法律技术专栏｜大模型已经可以“破解”软件源码了吗？（下篇）——AI把代码“改头换面”后，软件侵权还怎么证明?

原创 吴圣健 Law Review 2026年8月10日 15:26 美国

作者：吴圣健｜ 新加坡法律出版社特邀技术顾问

大模型反编译背景下的软件著作权、商业秘密与侵权举证

上篇回顾｜上篇我们讨论了大模型反编译最近两年的快速进展：AI 目前通常还不能精确找回开发者当初写下的原始源码，但已经可以在部分场景中理解二进制程序的逻辑，并生成能够重新编译、通过运行测试的替代代码。真正发生变化的，是理解和重建软件的门槛正在下降。

技术问题到这里，一个更现实的法律问题随之出现：如果 AI 把代码重新写了一遍，甚至表面上已经“不像”原代码，我们还怎么判断它是不是从原软件而来?

一个可能越来越常见的争议场景

A 公司开发了一套本地运行的工业软件，只向客户交付可执行文件。B 公司取得该软件后，使用传统反编译工具和大模型分析程序，并重新生成了一套功能相近的代码。

新代码的变量名、函数划分和代码风格都与 A 公司不同，但部分特殊参数、异常行为和处理流程高度一致。A 公司认为其软件被复制，B 公司则主张，其只是理解了软件功能，并由 AI 重新实现。

在这种情况下，问题已经不只是“两份源代码有多像”，还包括：大模型究竟接触了什么材料、恢复了什么内容、哪些相似具有功能必然性、哪些异常一致可能反映共同来源，以及双方能否提供完整的开发过程证据。

导语｜AI 可以在反编译结果基础上重新命名变量、调整函数结构并重写代码，使新代码在文字层面与原程序明显不同，但这并不当然意味着独立开发。本文不再重点讨论 AI“能不能反编译”，而是进一步关注：当理解和重建软件的门槛下降后，软件著作权、商业秘密和侵权举证将面临哪些新的问题？又应如何通过接触路径、数据指纹、结构指纹、行为指纹和开发过程判断代码来源？

一、从“能不能恢复代码”，到“恢复之后怎么用”

1、从“恢复源码”到“重建可运行代码”

如果所谓“破解源码”，是指完整找回开发者当初使用的变量名、注释、文件结构和原始工程，目前的答案通常是否定的：编译过程中已经丢失的信息，大模型无法仅凭二进制可靠地重新创造出来。

但如果所谓“破解”，是指理解程序逻辑、重建主要数据结构，并生成能够重新编译、在一定测试范围内准确运行的替代代码，那么答案已经不能简单说“不可能”。

一个较容易理解的比喻是：AI 根据已经建成的房子，重新画出一套可以施工的图纸，而不是从房子里找回建筑师当年的原始草稿。

2、反编译行为和反编译结果的后续使用，应当分开看

技术上能够分析一个程序，并不直接回答具体行为是否合法。实际评价通常需要结合软件如何取得、是否存在许可或保密义务、是否规避技术保护措施、分析目的是什么，以及恢复结果随后被如何使用。

例如，为兼容性、安全研究或维护旧系统进行分析，与把恢复结果直接用于竞争产品，并不是同一种场景。真正进入纠纷时，焦点往往会从“能不能反编译”转向“取得了什么、复制了什么、使用了什么”。

3、软件著作权不会因为代码更容易恢复而失效

我国《计算机软件保护条例》明确规定，同一计算机程序的源程序和目标程序为同一作品。[7] 因此，软件以二进制形式交付，并不意味着它脱离了著作权保护；能够通过工具恢复代码，也不等于取得了自由复制、发行或用于商业产品的权利。

让 LLM 更换变量名、调整函数结构和统一代码风格，也不会当然把复制内容变成独立创作。另一方面，软件著作权保护的是具体代码表达，而不是对抽象功能、处理方法或者数学思想的

垄断。

于是，AI 反编译最容易制造的争议正是：被诉方究竟复制了原软件的具体表达，还是理解了其功能后真正独立写出了另一套实现?

4、“AI 生成”不会自动切断代码来源

将反编译代码交给大模型，再要求它“改写成不同版本”，并不会自动完成法律上的“洗白”。代码文字虽然可以迅速变化，但输入材料、特殊结构、异常处理和开发过程仍可能暴露其来源。

在具体争议中，值得关注的可能包括：模型是否直接接触过目标程序或伪代码；生成结果是否保留了非必要的特殊常量和处理顺序；是否出现相同错误和异常行为；使用者能否提供连续的需求、设计、提交和测试记录。

5、商业秘密可能面对更直接的挑战

企业真正希望保护的，往往不只是几行代码，而是代码背后的风控规则、定价策略、授权逻辑、通信协议、设备控制算法、模型推理流程和关键参数。只要这些内容必须在客户设备上本地执行，就会以某种形式存在于可取得的程序中。

新修订的《反不正当竞争法》继续将不为公众所知悉、具有商业价值并采取相应保密措施的技术信息和经营信息纳入商业秘密保护，并规制以电子侵入等不正当手段获取、使用商业秘密的行为。[9]

LLM 不会自动改变某项行为的合法性，却可能缩短“别人暂时看不懂”带来的事实保密寿命。对完整下发客户端、主要依靠技术难度维持秘密的方案而言，这种影响可能比代码文字是否相似更加直接。

6、专利属于另一种保护逻辑

著作权主要保护代码表达，商业秘密依赖信息持续保持秘密；专利在符合授权条件时，可以保护技术方案。即使竞争者使用大模型生成了完全不同的代码，只要实施方案仍落入有效专利权利要求，也可能产生专利风险。

但专利意味着公开技术方案，并非所有软件功能都适合申请。企业需要根据技术生命周期、商业价值、是否容易从产品中识别以及侵权是否容易发现，在著作权、商业秘密和专利之间作出组合选择。

保护方式	主要保护对象	LLM 反编译带来的主要变化	企业应关注什么
软件著作权	源程序、目标程序及具体代码表达	代码可被重新命名和重构，单纯文本相似度可能下降	保留版本和开发过程；从结构、行为和异常特征补充证明

保护方式	主要保护对象	LLM 反编译带来的主要变化	企业应关注什么
商业秘密	未公开的算法、规则、参数、协议等信息	理解本地二进制的门槛下降，事实保密寿命可能缩短	核心秘密尽量不完整下发；落实访问控制、保密制度和取证
专利	符合授权条件的技术方案	代码写法不同并不当然避开技术方案	结合公开代价、生命周期和可检测性评估是否申请

表 2｜LLM 反编译背景下三类软件保护方式的不同作用

二、当代码不再相似，如何证明它来自同一个软件

大模型可以批量修改变量和函数名称，拆分或合并函数，把 for 循环改成 while 循环，更换数据结构，并重新组织条件分支。经过这些处理，两份代码的文字相似度可能明显下降。

这并不意味着共同来源一定无法识别。我们此前分析游戏换皮案件时，曾经讨论过“复制指纹”：相比普通相似，异常一致的数值、错误、无意义设计和开发文档痕迹往往更有来源指向性。类似思路同样可以延伸到软件程序，把分析从文本相似扩展到接触路径、数据、结构、行为和开发过程。

1、第一步：证明可能的接触路径

首先需要回答，对方是否有机会取得并分析目标软件。可以关注软件的获取时间和版本，对方是否曾是客户、合作方、员工或外包方，是否公开讨论过目标产品，以及其产品开发时间是否与接触时间吻合。

接触本身不等于复制，但它会影响后续异常一致性应当如何解释。

2、第二步：寻找数据指纹

数据指纹包括特殊常量、配置表、错误码、协议字段、固定排序和异常边界数值。一个常见数值相同可能只是巧合，但一组缺乏功能必然性的特殊参数同时出现，说明力会明显增强。

3、第三步：寻找结构指纹

结构指纹关注程序怎样组织问题，例如函数调用顺序、数据流、控制流、冗余步骤、无意义分支和特殊文件结构。两份代码即使变量名完全不同，如果都保留了相同的非必要处理顺序，仍值得进一步分析。

4、第四步：寻找行为指纹

行为指纹是让两款软件处理相同的特殊输入，再观察它们是否出现相同输出、相同边界缺陷、相同错误信息、相同异常响应或相同隐藏功能。

最高人民法院知识产权法庭 2025 年公布的一则软件著作权案例指出，源程序比对并不是判断软件相同或实质相似的唯一标准；相同的权利管理信息、设计缺陷、冗余设计、名称、目录结构和错误信息，也可能成为重要证据。[8]

未来的软件侵权比对，可能不只是看两份代码“长得像不像”，还要解释它们为什么连特殊结构和错误都如此一致。

5、第五步：审查开发过程

当代码表面可以被 AI 迅速重写时，开发过程本身的重要性会上升。需求来源、架构设计、Git 提交历史、任务分配、测试用例、版本演进、第三方依赖和构建记录，可以共同说明一个软件如何逐步形成。

如果企业在开发中使用代码大模型，还应保留重要输入材料的来源、生成结果、人工修改和审核记录。这些材料既可以帮助证明独立开发，也可以发现员工或外包人员是否把来源不明的反编译代码输入模型，再以“AI 生成”为由掩盖来源。

6、技术分析的目标，是形成一条可以复查的证据链

单独一个特殊常量、一次异常行为或者一份提交记录，通常都不足以自动得出侵权结论。更可靠的方式，是让接触路径、数据指纹、结构指纹、行为指纹和开发过程相互印证。

技术专家负责把版本、代码、运行行为和时间线拆成可验证的事实；法律专业人员再结合权利范围、接触可能性、表达与思想的区分以及证据规则进行评价。

图3 AI辅助反编译争议中的证据链 众号 · Law Review

三、面对 AI 反编译，权利方和开发方分别应该做什么

前面的证据链不仅决定权利方如何证明复制，也决定开发方应当提前保留什么材料证明独立开发。

下面分别从权利方、被质疑方和软件企业三个角度给出实务建议。

1、权利方：发现疑似 AI 逆向复制后如何取证

尽快固定被控软件的版本、下载来源、发布时间、安装包、签名和文件哈希，避免后续更新后无法回溯。

保存权利源码、正式发布二进制、符号文件、构建日志和提交编号之间的对应关系，证明被主张版本真实存在。

不要只做源码文字相似度比对，还应检查特殊常量、配置表、协议、错误码、冗余结构和异常行为。

设计一组具有区分度的特殊输入，让权利软件和被控软件完成同一任务，记录输出、错误和副作用。

调查对方接触目标软件的时间和渠道，并保全其宣传资料、招聘信息、更新记录和可以公开取得的开发线索。

在符合法律和程序要求的条件下，及时评估证据保全、公证、鉴定或专家辅助的必要性。

2、被质疑方：如何证明真正的独立开发

保留连续、完整的 Git 提交、代码评审、任务记录和版本发布历史，避免只剩下诉讼前临时整理的材料。

保存需求、原型、架构、算法选择和测试用例的形成过程，说明产品为何采用相关结构和参数。

记录第三方代码、开源依赖和公开资料来源，并保留相应许可证。

将竞品分析材料与正式开发资料适当隔离，避免竞品截图、反编译结果或来源不明代码进入正式设计文档。

保存重要 AI 辅助开发任务的输入来源、输出和人工修改记录，能够说明模型究竟参考了什么。

要求员工、外包方和供应商对代码来源、第三方材料和 AI 使用情况作出明确说明。

3、软件企业：不要把“二进制难读”当作唯一防线

更现实的安全原则，是假设客户端代码最终可能被理解。可以放在服务端完成的高价值逻辑，不应仅因开发方便而全部下发本地。

签名私钥和核心授权材料不得硬编码在客户端。

高价值风控、定价和策略规则尽量由服务端执行。

关键参数和阈值支持远程配置、轮换和失效。

本地模块只保留完成用户功能所必需的逻辑。

对于高价值算法，可以结合服务端校验、可信执行环境或硬件安全模块进行隔离。

4、混淆等技术措施仍然有价值，但目标应更现实

符号裁剪、代码混淆、完整性验证、反篡改、代码签名和可信执行环境仍然能够增加分析成本。它们不一定让软件永远无法反编译，却可以延长分析时间、限制一次泄露的范围，并提高篡改和复制的成本。

与此同时，企业还可以为关键模块建立合法、可解释的行为基线和识别性特征，便于未来发现异常复制。这里的目标不是故意制造影响用户的缺陷，而是让权属和版本具有可追溯性。

5、建立组合保护和长期证据体系

软件著作权登记有助于证明登记事项和权属，但它不是防逆向技术。企业需要把著作权、商业秘密、专利、许可协议、员工与外包知识产权条款、访问控制和版本存证组合起来。

其中，容易从产品中识别、生命周期较长且侵权行为较容易检测的核心技术，可以评估专利保护；需要持续更新且不适合公开的参数、数据和规则，更依赖商业秘密管理；具体代码、文档和版本形成过程，则需要通过著作权和完整证据链保护。

图4 AI反编译背景下的双向实务清单

结语：当代码可以被重新表达，来源证明反而更重要

大模型真正带来的变化，不是“复制”这个问题突然消失，而是复制留下的表面痕迹可能越来越容易改变。变量名可以替换，函数可以拆分，代码结构可以重新组织，但一段程序从何而来，并不只记录在源代码文字中。

接触路径、特殊参数、程序结构、异常行为以及完整的开发时间线，可以从不同侧面回答同一个问题：这套程序究竟是独立形成的，还是建立在对既有软件的分析和利用之上？

对权利方而言，未来的软件侵权取证不能只依赖源码文字相似度；对被质疑的开发方而言，“AI 生成”也不能替代完整、连续的独立开发证明。与此同时，企业的软件保护也需要从“尽量不让别人看到源码”，进一步转向著作权、商业秘密、专利、技术隔离和开发过程证据共同构成的体系。

当代码越来越容易被机器重新表达，软件纠纷真正需要回答的，可能不再只是“这两段代码像不像”，而是“这套程序究竟是怎样形成的”。

附：文中几个概念

概念	简释
二进制文件	电脑可以直接加载和执行、但人很难直接阅读的程序文件，例如 EXE、动态库和设备固件。
反编译	把机器码或汇编恢复成更接近 C/C++ 等高级语言的伪代码或代码。
可重新编译	恢复代码可以被编译器接受，并生成新的目标文件或可执行文件。
可重新执行	重新编译后的代码能够运行，并通过规定的测试。它比“能编译”更严格。
行为一致	在给定输入和观察范围内，恢复程序与原程序产生相同结果。
程序指纹	特殊常量、数据结构、控制流程、错误行为等具有来源分析价值的程序特征。
独立开发证据	用于说明软件由开发方自行形成的需求、设计、代码提交、构建、测试和发布记录。

主要参考资料

[1] Hanzhuo Tan et al., “LLM4Decompile: Decompiling Binary Code with Large Language Models,” EMNLP 2024.

[2] Wai Kin Wong et al., “DecLLM: LLM-Augmented Recompilable Decompilation for Enabling Programmatic Use of Decompiled Code,” ISSTA 2025.

[3] Yuxin Cui et al., “PCodeTrans: Translate Decompiled Pseudocode to Trace-Level Equivalent Source Code,” ASE 2026 Research Papers / arXiv:2603.14855.

[4] Yifan Zhang et al., “Constraint-Guided Multi-Agent Decompilation for Executable Binary Recovery,” arXiv:2604.23940v2, 2026.

[5] Peipei Liu et al., “Binary Decompilation LLM with Feedback-Driven Multi-Turn Refinement (AutoDecompiler),” arXiv:2606.16162, 2026.

[6] Luke Dramko et al., “A Taxonomy of C Decompiler Fidelity Issues,” USENIX Security 2024.

[7] 《计算机软件保护条例》，国家行政法规库。

[8] 最高人民法院知识产权法庭：“美国某科技有限公司诉武汉某科技有限公司侵害计算机软件著作权纠纷案”，2025 年 11 月 11 日。

[9] 《中华人民共和国反不正当竞争法》（2025 年修订）。

作者简介

吴圣健

新加坡法律出版社研究院

特邀技术顾问

吴圣健先生毕业于华东师范大学，先后取得软件工程本科学位及电子信息硕士学位。其长期从事人工智能、计算机视觉、图像处理、网络数据分析及软件系统研发工作，具有十余年技术研发与项目落地经验。技术方向涵盖目标检测、多目标跟踪、人脸识别、ReID、图像处理、模型部署推理、网络数据分析及后端系统开发等领域。曾以第一作者身份在AI领域的国际顶级会议（CCF-A类）上发表多篇论文，并作为第一完成人获得多项国家发明专利授权。其产业实践涉及无人智能零售图像识别、金融反欺诈、伪造检测、网络流量预警及后端系统建设等方向。


关于新加坡法律出版社研究院
Singapore-Law-Press Research Institute
新加坡法律出版社研究院系依托新加坡法律出版社设立的研究与内容平台，聚焦知识产权、互联网争议、电子证据与法律技术等方向，持续开展研究观察、专题交流与内容出版，致力于输出兼具专业性与实践价值的法律内容。
官网： www.chinalegal.com.sg/slp
邮箱： sglawpress@chinalegal.com.sg
地址： 7500A Beach Road, #08-305, The Plaza, Singapore 199591
知识产权 | 互联网争议 | 电子证据 | 法律技术
公众号 · Law Review