Enquanto isso, o sigilo custa caro à ciência. Ince e colegas argumentaram na *Nature* que, com poucas exceções, "anything less than the release of source programs is intolerable for results that depend on computation" (em tradução livre: qualquer coisa aquém da liberação dos programas-fonte é intolerável para resultados que dependem de computação) (Ince, Hatton, e Graham-Cumming 2012). O peso econômico da alternativa é hoje mensurável: o código aberto que as empresas usam custaria cerca de 4,15 bilhões de dólares americanos

para ser escrito uma vez, e cerca de 8,8 trilhões de dólares americanos se cada empresa tivesse de substituí-lo por conta própria (Manuel Hoffmann, Nagle, e Zhou 2024).

[ANÁLISE] As linhas 1 a 4 dizem que o sigilo já não protege; a linha 5 diz que ele nunca saiu de graça. Juntas, deixam o pesquisador diante de uma escolha que já não é entre *oculto* e *exposto*. É entre *exposto e sem crédito* e *exposto e com crédito*. A próxima seção trata de escolher a segunda.

6 Ofuscação, as cartas de Nash e como proteger código-fonte em 2026

Se o sigilo falha como proteção, o que resta para o código que precisa permanecer privado, e para o código público do qual outros dependem? Esta seção examina uma resposta tentadora, a ofuscação, e a resposta que a criptografia oferece em seu lugar.

6.1 As cartas de Nash: segurança pela computação, não pela ocultação

No início de 1955, John Nash, autor do conceito de equilíbrio para jogos não cooperativos (Nash 1950), escreveu à National Security Agency dos EUA para propor uma máquina de cifragem (Nash e National Security Agency 1955). A NSA tornou pública a correspondência em 2012 (Branwen 2012). Nela, Nash argumentava que, para cifras suficientemente complexas, "the mean key computation length increases exponentially with the length of the key" (em tradução livre: o comprimento médio da computação da chave cresce exponencialmente com o comprimento da chave) (Branwen 2012). A ideia é que a segurança deve repousar sobre o custo da computação, e não sobre a ocultação do projeto. Rivest, Shamir e seus alunos leram as cartas como uma antecipação da teoria da complexidade e da criptografia moderna (Rivest, Shamir, et al. 2012). A máquina do próprio Nash não teve a mesma sorte. A NSA julgou que ela "affords only limited security" (em tradução livre: oferece apenas segurança limitada) (Nash e National Security Agency 1955), e, quando Ronald Rivest, Adi Shamir e seus alunos a analisaram em 2012, encontraram um ataque de texto claro escolhido em tempo polinomial e um ataque de texto claro conhecido em tempo subexponencial, e concluíram que Nash era "a much better game theorist than a cryptographer" (em tradução livre: um teórico dos jogos muito melhor do que criptógrafo) (Rivest, Shamir, et al. 2012).

[ANÁLISE] As duas metades da história apoiam Kerckhoffs. O princípio de Nash, segurança a partir do trabalho que um atacante precisa realizar, é o que a criptografia moderna conservou. Seu projeto, uma vez publicado, foi avaliado e quebrado por outros. É assim que a criptografia deve funcionar: o método é aberto, e somente a chave é secreta.

6.2 Ofuscação: o que ela oferece e o que não pode oferecer

A ofuscação transforma um programa de modo que ele faça a mesma coisa, mas seja mais difícil de ler. Um dos Eu construí um exemplo pequeno e honesto. *nashkell*, um pipeline em Haskell que executa OCR sobre as cartas desclassificadas de Nash e pede a um modelo de linguagem que as anote e as confronte entre si, armazena seu código-fonte no Git com os identificadores traduzidos para o latim (Matheus Hoffmann 2026a). Um filtro `clean` do Git traduz identificadores do inglês para o latim no commit, e um filtro `smudge` os traduz de volta no checkout, por meio de um dicionário fixo; quem clona sem o filtro vê, e o GitHub mostra, a forma em latim.

A própria documentação do projeto lista os limites, e eles são gerais (Matheus Hoffmann 2026a):

- **A estrutura sobrevive.** Somente os nomes de nível superior mudam. Fluxo de controle, tipos e variáveis locais permanecem intactos e dedutíveis, de modo que o resultado "raises, not eliminates, reading difficulty" (em tradução livre: aumenta, mas não elimina, a dificuldade de leitura) e "not a security boundary" (em tradução livre: não é uma fronteira de segurança).
- **A chave é o dicionário.** O mapeamento é uma bijeção armazenada ao lado da ferramenta. Quem o possui, ou consegue adivinhar o suficiente dele, desfaz a transformação.
- **A forma armazenada não compila.** Os caminhos de arquivo não são renomeados, de modo que os módulos em latim deixam de corresponder a seus caminhos. A ofuscação também tem um custo para o dono.
- **É uma heurística.** O filtro opera linha a linha, não é um parser; não trata comentários de bloco nem strings de múltiplas linhas, e nomes novos precisam ser acrescentados ao dicionário à mão.

A teoria diz que esses limites não são uma questão de esforço. Barak e colegas provaram que não pode existir um ofuscador geral de "virtual black box" (caixa-preta virtual). Tal ofuscador não permitiria a ninguém computar, a partir do programa ofuscado, nada que não pudesse ser computado apenas com acesso de entrada e saída. Os autores construíram uma família de programas que podem ser obtidos por engenharia reversa a partir de qualquer programa equivalente, embora o acesso de caixa-preta não revele nada de seu código (Barak et al. 2012). O resultado deixa espaço para noções mais fracas de ofuscação, mas não para um método geral que torne o código-fonte secreto. **[ANÁLISE]** E, diante dos modelos de linguagem da Seção 4, que já recuperam código-fonte a partir de binários, renomear identificadores para o latim é apenas um quebra-molas.

6.3 O que proteger em vez disso: integridade, identidade e chaves

[ANÁLISE] A lição de Nash e da ofuscação é parar de proteger a *legibilidade* do código e proteger aquilo que a criptografia é capaz de proteger: que o código é o que seus autores escreveram, e que somente eles poderiam tê-lo escrito. Em 2026, isso significa quatro camadas, cada uma com ferramentas públicas e maduras.

1. **Assine commits e tags.** O Git assina tags e commits com GnuPG, e `git merge --verify-signatures` recusa um histórico sem assinaturas confiáveis (Chacon e Straub 2014). Desde o Git 2.34, chaves SSH podem assinar, assim como chaves GnuPG (The Git project 2021).
2. **Mantenha as chaves de assinatura em hardware.** Desde o OpenSSH 8.2, chaves de segurança FIDO dão suporte aos tipos de chave `ecdsa-sk` e `ed25519-sk`; a chave privada nunca sai do token, e o token precisa estar presente quando a chave é usada (OpenSSH project 2020). Um laptop roubado deixa de significar uma chave de assinatura roubada.
3. **Exija autenticação de dois fatores na forja.** Em 2022, o GitHub se comprometeu a exigir que todo usuário que contribui com código no GitHub.com ativasse a autenticação de dois fatores até o fim de 2023 (Hanley 2022). Uma pessoa que se apodera de uma conta pode enviar código em nome de seu dono.
4. **Torne as assinaturas verificáveis por todos.** O Sigstore permite que desenvolvedores assinem com chaves efêmeras vinculadas a uma identidade existente por meio do OpenID Connect, e registra as assinaturas em logs de transparência, o que elimina a

maior parte do ônus e do risco da gestão de chaves (Newman, Meyers, e Torres-Arias 2022).

As funções de hash também importam. Em 2017, Stevens e colegas produziram a primeira colisão para o SHA-1 completo (Stevens et al. 2017), o hash sobre o qual os nomes de objetos do Git foram construídos. O Git oferece um formato de objetos SHA-256 desde a versão 2.29, inicialmente marcado como experimental (The Git project 2020).

O que permanece secreto sob esse modelo é pouco e explícito: chaves privadas, credenciais e dados pessoais ou sujeitos a acordo, como na Seção 3. Esses merecem criptografia e controle de acesso. O próprio código-fonte é mais bem servido sendo publicado, assinado e citado.

[**ANÁLISE**] No vocabulário do próprio Nash, trata-se de uma mudança de estratégia em um jogo cujos payoffs se deslocaram. Quando a engenharia reversa era cara, o sigilo podia ser uma estratégia sensata para um autor. Quando a engenharia reversa é barata, o payoff do sigilo cai para perto de zero, enquanto o payoff da publicação creditada e verificável não cai. Não afirmo ter calculado um equilíbrio (Nash 1950); afirmo apenas que os custos sobre os quais a velha estratégia se apoiava desapareceram.

7 A resposta ética: citação profissional

Se qualquer pessoa pode ler o seu trabalho, a proteção que resta é a norma de que quem o usa diz que o usou. Essa norma é antiga na ciência; é jovem no software, e mal está formada para agentes.

7.1 Citação para pessoas

Os *Software citation principles* da FORCE11 estabelecem o que significa citar software adequadamente: software é um produto legítimo de pesquisa e deve ser citado como tal; as citações devem dar crédito, ser únicas e persistentes, ser acessíveis e ser específicas à versão usada (Smith et al. 2016). Os princípios FAIR pedem que os produtos de pesquisa sejam localizáveis, acessíveis, interoperáveis e reutilizáveis (Wilkinson et al. 2016), e o FAIR4RS os adapta ao software de pesquisa (Barker et al. 2022).

Na prática, isso exige três coisas em todo repositório. A primeira é um identificador persistente: um DOI emitido por um arquivo como o Zenodo, com um DOI de *conceito* que sempre aponta para a versão mais recente e um DOI de *versão* para cada release. A segunda são metadados legíveis por máquina, para que uma citação possa ser gerada em vez de digitada: `CITATION.cff` para o "Cite this repository" do GitHub, e `.zenodo.json` para o arquivo. A terceira é um processo de release em que cada versão publicada recebe seu próprio identificador, para que o leitor possa citar exatamente o que usou.

Este repositório segue essa receita; a Seção 8 dá os detalhes. O mesmo vale para os trabalhos relacionados da Seção 9, cada um dos quais tem um DOI de conceito.

7.2 Citação para agentes: tomar emprestado, citar, propagar

Agentes de programação e de pesquisa escrevem hoje uma grande parte do código e do texto novos, inclusive a redação deste artigo (ver a declaração de uso de IA). Agentes leem muito mais do que qualquer pessoa e copiam com fluência. Se não citam, a explosão de REA se torna uma máquina de lavagem: código e ideias entram com autores e saem sem eles. Por

isso, proponho três regras para agentes, que o `AGENTS.md` do Paper Fabric torna obrigatórias para qualquer agente que trabalhe em seus repositórios.

1. **Tome emprestado apenas o que a licença permite.** Antes de reutilizar código, texto, figuras ou dados, leia a licença e registre-a. Reutilização que a licença não permite não é empréstimo. Na dúvida, descreva a ideia e cite-a em vez de copiar sua expressão.
2. **Cite no ponto de uso, com um identificador persistente.** Todo elemento tomado emprestado recebe uma citação onde é usado, e não apenas em uma lista final. Prefira um DOI; na falta dele, use uma URL com o hash de um commit ou um snapshot de arquivo. Nunca invente uma referência: um agente que não consegue obter e verificar uma fonte deve marcar a afirmação como não verificada.
3. **Propague o crédito adiante.** Quando a produção de um agente é ela mesma publicada, as citações viajam com ela: na bibliografia, nas referências do `CITATION.cff` e nos identificadores relacionados do registro no Zenodo (`cites`, `isDerivedFrom`, `isSupplementTo`). O crédito que para no primeiro salto se perde no segundo.

[**ANÁLISE**] As regras não pedem nada exótico. São o que faz um pós-graduando cuidadoso. A diferença está na escala e na verificabilidade: uma pessoa que esquece uma citação esquece uma; um agente que esquece esquece milhares, e um agente que inventa referências envenena o registro. A proibição de referências inventadas da Regra 2 é, portanto, aplicada mecanicamente no Paper Fabric, por meio de um log de referências que registra como cada entrada foi verificada.

8 Paper Fabric: uma fábrica ourobórica de artigos

Paper Fabric é o pipeline aberto que produziu este preprint. Ele foi projetado para ser reutilizado em qualquer artigo e segue a filosofia do Ouroboric, o projeto de Bruno Valério, cuja carta descreve "open, self-improving factories" (em tradução livre: fábricas abertas e autoaperfeiçoáveis) que produzem trabalho, medem-no e realimentam a si mesmas com o que aprendem (Ouroboric DEV 2026). A carta do Ouroboric lista uma fábrica de pesquisa "for scientific research and papers" (em tradução livre: para pesquisa científica e artigos) como uma possível direção futura, explicitamente não um compromisso (Ouroboric DEV 2026). Paper Fabric é um experimento independente nessa direção.

8.1 Princípios

Paper Fabric adota diretamente quatro dos princípios do Ouroboric; as ideias são de Valério. *Humanos conduzem, agentes constroem*: eu aprovo a direção e cada afirmação que se apoia na minha própria experiência; os agentes redigem, verificam e constroem. *Verificações, não promessas*: as citações são conferidas com a fonte, e o build falha diante de uma citação ausente. *Leia antes de tomar emprestado*: as licenças decidem o que pode ser reutilizado, e dá-se crédito pelo que é tomado. *Aprenda em público*: o pipeline, os prompts e a configuração de agentes que produziram um artigo são publicados junto com ele.

8.2 Pipeline

O conjunto de ferramentas segue os livros de Sandy Maguire (Maguire 2019): a prosa é escrita em Markdown, um arquivo por seção, e um `justfile` executa o pandoc com um

template LaTeX personalizado e uma bibliografia BibTeX, depois o XeLaTeX, e carimba o PDF com a revisão do Git a partir da qual foi gerado. Um flake Nix fixa cada ferramenta em uma única versão, por meio do lab `paper` do picasso (Matheus Hoffmann 2026b). O citeproc embutido do pandoc resolve as citações, e o build roda com avisos tratados como erros, de modo que uma chave de citação não resolvida interrompe o build.

Cada artigo fica em seu próprio diretório, `papers/<slug>/`, com um `metadata.md`, arquivos de prosa numerados e um `refs.bib`. Os metadados de citação do repositório (`.zenodo.json`, a partir do qual o `CITATION.cff` é gerado) são tomados emprestados, com crédito, dos meus projetos anteriores (Matheus Hoffmann 2026b; Matheus Hoffmann et al. 2026). Um release é uma tag do Git. Para a v0, o workflow de release gera o PDF, anexa-o a um release do GitHub e, somente quando um mantenedor fornece um token do Zenodo, envia *apenas o PDF* para um depósito em rascunho no Zenodo. Publicar o depósito, o que emite um DOI permanente, continua sendo uma decisão humana.

8.3 O ciclo

[**ANÁLISE**] O que torna o pipeline ourobórico, e não meramente automatizado, é que cada artigo alimenta o seguinte. O log de referências, a avaliação de prontidão e a configuração de agentes exportada são mantidos no repositório. O artigo seguinte parte deles, e o diário registra o que deu errado. Os defeitos conhecidos deste próprio artigo estão listados na avaliação de prontidão do repositório, que faz parte do mesmo pull request que o introduziu.

9 Trabalhos relacionados do mesmo autor

O argumento deste artigo é também uma descrição de como trabalho desde a tese fechada da Seção 3. Cada projeto abaixo é de código aberto, está arquivado no Zenodo com um DOI de conceito e traz metadados de citação legíveis por máquina.

- **WW3 GPU Lab** (Matheus Hoffmann 2026c): um livro em andamento sobre programação agêntica e pesquisa agêntica, construído em torno da portabilidade de kernels do modelo espectral de ondas WAVEWATCH III para GPUs com agentes de programação, sob um arcabouço de paridade bit a bit com a referência em Fortran.
- **marola** (Matheus Hoffmann et al. 2026): uma plataforma aberta e sem fins lucrativos sobre condições do mar e qualidade da água para banho nas praias brasileiras, construída sobre dados públicos. Estou entre os seus criadores.
- **picasso** (Matheus Hoffmann 2026b): flakes Nix autocontidos que fixam as cadeias de ferramentas usadas pelos projetos acima, incluindo um pipeline de publicação com pandoc e TeX Live. O picasso é também um caso de uso de um sistema que se aprimora a si mesmo: quando um projeto precisa de uma ferramenta, a necessidade volta ao picasso como um novo lab ou uma nova ferramenta, e todo projeto que usa o picasso a recebe na sua próxima atualização. Os labs `paper` e `gleam-website`, que constroem este artigo e o seu site, chegaram ao picasso dessa forma.
- **nashkell** (Matheus Hoffmann 2026a): um pipeline em Haskell que lê as cartas desclassificadas entre Nash e a NSA com OCR e um modelo de linguagem, e armazena seus identificadores em latim por meio de um filtro do Git; a Seção 6 o discute como exemplo de ofuscação e de seus limites.

Uma influência não é obra minha. As ideias ouroboric deste artigo são de Bruno Valério: Ouroboric (Ouroboric DEV 2026), fábricas abertas que se aprimoram a si mesmas, para

software e para empresas. Seu padrão de documentação, seus estilos de escrita e sua lista de verificação de prontidão moldaram o Paper Fabric.

[RELATO DO AUTOR: por que o autor trabalha abertamente há anos, nas próprias palavras do autor. Por exemplo, o que mudou depois do Chemo-RX e o que o trabalho aberto devolveu ao autor.]

Para trabalhos anteriores de outros autores sobre os temas deste artigo, ver Raymond sobre o método social do código aberto (Raymond 1998), Samuelson e Scotchmer sobre o direito e a economia da engenharia reversa (Samuelson e Scotchmer 2002), Smith, Katz e Niemeyer sobre citação de software (Smith et al. 2016) e Barker e colegas sobre software de pesquisa FAIR (Barker et al. 2022).

10 Limitações

Declaro os limites do meu argumento com franqueza, em consonância com o valor Ouroboric da honestidade quanto aos resultados.

- **Posição, não medição.** Este artigo argumenta a partir de evidências citadas; não relata nenhum experimento novo. Suas afirmações sobre o custo da engenharia reversa apoiam-se nos estudos citados, não em medições minhas.
- **Alegações, não conclusões.** A disputa Apple v. OpenAI e as acusações de destilação são afirmações de partes interessadas. Nenhum tribunal ou órgão independente havia se pronunciado sobre elas no momento da redação.
- **Fontes secundárias para os eventos de 2026.** Baseei-me em reportagens e em um resumo de um escritório de advocacia para os eventos de 2026. O registro processual não foi obtido diretamente, e as fontes secundárias divergem quanto ao sufixo do juiz no número do processo.
- **Nem todo fechamento é errado.** Dados de pacientes, dados pessoais, detalhes sensíveis do ponto de vista da segurança e material sob embargo têm boas razões para permanecer fechados. Meu argumento diz respeito a métodos, código e modelos cujo sigilo protege apenas o seu dono.
- **Interesse do autor.** Construo ferramentas de código aberto e cito meus próprios projetos. Os leitores devem ponderar a Seção 9 levando isso em conta.
- **Ainda sem revisão por pares.** Este é um preprint v0.

11 Conclusão

O código fechado pedia aos pesquisadores que confiassem que o sigilo protegeria o seu trabalho. Entre 2024 e 2026, essa confiança falhou publicamente: binários descompilados por modelos de linguagem, modelos em produção lidos por meio das suas próprias respostas, capacidades copiadas pela porta da frente e a maior disputa de segredo industrial da área travada em torno de laptops e entrevistas de emprego. Linux e Git mostraram há muito tempo que a alternativa funciona em escala.

O que resta é uma questão de boas maneiras, tornada urgente pelas máquinas. Se qualquer pessoa pode fazer engenharia reversa de qualquer coisa, o caminho honesto é publicar o método, emitir um identificador e citar tudo o que se usa, e exigir o mesmo dos agentes que agora fazem boa parte da leitura e da escrita. O Paper Fabric é a minha tentativa de tornar esse caminho o padrão para o meu próprio trabalho. Seu código-fonte, sua configuração de agentes e seus defeitos conhecidos são publicados junto com este artigo.

Declarações

Autoria. A capa nomeia Mr. Nobody. Eu, Matheus Hoffmann, escrevi este artigo na primeira pessoa, e os metadados de citação me nomeiam.

Agradecimentos. As ideias ouroboric deste artigo são de Bruno Valério (Ouroboric DEV 2026).

Uso de IA. A estrutura, os rascunhos do texto, as verificações de referências, as ferramentas de build e a configuração do repositório desta v0 foram produzidos por um agente de programação Claude (Anthropic) trabalhando em um projeto Claude, sob a minha direção. Cada referência foi obtida e verificada pelo agente no momento da redação; as verificações estão registradas no log de referências do repositório. A configuração do agente está publicada no diretório `.claude/` do repositório. Sou responsável pelo conteúdo.

Conflitos de interesse. Mantenho os projetos de código aberto citados na Seção 9. A Anthropic, cujo relatório é citado na Seção 4, também produz o modelo que redigiu este artigo. Não declaro nenhum outro conflito de interesse.

Disponibilidade. Código-fonte, pipeline de build e configuração do agente: https://github.com/h0ffmann/closed-source. Licença: texto CC BY 4.0; scripts de build AGPL-3.0-only; template LaTeX MIT.

Referências

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, e 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". Em *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, e 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, e Jeff Dean. 2015. "Distilling the Knowledge in a Neural Network". https://arxiv.org/abs/1503.02531.

Hoffmann, Manuel, Frank Nagle, e 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/zeno

do.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, e 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., e 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, e Santiago Torres-Arias. 2022. “Sigstore: Software Signing for Everybody”. Em *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, e Suzanne Scotchmer. 2002. “The Law and Economics of Reverse Engineering”. *Yale Law Journal* 111: 1575. https://www.yalelawjournal.org/article/the-law-and-economics-of-reverse-engineering.

Smith, Arfon M., Daniel S. Katz, Kyle E. Niemeyer, e 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, e Yarik Markov. 2017. “The First Collision for Full SHA-1”. Em *Advances in Cryptology – CRYPTO 2017*. Vol. 10401. Lecture Notes em Computer Science. Springer. https://doi.org/10.1007/978-3-319-63688-7_19.

Tan, Hanzhuo, Qi Luo, Jing Li, e Yuqun Zhang. 2024. “LLM4Decompile: Decompiling Binary Code with Large Language Models”. Em *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, e Thomas Ristenpart. 2016. “Stealing Machine Learning Models via Prediction APIs”. Em *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.