Protecting confidential data when using AI coding assistants: A practical guide

James A. Smith[1]*, Adam Roff[2], Christopher J. Brown[3]

[1] School of Biological, Earth and Environmental Sciences, University of New South Wales, NSW, Australia

[2] New South Wales Department of Climate Change, Energy, the Environment and Water, Parramatta, NSW, Australia

[3] Institute for Marine and Antarctic Studies, University of Tasmania, Hobart, Tasmania, Australia

* Corresponding author: james.a.smith@dpird.nsw.gov.au

Data availability

There are no original data used in this study. The *confideR* R package is available at https://github.com/smithja16/confideR.

ABSTRACT

1. Generative artificial intelligence (AI) coding assistants can substantially accelerate research workflows in ecology, fisheries science, and related quantitative disciplines, but they also create new pathways for accidental or adversarial disclosure of confidential information. Researchers in these fields routinely work with legally protected and commercially sensitive records (e.g., individual-level records, accurate geographic locations, pre-release data), and even small snippets of code context, console output, comments, or file paths can be disclosive when transmitted to cloud-hosted large language model (LLM) services.
2. This paper provides a practical guide for researchers and institutions to use LLM-based coding assistants while maintaining confidentiality. We (i) catalogue common AI integration points in widely used development environments and R tooling, highlighting features that may activate or transmit context without obvious user intent; (ii) summarise key data exposure pathways – intentional sharing, accidental transmission, and adversarial exploitation such as prompt injection; (iii) propose a four-scenario framework spanning browser chat, editor autocomplete, agentic assistants, and local models, with increasing capability and risk; and (iv) offer actionable mitigations centered on a "develop on simulated data, run on real data" workflow and a complementary two-computer separation strategy.
3. We also introduce the *confideR* R package, which supports session auditing, risk reminders, script scanning for sensitive text, and generation of prompt-ready data fingerprints and simulated datasets.
4. Commercial tools for running LLMs safely are likely to rapidly improve in usability and capability, but so is the sophistication of malicious attacks. Our recommended approach is robust to future changes, because it uses simulated data and the physical separation of confidential data and LLM tools. Together, these recommendations aim to enable responsible uptake of AI coding assistants in the environmental sciences without compromising data confidentiality.

Keywords: artificial intelligence, data governance, information security, agentic AI, environmental science

1. INTRODUCTION

Researchers are increasingly incorporating generative artificial intelligence (AI) in all aspects of the scientific research process (Andersen et al. 2025, Rafiq et al. 2025). Large language model (LLM) tools are proving especially valuable, and can assist scientists with brainstorming ideas, project design, statistical analysis, and code development (Cooper et al. 2024, Gallois et al. 2025, Brown and Spillias 2026, Brown et al. 2026, Spillias et al. 2026a). LLM-based tools such as GitHub Copilot and Claude Code are transformative with their ability to quickly and accurately generate, explain, and debug code (Jiang et al. 2026). But the increasing level of involvement of AI in research, and the connectivity and complexity of AI systems, increases the risks of unintended sharing of data (Bommasani et al. 2021, Tang et al. 2025).

Quantitative ecologists, fisheries scientists, and environmental researchers often work with data that are legally protected, commercially sensitive, or provided in trust by industry or community stakeholders (Tulloch et al. 2018, Tomasic 2023, Fox et al. 2025, Wang et al. 2026). Although specific data types vary across disciplines, most fall into a small number of recurring categories. *Individual-level records* are the most commonly protected type, including catch records linked to vessels or licence holders in fisheries, high-resolution animal tracking records in wildlife ecology, and individual survey responses protected by research ethics frameworks or statistical disclosure rules in social and human-dimensions research (e.g., phone surveys of recreational fishing habits). These also include protected personal information (e.g. contact details). *Location and spatial data* are often considered sensitive data. These can include exact fishing locations derived from vessel monitoring systems, habitats of endangered species, and the locations of culturally significant areas. Spatial coarsening and aggregation are sometimes used to help protect these data. *Commercially sensitive information*, such as data shared by industry under an expectation of confidentiality or preliminary assessment and monitoring results, adds a further layer of obligation. Data breaches, even accidental ones, can have legal and operational consequences and damage trust with stakeholders.

The legal basis for this confidentiality varies by jurisdiction and data type, but commonly includes sector-specific legislation (e.g., the NSW Fisheries Management Act 1994 and the

Magnuson–Stevens Act in the USA), statistical confidentiality rules (e.g., aggregated data can be shared or omitted when too few entities exist), and data-sharing agreements between research institutions and industry or government bodies. Confidentiality expectations can also be informal rather than statutory, with data providers sharing data under an implied expectation of protection. An unresolved question is liability when an AI agent, rather than the researcher, performs the action that discloses the data, a scenario that existing agreements and legislation rarely explore.

When a researcher uses a cloud-based AI assistant, their input – including code, comments, error messages, and any data they provide – is typically transmitted to external servers for processing. A challenge is extracting the benefits from AI tools without transmitting confidential information (NSA 2024, BSI & ANSSI 2024). This risk is not hypothetical: high-profile incidents of accidental data exposure through cloud services have occurred across research and government contexts (e.g. UK Biobank, Devlin and Burgis 2026), and the specific workflows of quantitative environmental science create several non-obvious exposure pathways that have received little attention. In the UK Biobank case, approved researchers accidentally published participant data to public GitHub repositories alongside their code, with takedown notices then served to remove these data. It was shown that a participant could be reidentified from a birth month and year and the date of a single surgery (Devlin and Burgis 2026). The risk of accidental data publication increases with agentic coding tools, which can manage interactions with public data repositories (Section 3).

We aim to provide: (1) a practical catalogue of how AI features are integrated into the integrated development environments (IDEs) and R packages commonly used by environmental researchers; (2) a summary of data exposure pathways, including intentional sharing, accidental transmission, and exploitation via prompt injection; (3) a tiered framework of four usage scenarios with increasing capability and risk; and (4) practical recommendations centered on a "develop on simulated data, run on real data" workflow, and promoted using the developed *confideR* package. Although exposure of confidential information can occur for other research workflows, e.g. agentic AI edits of manuscripts or spreadsheets, our focus here is on coding assistants, and the exposure of data through the data analysis process. Much of this advice is

useful for coding in any language, but R is the focus here given its popularity for ecological research.

Box 1. What are LLM-based coding assistants?

Large language models (LLMs) are AI systems trained on vast quantities of text and code. Coding assistants built on LLMs (such as Claude Code) can generate, explain, debug, translate, and execute computer code, including in languages common for research scientists, e.g. R, Python, Stan, and TMB. Coding assistants range from simple *chat interfaces* (where you paste code and ask questions) to *autonomous agents* that can read project files, run commands, and iteratively develop entire analysis pipelines (e.g. Fig. 1). When a user interacts with these tools, the input – including any code, comments, data, or console output you provided – is usually transmitted to remote servers for processing. AI models can be run locally and do not connect to remote servers, but these models are smaller and can be less competent. Also, a local model may still be able to connect to the internet and expose data through less obvious pathways.

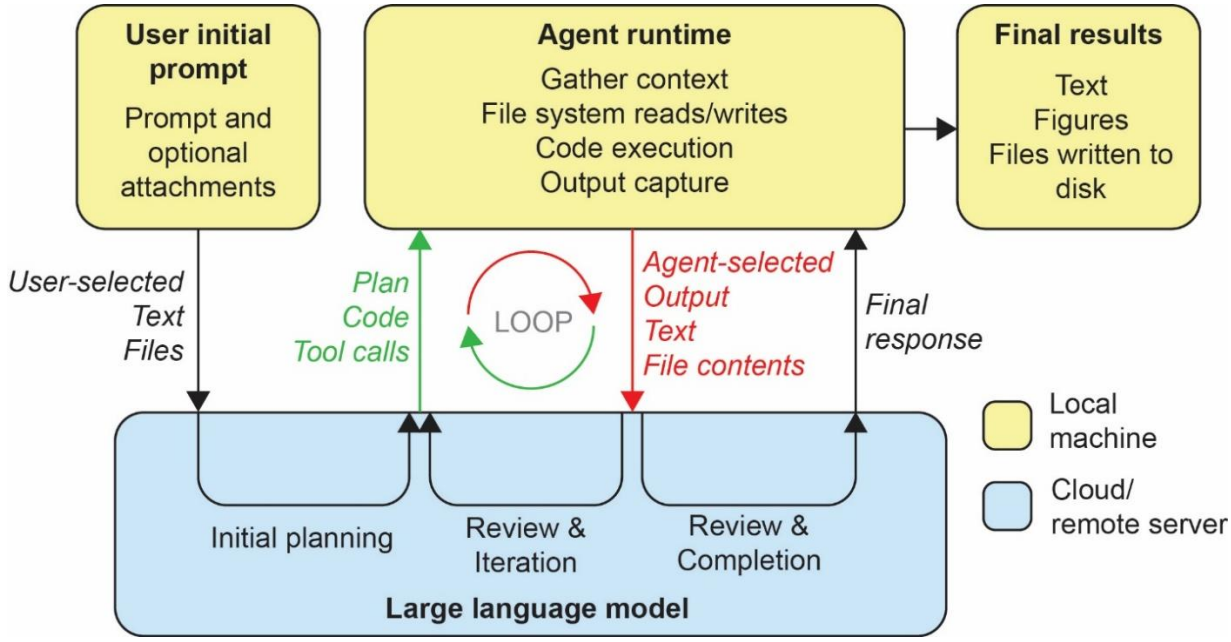

Fig. 1. Example of how a cloud-based AI agent (e.g. Claude Code) contributes to an analysis workflow. User- and agent-selected data leaves the local machine and is used by the remote LLM. The iterative loop between the local agent process and the remote LLM can lead to accidental transmission of confidential information via agent-level decisions regarding what output and context should be sent to the LLM. Green arrows show the model's plans, code, and tool calls arriving at the local machine; the red arrow shows agent-selected local content leaving it.

2. HOW AI TOOLS CAN SEE YOUR DATA

LLM-based AI tools can run locally (i.e. personal servers or PCs), on institution-managed servers, or on internet-hosted cloud services. The most powerful and widely used tools are typically cloud-based, meaning that prompt data and uploaded files are transmitted to remote servers and handled according to the provider's data governance arrangements. Agentic tools can also expose data that is stored outside an accessible working folder by bypassing these ‘soft’ access permissions, which are typically instruction-level rather than a technical barrier. The outcomes of sharing confidential data with a cloud AI are uncertain. While it is generally unlikely that another user could extract that specific data via direct prompting, it is not impossible, especially if the data are used in model training or fine-tuning (Carlini et al. 2021, Sun et al. 2023). Regardless, the data are typically stored outside permitted environments, and this alone is considered a form of data leakage. Also, some data agreements or legislation may require data to be stored domestically, which could be violated by the use of cloud-based LLMs.

2.1. AI TOOLS IN YOUR INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

AI coding tools do not need to use an IDE and often are used via the command line (De Masi 2026). But most researchers in the environmental sciences use an IDE, and most issues related to IDEs can be considered relevant to using the command line. The most common IDEs are RStudio (Posit Team 2025a) for coding in R, Stan or TMB, and increasingly Positron (Posit Team 2025b) which includes Python support, and Visual Studio Code (VS Code) for diverse language support. Each IDE has multiple pathways for integrating AI tools – some requiring explicit opt-in while others are enabled by default or configurable to start automatically. A critical concern is that a researcher may be using AI features without realising it, especially when AI integration is not obvious or connects when the IDE is opened (e.g. enabling the Positron Assistant without realizing its abilities). Further, a researcher may activate AI assistants in one project, which can be automatically carried over to other projects. Table 1 catalogues the primary AI integration points in each IDE.

Table 1. Example AI integration points in three common IDEs. * There are many R packages and IDE tools built for AI integration, some of which are detailed in Table S1 and Table S2.

IDE	AI feature	What is transmitted	Activation not obvious?
RStudio	GitHub Copilot autocomplete	Code in current file, open files, comments	Unlikely, requires explicit opt-in
RStudio	*chattr** package (chat widget)	Prompts, data frame names/structures, file paths	Possible, can auto-connect via `.Rprofile`
RStudio	*gptstudio** add-in	Selected code, prompts	Possible if pre-installed on shared workstation
RStudio	*gander** inline suggestions	Code context around cursor	Low, requires explicit activation
Positron	Positron Assistant (chat & agent)	Files, code, console history, loaded data, session state	Yes, provides rich context by design
Positron	Copilot autocomplete	Code in current file, nearby files	Yes, enabled by default with Copilot subscription
Positron	Agent mode	Can read/write files, execute code, run commands	No, requires explicit activation
VS Code	GitHub Copilot (autocomplete)	Code in current file, open tabs, comments	Requires explicit setup but easy to reengage in subsequent projects
VS Code	Agentic assistant (e.g. Claude Code)	Workspace files, terminal output, conversation	Requires explicit setup but easy to reengage in subsequent projects
VS Code	Cursor / Windsurf / Continue.dev	Varies; generally broad workspace access	Depends on tool

Box 2. The .Rprofile risk

Several AI-enabled R packages can be configured to start automatically when you open R, using options set in a `.Rprofile` file. For example, `options(.chattr_chat = ellmer::chat_anthropic())` will silently establish a connection to Anthropic's API every time R is started. This is easy to overlook or forget, meaning that AI features may be active without explicit activation that session. Before working with confidential data, check the `.Rprofile` (run `usethis::edit_r_profile()` to view it) and remove or comment out any AI-related options. Similarly, check RStudio's Global Options (e.g. if Copilot is turned on). The *confideR* package (see Section 4.5) is designed to encourage this process. The same should be done for Positron and VSCode, the latter having a Restricted Mode to limit AI connectivity during code exploration (before it is run).

3. PATHWAYS TO DATA EXPOSURE

We organise data exposure risks into three categories of increasing sophistication: intentional sharing, accidental transmission, and adversarial exploitation (Table 2; Greshake et al. 2023; Shayegani et al. 2023). This taxonomy helps researchers understand that the risk spectrum extends to include threats that may not be obvious.

Table 2. The three data exposure pathways.

	Intentional sharing	Accidental transmission	Adversarial exploitation
Mechanism	User pastes data or uploads files to AI	AI tool silently sends code context, comments, environment metadata, pushes data to public GitHub repository	Malicious instructions in documents trick AI into exfiltrating data
User awareness	High, but consequences may not be understood	Low, user may not know what is transmitted, or what triggers the transmission	None, attack is invisible to user
Primary defence	Education; use simulated data	Audit IDE settings; use simulated data; sandboxing or two-computer approach	Human approval turned on; two-computer approach

Who receives the data	AI company/owner of the server	AI company/owner of the server; potential for data to be made public online via an AI agent's actions	Attacker

Intentional data sharing is the most straightforward pathway: a researcher knowingly provides data to an AI tool. This includes pasting data frames into a browser chat to ask formatting questions, uploading CSV files to ChatGPT or Claude for analysis, or using R packages like *mall* (Table S1) that are designed to process data frame columns through an LLM. The risk is often underestimated: researchers may not realise that the AI provider stores or processes the data, or may not consider that even aggregate statistics can be disclosive when combined with other information.

Accidental transmission is the primary realistic threat for most researchers. AI tools silently transmit contextual information like code, comments, console output and environment metadata to a server. That data may contain confidential information the researcher did not intend to share. Table 3 details the most common pathways with examples. Agentic tools create an additional risk of unintended data access, whether because sensitive files were accidentally included in a shared folder and used by the agent as context, or because the agent reached beyond the files and folders it was permitted to access. Agentic tools can also publish data on the web, for example via committing private files to a public GitHub repository (Fig. 2). Agents often execute code directly in the terminal or with Python. A user who is not familiar with these languages may not realize the agent is viewing confidential data, even if they have required 'approve all' on agent tool use.

Adversarial exploitation (e.g. prompt injection) is a more sophisticated threat, particularly relevant for researchers using AI agents. Prompt injection is when malicious instructions are hidden inside content that the AI processes, tricking it into taking unintended actions (Greshake et al.. 2023, Chen et al. 2025). This is the AI equivalent of a phishing attack, but instead of tricking a human into clicking a link, it tricks the AI into executing hidden commands. Examples relevant to coding workflows include: hidden instructions in GitHub repository README files that cause Copilot to modify IDE settings and enable auto-execution of commands without user

approval; and PDF documents containing invisible text that instruct AI assistants to search for and exfiltrate confidential files. Recently, a malicious repository faking the popular 'privacy filter' confidentiality tool was downloaded many times (HiddenLayer Research Team, 2026).

Issues with adversarial and accidental transmission are not necessarily contained to the working project folder. An AI agent can theoretically navigate anywhere on your computer using terminal commands. While this is typically well constrained, this capability has led to serious unintended actions (Tian et al. 2023, Ruan et al. 2024), including a widely reported 2025 incident in which an agent deleted a company's production database contrary to explicit instructions (AI Incident Database 2025). The instructions to agents to restrict their actions to a working folder (e.g. in AGENTS.md or .ignore files) are prompts that can be bypassed by conflicting or malicious instruction (Andriushchenko et al. 2025a). Recent work shows that LLM safeguards themselves can be overridden with relatively simple adaptive prompting ('jailbreaking'; Andriushchenko et al. 2025b), which increases the risk that injected instructions will be followed rather than ignored. Thus, use of agents on your computer without strict and enforced sandboxing and permission controls poses a risk to any data on that computer.

4. A FRAMEWORK FOR SAFE AI USE

This section presents a unified framework integrating both researcher behaviours and technical tools. First, we summarise behaviours and tools for protecting confidential data (Section 4.1). Then we discuss the value of using simulated or 'fingerprint' data to avoid leakage (4.2), and more extreme solutions such as using two computers (4.3) or a sandboxed or local environment (4.4). Then we introduce the *confideR* R package (4.5), which implements the tool-based protections for R users, then we cover some existing enterprise and commercial options for institutions (4.6). Finally, we provide a comparison of general AI use scenarios – ranging from browser chats to running local LLMs – and their potential for exposing data (4.7).

211 **Table 3.** Some likely pathways for accidental data exposure when using AI tools, with
212 environmental science examples.

Pathway	How it happens	Example	What is exposed
Pasting console output	`head()` or `summary()` output pasted into AI chat	`head(survey_data)` showing name = “James Smith”, postcode = 2000, vessel_name = "FV Doris Bay"	Individual identifiers, location, sensitive attributes
Descriptive comments	In-line comments read by AI autocomplete or agent	`# VMS data from Snapper fishery, Spencer Gulf` `# Tracking data: white sharks NSW`	Topic, location, time period, species, etc.
Error messages	R errors include sensitive values, pasted into AI chat	`Error: duplicate key 'VESSEL_SEALORD_0892'` `Error in merge: 'site_GPS_threatened_orchid' not found`	Real identifiers (e.g. names, permit numbers, locations, etc.)
File paths	Absolute paths in code or error output, pasted into chat or read by AI agents or autocomplete	`setwd("/home/jsmith/DPIRD/Observer_Gillnet_Sydney/")` `read.csv("/projects/endangered_species/nesting_sites_2025.csv")`	Species, location, organisation, etc.
Automatic file indexing	AI indexes project files for faster look-up	GitHub copilot workspace indexing; Positron AI context	Any file in project folder
Autocomplete context	AI (e.g. Copilot / Posit AI) reads open editor tabs and console history	CSV open in IDE tab; console showing `summary(cpue_data)` or `str(nesting_data)`	Data visible in the editor or console at time of query
Packages enriching prompts	R package augments prompts with session context	`chattr` included data frame name `observer_gillnet_2026` in prompt	Data frame and column names, potentially values
.Rprofile auto-connect	AI API established silently at session startup	`options(.chattr_chat = ellmer::chat_anthropic())` in .Rprofile	All subsequent AI interactions routed to cloud
Rendered notebooks	`.Rmd` or `.qmd` stores output in file	`kable(head(catch_data))` or `print(site_locations)` rendered into output	Any data printed during notebook development
Agentic coding – model objects	Agent accesses model object which contains raw training data	`M1 = glm(Y ~ X1 + X2, data = mydata)`; unloading `mydata` does not remove it from `M1$data`	Raw data used to fit a model
Agentic coding – repository push	Agent pushes files to public GitHub repository	Agent pushes project folder containing raw data, cached .Rdata, or rendered notebook with real values	Any data within the project
Agentic coding – file review	Agent reads files beyond the immediate script to gain greater context for its actions	Agent searches within the working directory or potentially outside for what it deems required additional context. This rarely requires additional user permission	Potentially any file on your computer, more likely files within the current project

213

Fig. 2. Example of an agentic AI coding tool (Claude Code) preparing to commit a data file to version control. Although the agent correctly seeks approval, accepting the second option ("allow git add * and git commit for all projects") would grant standing permission to stage and commit any file, including datasets containing confidential records, without further prompts. This illustrates the risk of agent tools gaining persistent, broad file access through routine approval decisions. Working with simulated data eliminates the confidentiality consequences of this risk, as any file the agent commits contains no real records.

4.1. Protection as a combination of behaviours and tools

Protecting confidential data when using AI requires changes to researcher behaviour and the deployment of technical tools. Table 4 shows a set of protection approaches organised in this framing. Each row identifies whether an approach is primarily a behaviour (requiring researcher action), a tool (enforced by software), or both; which of the three threat pathways it addresses;

which usage scenario it applies to; and whether the *confideR* package implements it. The four scenarios for using AI coding tools with confidential data (Table 5) range from lowest risk and simplest setup (browser chat) to highest capability with maximum protection with limited data transmission (local model).

4.2. Simulated and fingerprint data strategy

The core recommendation of this paper is that when using AI we only reveal simulated or summarised data, a process known as 'data masking' (Australian Cyber Security Centre, 2025). Provided that the simulated data accurately represents the structure of the data, it will be valuable to an AI assistant. These data can be a simulated set of observation-level data (i.e. a dataset suitable for analysis) or a fingerprint (i.e. summary or schema) representing the structure of the data (number of variables, data types and ranges, etc.). A fingerprint is useful for getting advice on topics such as model structure, whereas simulated data can also be used to fit models. The challenge with simulated data is ensuring representativeness as well as adequate masking of sensitive data. Some data features may be more difficult to simulate, e.g. zero inflation or correlation among variables, so allowing some generality in prompts to AI, and using detailed diagnostics, are useful ways to encourage accurate analysis of the raw data. Simulated or summarised data derived from confidential data should themselves be checked for remaining disclosive detail before they are shared (see Recommendation 7, Section 5).

Removing sensitive information from data will typically include removing or renaming personal information (e.g. in fisheries, captain or vessel names), and likely simulating continuous variables (e.g. in fisheries, catch and fishing effort) with realistic statistical properties, e.g. `rnorm()`. Additional masking is achieved by changing variable names or levels to nondescript placeholders, e.g. "Variable_1", "Species_A", to reduce the interpretability of shared data. Geographic data revealing precise locations (e.g. in fisheries, individual fishing events) is often considered confidential, so these can be coarsened to an acceptable spatial resolution (or jittered by a random amount from an adequate range), as can the date of observation. Matched environmental data (e.g. sea surface temperature or depth) should also be jittered to avoid any reconstruction of location. The *confideR* package provides tools to create prompt-ready

fingerprints and simulated data (Section 4.5). Other packages potentially useful for data generation, sensitive data detection, or data masking include: *FakeDataR* (Ahmed 2025) which is designed for LLM workflows, *synthpop* (Nowok et al. 2016), *deident* (Cook et al. 2024), *SecretSentry* (python; Jilani 2025), and *llmshieldr* (operates during LLM interactions; Chakraborty 2026).

The full 'develop on fake, run on real' workflow (Table 4) is to: 1) create realistic simulated data, or create a detailed fingerprint; 2) use AI assistance to develop the analysis and code using only the simulated data; 3) transfer this code to a secure environment (e.g. IDE with AI disabled, or a separate computer); 4) run the code using the real data, and without AI assistance; 5) if bugs arise when using the real data, use these to update the simulated data or report errors to the AI provided that no real data are also reported.

4.3. Physical separation strategy

Physical separation is an extreme strategy to avoid exposure of confidential data by any route, and especially useful when using AI agents. One machine is *secure* and AI-free and optionally disconnected from the internet; this machine is used for loading and manipulating the raw data. A second machine is the *workshop* and used for AI-assisted analysis; this machine has no access to the raw data. The second machine can be a cloud workspace rather than a physical computer. The secure machine holds the raw data, generates clean data fingerprints or simulated versions to send to the workshop computer, and runs the final analysis. This physical separation ensures that even unrestricted access by an AI agent cannot leak confidential data. It is also a neat solution for users who want an extremely effective but low-tech strategy. A limitation to this strategy is the ever-increasing integration of AI tools in standard software, which may present a challenge to creating a truly AI-free computer; although disconnecting the first computer from the internet during data manipulation can help address this.

Table 4. Some protection approaches for confidential data, organised by type (behaviour, tool, or both), threat pathway addressed, and implementation. B = behaviour (researcher action required); T = tool (software-enforced); B+T = both. Threat types: Intentional = deliberate sharing; Accidental = silent transmission; Adversarial = prompt injection. *confideR* functions (Section 4.5) assume `confidential_mode_on()` has been called. * optional sandboxing layer.

Protection approach	Type	Threat addressed	Scenarios (Table 5)	*confideR* function	Notes
Never paste real data into AI chat	B	Intentional	1–4	`audit_session()` warns if AI is active	Fundamental protection
Use simulated data for development ('develop on fake, run on real')	B+T	Intentional, Accidental	1–4	`simulate_data()`, `simulate_from_fingerprint()`	Fundamental protection; see Section 4.2
Summarise data structure for AI; no raw values	T	Intentional	1–3	`fingerprint()`, `format_for_prompt()`	Provides AI with data schema only; extreme values and sensitive variable names redacted
Two-computer separation	B	All three	1–4		Strongest practical protection; see Section 4.3
Clear AI API keys before data work	B+T	Accidental	1–4	`confidential_mode_on()` backs up and clears keys	Keys restored automatically by `confidential_mode_off()`
Don't load AI packages while real data is loaded	T	Accidental	1–4	`confidential_mode_on()` installs package hook to block some packages	Prevents `library(chattr)`, `library(mall)` etc. silently loading
Audit IDE and `.Rprofile` for active AI	B+T	Accidental	1–4	`audit_session()`, `audit_rprofile()`, `audit_env_keys()`	Checks RStudio Copilot, Positron Assistant, `.Renviron` keys, VSCode extensions, etc
Scan scripts before sharing or pasting	B+T	Accidental, Intentional	1–3	`scan_script()`, `check_notebook_outputs()`	Detects sensitive variable names, file paths, `head()` calls in code, etc
Close data file tabs; clear console before using autocomplete	B	Accidental	2		Prevents Copilot/Positron reading open files or console output
Use generic comments while AI tools are active	B	Accidental	2–3	`scan_script()` flags descriptive comments	Comments are transmitted as context by all autocomplete tools
Container-based encryption	T	Accidental, Adversarial	1–4		Data stored in encrypted locations when AI agents are active
Pre-filter text for sensitive info before cloud submission; 'prompt filtering'	T	Intentional	1–2		External: OpenAI Privacy Filter (text only, not attached files); Microsoft Presidio

Use local model; disconnect internet	B+T	Intentional	4	External: e.g. Ollama (Llama, Qwen, DeepSeek); eliminates transmission risk
Use enterprise-tier AI subscription	T	Intentional	1–3	External: Microsoft 365 Copilot Enterprise Data Protection, Claude Code with Zero Data Retention, AWS Bedrock; companies state that (some) data not used for model training
Enable security features in agent-able IDEs, e.g. Restricted Mode in VS Code; 'assistant.aiExcludes' setting in Positron to exclude certain file types, e.g. "*.csv"	T	Accidental, Adversarial	3	Numerous features exist to avoid malicious code and unwanted agent actions; download only trusted extensions
Enable human approval for all agent actions, and leave this option on	B+T	Accidental, Adversarial	3	Avoids unwanted file access; blocks prompt injection attacks from executing file access or commands without review
Sandbox AI agent to specific directories	T	Adversarial, Accidental	3*, 4*	External: Docker (--network=none, explicit volume mounts) or virtual machine

4.4 Local LLMs and Sandboxing

The software equivalent of the ‘two computer’ solution is to run local LLMs in a sandboxed execution environment. Local LLMs and sandboxes address different links in the exposure chain. A local model controls where data are transmitted, whereas a sandbox controls what an executing agent can read, write, and contact – and they can be adopted independently or together.

A local LLM is a downloaded model run entirely on local hardware (e.g. via Ollama, LM Studio, or llama.cpp), with the capability that prompts, code context, and any data values can remain entirely on the researcher's machine. The downsides of local LLMs are substantial hardware requirements (large VRAM or unified memory) and trade-offs in model capability. Open-weight models (freely available LLMs) typically have fewer parameters than frontier models, so encode smaller amounts of knowledge and may be less effective at complex tool use, reasoning, and planning. Local LLMs are often quantised, meaning that model weights are rounded to save memory, improving their speed of inference. Moderate quantisation (e.g. 4-bit) typically sacrifices little accuracy, but aggressive quantisation noticeably degrades performance. Despite these limitations, local LLMs can still provide useful research assistance, including for ecological modelling tasks (Spillias et al. 2026b). Hardware requirements increase with model size, ranging from models that can run on laptops or consumer GPUs to larger models that require specialised workstations or multi-GPU servers, with four broad classes emerging (Table S3). Consequently, not all open-weight models are practical for local deployment by individual researchers. In general, larger models provide stronger coding, reasoning, and agentic capabilities, although performance differences depend on both model architecture and task. For many research coding workflows, mid-sized local models can provide a useful balance between capability, cost, and confidentiality. We stress that ‘local’ describes a configuration rather than a category of software: local LLM interfaces may still contact the internet for update checks, model downloads, telemetry, or web-searches. Verifying that inference is done offline, by disabling network access (see Scenario 4 in Table 5), is more reliable than trusting application settings.

A sandbox is a controlled execution environment that isolates software from the host system and restricts its access to resources such as files, networks, and system processes. Sandboxes are typically implemented using technologies such as containers (e.g. Docker), virtual machines, or other isolation mechanisms. Containers provide lightweight isolation by sharing the host kernel while constraining filesystem and process access, whereas virtual machines provide stronger isolation by emulating separate operating systems. In terms of LLM agents, a sandbox limits what the agent can execute, what files it can access, and whether it can communicate externally. For example, filesystem access can be restricted to a specific working directory, and network access can be disabled. A well-configured agent sandbox follows a few simple rules: mount only the project folder into the sandbox, never a home directory; keep credential stores outside it; disable outbound network access (e.g. Docker's --network=none) or restrict it to the model provider's endpoint; and treat the environment as temporary, so that nothing sensitive accumulates in it. In a simulated-data workflow (Section 4.2), the sandbox holds only simulated data and become a second layer of defence.

Sandboxing reduces but does not eliminate risk (Marchand et al. 2026), and even well-resourced users cannot treat isolation as absolute (OpenAI 2026a). A sandbox bounds risk rather than eliminates it, and an agent that escapes a sandbox containing only simulated data has nothing confidential to leak. If the sandboxed environment contains sensitive data (e.g. the working directory is mounted into the container), that data remains exposed to any cloud-connected component of the tool. Setup complexity is a limiting factor to uptake by researchers. Configuring local LLM infrastructure or sandboxes requires skills that are outside the typical training of environmental scientists, and misconfigured environments may allow unintended data access or network communication. Sandboxing tools are developing quickly (e.g. NVIDIA 2026), and leading AI providers are beginning to integrate sandboxes into their LLM agents. Improvements in this software should make sandboxes more accessible to a broader range of computer users.

4.5. The *confideR* package

We have created an R package *confideR* (https://github.com/smithja16/confider) to encourage safe data handling of AI-capable IDEs. The intention is that *confideR* is loaded at the start of any coding session to identify risks and provide reminders before confidential data are loaded (Fig. 3). It has four main features: 1) identify initial connectivity risks and provide reminders, 2) restrict the loading of R packages that enable risky AI features, 3) scan scripts for potentially sensitive text, and 4) generate safe fingerprints and simulated data that can be shared with an AI based on the confidential data which cannot be shared. *confideR* complements rather than competes with data-synthesis and anonymisation packages (Section 4.2) – those generate or transform data, whereas *confideR* audits the session and IDE environment in which AI tools operate. This package cannot detect all connections, especially for IDEs other than RStudio, so it provides an additional layer of support and encourages constant consideration of data exposure. Researchers using other coding languages such as Python can still use *confideR* (in an R terminal) to evaluate some IDE-specific risks, but must also rely more on manual checking to ensure confidential data are protected. Because the catalogues of AI integration points (Tables 1, S1 and S2) date quickly, we will maintain updated versions in the *confideR* repository, with the paper detailing the stable principles.

4.6. Enterprise and commercial protection tools

Beyond the individual-researcher tools described above, institutional and commercial options exist that address data exposure through contractual or architectural means. These are complementary to, not substitutes for, the personal and behavioural protections in Table 4. A common example is Microsoft 365 Copilot Enterprise Data Protection (EDP), which processes prompts and responses within an organisation's service boundary. EDP provides a contractually backed assurance that Copilot usage will not expose data externally or contribute to model training. This may not be sufficient protection in all cases, and researchers will often need to use tools other than Copilot. A recent example of increasing commercial protection is OpenAI's Privacy Filter (OpenAI 2026b). This is an open-weight, locally-runnable model designed to detect and redact personally identifiable information (PII) in text before it is submitted to a cloud AI service. This is a neat solution for PII, but may be difficult to verify, currently works only on unstructured text (not attachments or CSV data), and the PII detection may not automatically

```
> audit_session()

========================================
   confideR: Session Audit Report
========================================

  Confidential mode:  INACTIVE
  --> Call confidential_mode_on() before loading data

  [!!] IDE:                      VS Code
    [i] VS Code detected. Extension settings, Claude Code config
        (~/.claude/), and Copilot tokens are stored outside R's
        visibility. Automated checks cover environment variables
        and R packages only. Manual verification is essential.

    !  AI-related VS Code extensions detected in ~/.vscode/extensions/: anthropic.claude-code-2.1
.153-win32-x64, anthropic.claude-code-2.1.159-win32-x64. Verify these are disabled for this works
pace.

  [XX] AI packages loaded:       ellmer
  [OK] AI API keys:              (none)
  [OK] .Rprofile AI entries:     (none)
  [OK] AI processes:             (none)

  ----------------------------------------
  [XX] OVERALL:                  RED
  ----------------------------------------

  ACTION REQUIRED: AI features detected.
  Run confidential_mode_on() to clear keys and
  unload AI packages before loading data.

  Manual checks (confideR cannot verify these automatically):
    [ ] Extensions sidebar: confirm Copilot and other AI extensions
        are disabled (confideR scans ~/.vscode/extensions/ but cannot
        read per-workspace enabled/disabled state)
```

Fig. 3. Example reporting from the `audit_session()` function from *confideR*. The VS Code IDE was detected, as were signs of agent attachment, and a risky R package was loaded. This resulted in a RED rating, indicating user attention is needed before loading sensitive data.

cover sector-specific labels (e.g. vessel licence numbers, species names), and will not mask variables such as geographic coordinates.

Confidential data may also be subject to data sovereignty obligations, meaning that the LLM used for inference must occur within a specific jurisdiction. This can constrain which model a researcher is permitted to use, and lead to less capable models being used in order to process data inside a compliance boundary. Numerous cloud providers offer useful enterprise protections, but

these require organizational contracts, apply only when configured correctly, and still may not protect from adversarial data leakage. These kinds of tools will continue to develop quickly, but the burden for protecting confidentiality will still typically remain with the researcher, which highlights the value of simple fundamental approaches such as isolating confidential data from the entire AI process.

4.7. Scenario examples

In Table 5 we provide a comparison of four likely usage scenarios, including what data is transmitted, whether the AI can access files or run commands, prompt injection risk, setup difficulty, and key protections. The protection approaches in Table 4 apply across scenarios; this table shows how they are prioritised within specific scenarios. Most researchers can work safely at Scenario 1 or 2 for small to medium tasks. Note that the ordering reflects what each tool transmits by default, not how researchers behave: browser chat is technically the least exposed, but it is also where researchers most readily paste raw data, so the intentional-sharing pathway (Section 3) is most active in the first scenario.

The following are examples of safe AI use for each of the four scenarios. These are not prescriptive but demonstrate one possible approach:

- *Browser chat:* 1) Researcher (Res) wants modelling assistance and code; 2) Res generates a data summary free of sensitive information and posts it with their prompt into a browser chatbot; 3) Res copies analysis code from chatbot and pastes into IDE, then runs code on the raw data; 4) Conversation with chatbot can continue for error-checking etc. but prompts never include sensitive information.
- *IDE autocomplete:* 1) Res has GitHub Copilot subscription and installs it in RStudio; 2) Res starts a new session, ensuring RStudio is free of open data tabs and working directory free of sensitive file paths or names; 3) Res starts a script with commented analysis goals and accepts (or not) code suggestions; 4) Res never adds comments containing sensitive information; 5) Res turns GitHub Copilot off in settings before loading and analysing raw data.

- *AI agent*: 1) Res wants to refactor existing code to improve readability and update methods; 2) Res has a Claude subscription and downloads the Claude Code extension for VS Code; 3) Res uses confidential data to simulate realistic fake data then removes confidential data from this computer; 4) Res opens VS Code and Claude Code extension, sets a new clean working directory, and prompts for required tasks, including testing code on fake data; 5) User ensures every local agent action is manually approved and avoids "don't ask again" approvals. 6) Res closes VS Code, then tests new scripts on reinstalled confidential data.
- *Local model*: 1) Res wants chatbot assistance directly on confidential data, so finds a machine with sufficient memory to load a local LLM; 2) Res downloads an interface (e.g. LM Studio) and suitable local LLM (e.g. Qwen3.6-27B); 3) Res disconnects internet to ensure no accidental leakage; 4) Res pastes prompts including raw data into LM Studio interface; 5) Res runs code in separate IDE and continues chatbot conversation if necessary; 6) Res ensures chat logs, backups, clipboard etc. are sanitized before connecting to internet.

Table 5. Four general scenarios of how AI coding tools might interact with confidential data. *Scenarios 3 and 4 can optionally be run in a sandbox (e.g. Docker container or virtual machine) as an additional protection layer; this adds technical complexity but not the fundamental data transmission behaviour.

	1. Browser chat	**2. IDE autocomplete**	**3. AI Agent***	**4. Local model***
What it is	Copy-paste between chatbot and IDE	AI code suggestions as you type	AI reads files, runs commands	Model stored on local computer, no internet required, can include agentic AI
Data transmitted to cloud	Only what you paste	Code context from open files	Workspace files, commands, conversation	Usually nothing; but lack of sensitive data in outputs (e.g. .md or .html files) needs checking, as does lack of network connectivity
AI accesses files?	No	Open tabs only	Yes, with permission	Yes if agentic, but on local machine
AI runs commands?	No	No	Yes, with permission	Yes if agentic, but sealed

Injection risk	Very low	Low	High	Low–Moderate; injection can mislead a local agent, but offline operation stops data leakage
Setup difficulty	None	Extension install	Extension install; can require command line experience	Interface install, LLM download, configure local/private use
Best for	Quick questions, learning, generating code chunks	Day-to-day coding	Complex multi-file development	Privacy-sensitive work on any project; specific tasks suited to smaller models
Key protections	Don't paste real data; paste clean data summaries or simulated data	Close data tabs; use generic in-line comments; ensure agentic capabilities switched off	Use simulated data; approval of file changes turned on	Use local model; disconnect internet; optionally use simulated data

5. DISCUSSION AND RECOMMENDATIONS

AI software is developing and being integrated into consumer software faster than security protections are keeping up. While many researchers and organisations are enthusiastic about the opportunities that AI presents, there is limited advice on how to protect sensitive data. We hope our guidelines can contribute to better educating researchers so they can prevent leakage of sensitive data.

We provide several recommendations for researchers, research communities, and institutions:

1. Adopt "develop on simulated data, run on real data" as the default workflow for sensitive data; it is robust to changes in tools and attacks because confidential data never enters the AI process;
2. Know and regularly audit the processing and coding environments for their AI integration (IDE settings, .Rprofile, API keys);
3. Keep human approval enabled for all agent actions and never grant standing permissions, unless in a secure environment (e.g. a sandboxed workspace);
4. Institutions should provide training on AI tool and their data-exposure pathways rather than rely on blanket bans, which encourage unmanaged "shadow AI" use; and consider

hiring specialised data managers to remove some of the responsibility for data protection from researchers;

5. Provide sanctioned infrastructure, e.g. enterprise/zero-data-retention agreements, sandboxed workspaces, or local-model servers, so researchers aren't left to improvise;
6. Consider updating data-sharing agreements to state explicitly whether and how AI tools may be used with the provided data;
7. Researchers could focus on developing better tools and norms for simulating ecological data with realistic properties (zero inflation, spatial and temporal correlation) plus disclosure checks to verify the data contain no remaining sensitive information.
8. A layered approach to data protection is effective: combining policy controls, security checks, restricted permissions, and data simulation and masking provides multiple opportunities to prevent or detect leakage.

Notice of generative AI use

AI tools played a supporting role in this work. Copilot (GPT 5.4) and Claude (Sonnet 4.6) assisted with organising ideas, drafting and formatting sections, and improving readability. Claude Code (in VS Code) supported development of the *confideR* R package, working from detailed specifications provided by the authors, and run on a personal computer to isolate the environment. The intellectual content, scientific judgements, and final text are the authors' own, as is responsibility for both the article and the R package.

Conflict on interest

The authors declare no conflicts of interest.

Author contributions

JAS conceived the study, JAS and CJB developed the scope and structure, JAS led software development, all authors contributed to manuscript drafting and reviewing.

REFERENCES

Ahmed Z (2025). FakeDataR: Privacy-Preserving Synthetic Data for 'LLM' Workflows. R package version 0.2.2, https://CRAN.R-project.org/package=FakeDataR.

AI Incident Database (2025). Incident 1152: LLM-driven Replit agent reportedly executed unauthorized destructive commands during code freeze, leading to loss of production data. Available at: https://incidentdatabase.ai/cite/1152/ (Accessed: 11 August 2026).

Andersen, J. P., Degn, L., Fishberg, R., Graversen, E. K., Horbach, S. P., Schmidt, E. K., ... & Sørensen, M. P. (2025). Generative Artificial Intelligence (GenAI) in the research process–A survey of researchers' practices and perceptions. *Technology in Society*, *81*, 102813.

Andriushchenko, M., Souly, A., Dziemian, M., Duenas, D., Lin, M., Wang, J., ... & Davies, X. (2025a). Agentharm: A benchmark for measuring harmfulness of LLM agents. In *International Conference on Learning Representations* (Vol. 2025, pp. 79185-79220).

Andriushchenko, M., Croce, F. & Flammarion, N. (2025b). Jailbreaking leading safety-aligned LLMs with simple adaptive attacks. In *International Conference on Learning Representations* (Vol. 2025, pp. 40116-40143).

Australian Cyber Security Centre. (2025). AI data security: Best Practices for Securing Data Used to Train & Operate AI Systems. Australian Government. https://www.cyber.gov.au/sites/default/files/2025-05/CSI_AI_Data_Security.pdf

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., ... & Liang, P. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Brown, C. J., & Spillias, S. (2026). Prompting large language models for quality ecological statistics. *Methods in Ecology and Evolution*, *17*(4), 1012-1021.

Brown, C. J., Aitken, L. R., Takyi, R., & Tisseaux-Navarro, A. (2026). Automating ecological and fisheries modelling with agentic AI. *Fish and Fisheries*.

Bundesamt für Sicherheit in der Informationstechnik (BSI) and Agence nationale de la sécurité des systèmes d'information (ANSSI) (2024). *AI Coding Assistants*. Bonn/Paris: BSI & ANSSI. Available at: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/KI/ANSSI_BSI_AI_Coding_Assistants.pdf (Accessed: 29 June 2026).

Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., ... & Raffel, C. (2021). Extracting training data from large language models. In *30th USENIX security symposium (USENIX Security 21)* (pp. 2633-2650).

Chakraborty, I. (2026). llmshieldr: Safety Guardrails for Large Language Model Workflows (Version 0.1.0) [Software]. GitHub. https://github.com/ineelhere/llmshieldr

Chen, B., Zhang, Z., Langrené, N., & Zhu, S. (2025). Unleashing the potential of prompt engineering for large language models. *Patterns*, *6*(6).

Cook R, Assaduzaman M, Jones S (2024). deident: Persistent Data Anonymization Pipeline. R package version 1.0.0, https://CRAN.R-project.org/package=deident.

Cooper, N., Clark, A. T., Lecomte, N., Qiao, H., & Ellison, A. M. (2024). Harnessing large language models for coding, teaching and inclusion to empower research in ecology and evolution. *Methods in Ecology and Evolution*, *15*(10), 1757-1763.

De Masi, A. (2026). Terminal Is All You Need: Design Properties for Human-AI Agent Collaboration. *arXiv preprint arXiv:2603.10664*.

Devline, H. and Burgis, T. (2026, March 14). *Confidential health records from UK BioBank project exposed online*. *The Guardian*. https://www.theguardian.com/science/2026/mar/14/confidential-health-records-exposed-online-uk-biobank (accessed 2 July 2026). Archived at: https://web.archive.org/web/20260702013807/https://www.theguardian.com/science/2026/mar/14/confidential-health-records-exposed-online-uk-biobank

Fox, N., Di Minin, E., Carter, N., Tomkins, S., & Van Berkel, D. (2025). Balancing accessibility and security: Safeguarding citizen-sourced biodiversity data in the age of AI and open-sourced software. *Ecological Informatics*, 103443.

Gallois, E. C., Salili-James, A., Poon, S. T., Trebski, A., & Redding, D. W. (2025). Fast-tracking ecological interpretation using bespoke quantitative large language models. *Methods in Ecology and Evolution*, *16*(12), 2730-2740.

Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect

prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security* (pp. 79-90).

HiddenLayer Research Team (2026). *Malware Found in Trending Hugging Face Repository "Open-OSS/privacy-filter"*. Available at: https://www.hiddenlayer.com/research/malware-found-in-trending-hugging-face-repository-open-oss-privacy-filter (accessed 29 June 2026). Archived at: https://web.archive.org/web/20260628225950/https://www.hiddenlayer.com/research/malware-found-in-trending-hugging-face-repository-open-oss-privacy-filter

Jiang, J., Wang, F., Shen, J., Kim, S., & Kim, S. (2026). A survey on large language models for code generation. *ACM Transactions on Software Engineering and Methodology*, *35*(2), 1-72.

Jilani, A. (2025). SecretSentry (Version 2.0.0) [Software]. GitHub. https://github.com/y2ee201/secretsentry

Marchand, R., Cathain, A. O., Wynne, J., Giavridis, P. M., Deverett, S., Wilkinson, J., ... & Coppock, H. (2026). Quantifying frontier llm capabilities for container sandbox escape. *arXiv preprint arXiv:2603.02277*.

National Security Agency (NSA) (2024). Deploying AI Systems Securely: Best Practices for Deploying Secure and Resilient AI Systems. https://media.defense.gov/2024/Apr/15/2003439257/-1/-1/0/CSI-DEPLOYING-AI-SYSTEMS-SECURELY.PDF

Nowok B., Raab G. M., Dibben C. (2016). synthpop: Bespoke Creation of Synthetic Data in R. Journal of Statistical Software, 74(11), 1-26. doi:10.18637/jss.v074.i11

NVIDIA (2026). *OpenShell: Safe, private runtime for autonomous AI agents*. Available at: https://github.com/NVIDIA/OpenShell (Accessed: 29 June 2026).

OpenAI. (2026b). OpenAI and Hugging Face partner to address security incident during model evaluation. Available at: https://openai.com/index/hugging-face-model-evaluation-security-incident/ (Accessed: 11 August 2026).

OpenAI. (2026b). Privacy Filter [Software]. GitHub. https://github.com/openai/privacy-filter

Posit Team (2025a). RStudio: Integrated Development Environment for R. Posit Software, PBC, Boston, MA. URL http://www.posit.co/.

Posit Team (2025b). Positron. Posit Software, PBC, Boston, MA. URL https://positron.posit.co/

Rafiq, K., Beery, S., Palmer, M. S., Harchaoui, Z., & Abrahms, B. (2025). Generative AI as a tool to accelerate the field of ecology. *Nature Ecology & Evolution*, *9*(3), 378-385.

Ruan, Y., Dong, H., Wang, A., Pitis, S., Zhou, Y., Ba, J., ... & Hashimoto, T. (2024, May). Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations* (Vol. 2024, pp. 27031-27098).

Shayegani, E., Mamun, M. A. A., Fu, Y., Zaree, P., Dong, Y., & Abu-Ghazaleh, N. (2023). Survey of vulnerabilities in large language models revealed by adversarial attacks. *arXiv preprint arXiv:2310.10844*.

Spillias, S., Trebilco, R., Adams, M. P., Boschetti, F., Constable, A., Dunstan, P., ... & Fulton, E. A. (2026a). The future of artificial intelligence in ecosystem modeling. *BioScience*, *76*(1), 57-70.

Spillias, S., Rogers, J. G., Boschetti, F., Fulton, E. A., Guglielmo, M., Yong, S. Y., & Trebilco, R. (2026b). Data-driven discovery of mechanistic ecosystem models with LLMs. *Methods in Ecology and Evolution*, *17*(4), 1303-1321.

Sun, A. Y., Zemour, E., Saxena, A., Vaidyanathan, U., Lin, E., Lau, C., & Mugunthan, V. (2023). Does fine-tuning GPT-3 with the OpenAI API leak personally-identifiable information?. *arXiv preprint arXiv:2307.16382*.

Tang, X., Jin, Q., Zhu, K., Yuan, T., Zhang, Y., Zhou, W., ... & Gerstein, M. (2025). Risks of AI scientists: prioritizing safeguarding over autonomy. *Nature Communications*, *16*(1), 8317.

Tian, Y., Yang, X., Zhang, J., Dong, Y., & Su, H. (2023). Evil geniuses: Delving into the safety of LLM-based agents. *arXiv preprint arXiv:2311.11855*.

Tomasic, N. (2023). Balancing privacy with access to information for commercial fisheries data: A critical review of Fisheries and Oceans Canada's "rule of five" policy. *Facets*, *8*(1), 1-11.

Tulloch, A. I., Auerbach, N., Avery-Gomm, S., Bayraktarov, E., Butt, N., Dickman, C. R., ... & Watson, J. E. (2018). A decision tree for assessing the risks and benefits of publishing biodiversity data. *Nature Ecology & Evolution*, *2*(8), 1209-1217.

Wang, K., Wang, T., Wang, B., Li, Q., Liu, Y., Han, Q., ... & Ren, Y. (2026). Large Language Models (LLMs) for fisheries management research: understanding potential and navigating risks. *Reviews in Fish Biology and Fisheries*, *36*(1), 56.

Supporting Information

A growing ecosystem of R packages includes AI functionality that can transmit data to external servers. In **Table S1** we catalogue the most widely used packages with their data transmission behaviour. Of particular concern for analysis of confidential data are packages like *chattr*, which automatically enriches prompts with data frame names and structures from the R environment, and *mall*, which is designed to process data frame columns through an LLM and thus transmit actual data values.

In **Table S2** we summarise the AI tools available in the three IDEs most commonly used by environmental research scientists (RStudio, Positron, VS Code). The landscape of IDE-embedded AI tools is changing rapidly, and the AI risk landscape in a typical researcher's IDE is substantially more complex in 2026 than it was in 2024. A key limitation of automated protection (e.g. by *confideR*) is that IDE-level AI features are managed by the IDE itself and are largely opaque to R-based tools. For RStudio, the *rstudioapi* package allows partial inspection of settings. For Positron and VSCode, *confideR* can detect only indirect signs (environment variables, loaded R packages). This makes manual verification especially important.

Table S1. Some R packages that include AI features.

Package	What it does	What it sends externally	Risk level
`ellmer`	Core R interface to LLMs (Claude, GPT, Gemini, Ollama)	Whatever you include in prompts; tool calls can access R environment	Depends on use
`chattr`	Chat interface in RStudio Viewer; auto-enriches prompts	Prompt text + data frame names/structures + file paths	Medium–high
`gptstudio`	RStudio add-in for chat, code commenting	Selected code and prompts	Medium
`gander`	Inline code suggestions via keyboard shortcut	Code context around cursor	Medium
`mall`	Row-wise LLM operations on data frame columns	Actual data values from your data frame	High
`btw`	Assembles R session context for LLM prompts	Data frame descriptions, package info, session state	Medium
`side`	Full AI assistant in RStudio sidebar; reads/writes files	Broad access to files and R environment	High
`rollama` / `ollamar`	Interface to local models via Ollama	Nothing, all processing is local	Low

Table S2. Comparison of some AI coding assistant tools available in RStudio, Positron, and VS Code, categorised by the scope of assistance and data transmitted to external servers, and some notes on privacy protections. 'Access scope' is numbered to indicate extent of access (1 = least access; reads currently open file; 8 = most access; agent can read/write files and run shell commands). Note that the same tool (e.g. GitHub Copilot) can behave differently depending on the IDE. Also note that a tool's features and access often change and are sometimes opaque.

Tool	Type	Access scope	Activation	Notes
RStudio				
GitHub Copilot	Chat and autocomplete	2: Active file and metadata	Activate in Global Options; once activated will start on open	Enterprise tier: no-training policy; detectable by *confideR*
Posit Next Edit Suggestions	Autocomplete	2: Active file and metadata	Once enabled always operates in background	Paid subscription; Zero Data Retention (ZDR) agreement with Anthropic; runs through managed Posit AI service
Posit Assistant	Chat and Agent	5: Session and data contents	Once enabled is loaded in the background every session	Same as above
Positron				
GitHub Copilot	Chat and autocomplete	4: Session and data structures	Enabled by default if signed in; see Extensions > Settings	Enterprise tier: no-training policy; partially detectable by *confideR*; higher risk than RStudio due to broader context
Posit Assistant	Chat, autocomplete, and Agent	5: Session and data contents 7: Autonomous in session execution	Extension must be installed; once enabled is loaded in the background every session	Runs through managed Posit AI service; replaces Positron Assistant and Databot
VS Code				

GitHub Copilot	Chat and autocomplete and Agent	8: Autonomous filesystem and shell (agent mode)	Enabled by default when signed in; configurable per workspace	Enterprise tier offers some protection
Claude Code	Chat and Agent (CLI)	8: Autonomous filesystem and shell	Manual; invoked per task	Default API: 30-day data retention, opt-out of training available; Enterprise API: ZDR available
Continue.dev	Chat and autocomplete and Agent	1: Active file to 6: Full codebase – configurable	Manual; configure model in settings	Bring your own API key; can run local models which offer ultimate privacy
Codex	Chat and Agent (CLI)	8: Autonomous filesystem and shell	Manual; invoked per task	OpenAI API policies
Codeium	Chat and autocomplete	3: Session context	Install extension; activate once	Enterprise tier with data residency options; free tier has limited contractual protections
Cursor	Chat and autocomplete and Agent (VS Code fork)	8: Autonomous filesystem and shell	Install as separate application; enabled by default	Privacy Mode (code not stored on servers)

Table S3. Open-weight model classes, the hardware required to run them locally, the usable context window in that memory, and coding capability on a single common benchmark, SWE-bench Verified (% of 500 real GitHub issues resolved; accessed 13 August 2026). Weight storage is approximate at 4-bit quantisation (about 0.5 bytes per parameter); the context column is the window that fits in the remaining memory with the key-value cache at reduced precision. Scores: [I] independent multi-model harness (vals.ai); [L] official SWE-bench leaderboard submission; [V] vendor-reported; [R] community reproduction. Note that scores do not rise monotonically with total parameters: the 30-billion dense models here match or beat the 120-billion mixture-of-experts models, which activate only 10–13 billion parameters per token. Named models are examples of their class, not recommendations, and entries will shift within months.

Model class	**Weights at 4-bit**	**Hardware**	**Usable context (approx.)**	**Example model Scores – SWE-bench Verified (%)**
Small (9–30B total, ≤4B active)	~5–16 GB	Laptop or 16 GB GPU	~32–64K tokens	GLM-4.7-Flash (30B-A3B): 59.2 [V] GPT-OSS-20B: 34.0 [R]
~30B dense	~15 GB	Single workstation GPU, 24–32 GB	~128K tokens (measured: 131K in 32 GB)	Meta Muse Glimmer 30B: 76.0 [L] Qwen3.5-27B: 72.4 [L]
~120B MoE (10–13B active)	~60 GB	Unified-memory desktop, 128 GB (e.g. NVIDIA DGX Spark)	~256K tokens (measured: 262K in 121 GB)	Qwen3.5-122B-A10B: 72.0 [L] NVIDIA Nemotron 3 Super (120B-A12B): 60.5 [V] poolside Laguna S 2.1 (118B-A8B): not published
Very large MoE (700B–2.8T)	~0.4–1.4 TB	Multi-GPU server or rack; not a deskside option	Model maximum (256K–1M); memory is not the binding constraint	Moonshot Kimi K3 (2.8T-A104B): 93.4 [I] DeepSeek V4 Pro: 80.6 [V]
Frontier proprietary (reference)	n/a (cloud only)	Provider infrastructure	Provider-set (200K–1M)	GPT-5.6 Sol: 96.2 [I] Claude Fable 5: 95.0 [I]