Ingence: An indexing and inference engine that relates data organization to experimental structure

v0.4.0 Technical Report

Andrew Bazley
Institute for Systems Genetics, NYU Langone Health, New York, NY 10016

Abstract

Large research data collections are hard to explore by hand, especially without knowledge of their structure or contents. When researchers leave a lab, or when organization and annotation fail, the data must speak for themselves. To help make sense of such collections, we present Ingence v0.4.0: a read-only, indexing and inference engine and Dashboard that reads research data collections and proposes how its files relate. Ingence scans research data collections, reading files' names, metadata, and parts of their contents. A Rulebook of logical rules, common to most research practice, then infers which folders are experiments, when each was done, their conditions and repeats, and how files relate within and between experiments. Every result names its rule and evidence, the engine gives no result without supporting evidence, and corrections a user makes take precedence over the engine's assumptions. Ingence's design centers research data security and privacy. Because of risks and limitations associated with LLM agents interfacing with institutional data storage, and their inaccessible model logic, AI isn't required to use Ingence, and it doesn't use AI to produce its results. It contains a model context protocol for agentic use and outputs agent-friendly summaries, at a detail level the user chooses. We tested Ingence on an invented laboratory, whose generator writes the ground truth, and on three public archives: the BioImage Archive, OpenNeuro and ENCODE. In the invented laboratory, Ingence found copies, dates and the stages of files reliably. In the archives, it found the structure that file names state plainly and missed structure that follows conventions its rules do not yet read. When it proposed that one file was made from another, it never reversed the direction the depositor declared.

Introduction

A laboratory's data are the foundation of its research. Most laboratories leave the management and organization of those data to individual members, and over the years servers and drives fill with raw output, processed files, analysis tables, figures, scripts, notes and miscellaneous documents. Organization habits vary, but there is a shared expectation that how files are named and where they sit carries meaning. Reading that meaning back out of a filesystem requires familiarity with the depositor's abbreviations, names and wording. Lab notebooks and tables of contents can give a reader a high-level view, but a large filesystem quickly outpaces any lab notebook or hand-written notes. How files relate to one another is a separate question from what they are and where they sit. Names and arrangement suggest an experimental context, but until the data are published, anyone other than the depositor has to piece together by hand what science a collection describes as a whole. Public archives and data repositories face a similar problem at a larger scale, since each depositor organizes and annotates a deposit at their own discretion. The most reliable guide to a researcher's data is the researcher's own memory, but when people leave laboratories, this context becomes inaccessible. Years can pass without touching legacy data and, while new results take the lab's attention, it stagnates, wasting the resources it cost to generate and losing progress to repetitive work.

The questions scientists ask when they return to a large collection are about the collection, not the single file. Which experiment does this belong to? What was it made from? Where was it repeated? Which raw data were followed by an analysis we can find, and which were not? Many tools answer where a file is, and a smaller number what is inside it. Few say what a collection of files means taken together, or how they reached that reading. This report describes

Ingence v0.4.0: an indexing and logical inferencing engine that aims to address these questions, reading research data collections in a way that a person cannot. It observes what files are, where they sit, how and when they were created, and what similarities exist among files and the words that describe them. After Ingence reads a data collection, a set of auditable rules that the user can edit suggests how the findings are related, interpreting meaning from a vast set of *in situ* observations (Figure 1). Each result traces back to the rule and the evidence that produced it, and where the evidence is not enough, the engine gives no result and says why. Ingence keeps its interpretation in a Dictionary of word meanings and a Rulebook of rules, stored as plain text apart from the engine's code, so a person can read, version and change how it reads a collection. Its Dashboard organizes a collection by experiment, shows how complete each experiment is and where it was repeated, and lets a person correct one kind of assumption at a time. We evaluated it against ground truth that the engine never reads. Ingence maps a collection: it shows what is there and how experiments connect, and it makes no claim about the science in the files.

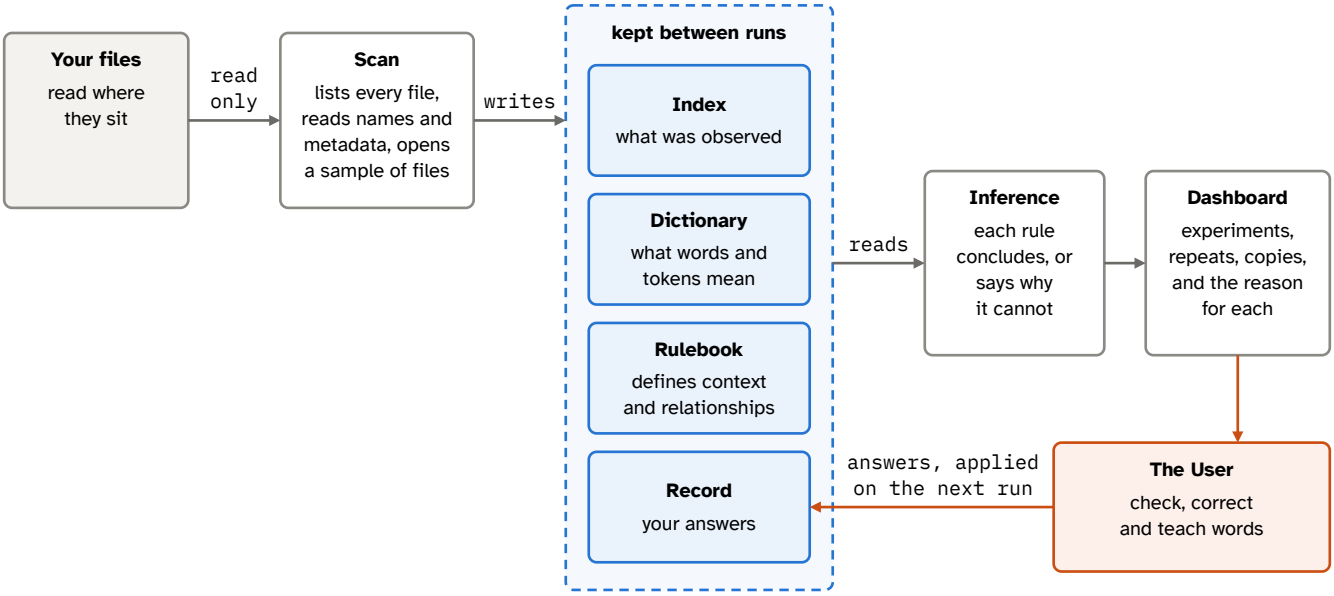


Figure 1. How Ingence reads a data collection and applies a person's answers at the next run. The scan reads a data collection where it sits and writes what it observes to the Index. Inference applies the rules of the Rulebook to the Index, reads words through the Dictionary, and includes a person's earlier answers from the Record. The Dashboard shows the results, and the answers a person gives there go to the Record and take effect at the next run.

Today, AI models and agents can explore a filesystem on a person's behalf: they list folders, open files and summarize what they find. For research data, agents come with two main risks: policy and integrity. Many research institutions restrict or forbid automated agents on their servers and shared storage, through IT security rules, data-governance policies and agreements that cover sensitive data. Some institutions have yet to formalize AI policy, exposing researchers to the risk and liability of what such agents may do. The collections people most need help understanding are therefore often the ones their own agents are not supposed to touch. Agent integrity is more abstract, but applies wherever an agent is allowed, from a laboratory server to a public data repository. An agent that can run commands independently has no built-in guarantee that it will only read. Even well-intentioned, carefully prompted agents can and do write, move, rename or delete what they encounter, and the change can be silent or impossible to undo. Reported incidents include a coding agent that deleted a live production database despite instructions not to change it (OECD.AI 2025b) and a command-line agent that deleted a user's files (OECD.AI 2025a). Perhaps most notable was the July 2026 incident in which about 1,200 OpenAI agents escaped their isolated test environment and compromised parts of Hugging Face's production infrastructure (OpenAI 2026; Greenblatt, Cotra, and Wijk 2026). This example shows the potential for agentic server abuse and is a reminder that even the most advanced AI developers are vulnerable to unforeseen activity and unknown consequences. On a research server, even a small, silent change to a raw file can compromise the record that a published result rests on, and nobody may notice until much later. None of this argues against agents, which can be

very helpful. It argues for limiting how they interact with original data, and suggests that the best AI safeguards may not involve AI at all.

Ingence offers a protective layer between an agent and the data. Ingence itself is a fixed, auditable program: it reads a collection in place and read-only, writes only into its own project folder, uses no language model and uses the network only when asked, so an institution can review and approve it like any other software, whereas an agent's actions differ from one session to the next. Once Ingence has read a collection, it can give a person's own agent most of what the agent would have learned by exploring the files itself: which experiments the collection holds, when each was done, how complete each is, which repeat and how they overlap, and the evidence behind each result. The person chooses how much the agent sees, from counts alone to file paths, through a written summary or a read-only query interface (see "Questions, and AI agents" below). The agent can then help the person interpret and correct what Ingence found, but it never touches the laboratory's server, and its suggestions enter the project only as answers the person has reviewed.

Tools similar to Ingence differ mainly in what they ask of their users before they can help. Provenance captured at the time of work is precise for the work it covers: provenance-aware storage (Muniswamy-Reddy et al. 2006) records how files are made at the level of the operating system, and noWorkflow (Murta et al. 2015) does the same for scripts. Groth, Gil and Magliacane called the task of working out, after the fact, which data were made from which "provenance reconstruction" (Groth, Gil, and Magliacane 2012). Ingence does this with explicit rules and leaves each proposal for a person to confirm. Research data management systems such as OMERO (Allan et al. 2012) and DataLad (Halchenko et al. 2021) make structure explicit once a person imports data or places them under version control. Ingence reads data where they are, and could give such systems a first map of a collection.

Community conventions are the other means to encode readable structure alongside data storage. The Brain Imaging Data Structure (BIDS) (Gorgolewski et al. 2016) made neuroimaging names machine-readable, and the converters into BIDS are Ingence's closest relatives in method: ezBIDS (Levitas et al. 2024) infers subjects, sessions and series from paths and headers with explicit heuristics a person can correct, and BIDScoin (Zwiers, Moia, and Oostenveld 2022) maps source files through an editable set of rules. Both end by writing a converted copy of the data, whereas Ingence ends with results about the original files and needs no predetermined convention. At larger scale, Xtract (Skluzacek et al. 2021) spreads metadata extractors across machines, and its authors report that users disagree about which metadata are useful, that standards drift, and that extractors are costly to maintain (Skluzacek, Chard, and Foster 2022), all arguments for rules their users can version and edit. Data-lake systems such as Goods (Halevy et al. 2016) and R2D2 (Shah et al. 2023) catalog an organization's tables and find duplicated or contained datasets. These systems describe files or tables; Ingence relates files to one another and to experiments. Discourse graphs (Chan et al. 2024) give researchers a means of organizing their hypotheses and conclusions through a graph of relations among claims and evidence. Ingence shares their commitments to local use, typed relations and gradual formalization, but extracts this meaning from existing data collections rather than from user input. Table 1 compares Ingence with these systems from each tool's documentation or publication, and Supplement S2 gives the source of each entry.

Table 1. Capabilities of Ingence compared to similar tools. "no*" means the tool draws no conclusions of its own, so the row does not apply.

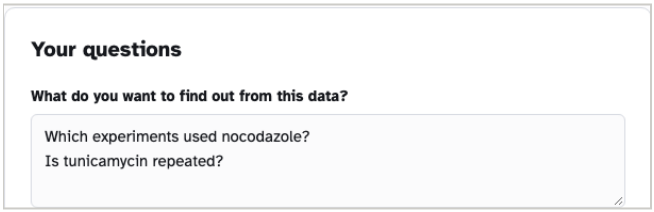
Property	Ingence	ezBIDS	BIDScoin	DataLad	OMERO	Xtract	Goods	R2D2
Works on data already stored	yes	yes	yes	partly	yes	yes	yes	yes
Leaves data in place, unchanged	yes	no	no	no	partly	partly	yes	yes
Relates files to one another	yes	partly	partly	yes	partly	partly	yes	yes
Groups files into experiments	yes	partly	partly	no	no	partly	partly	no
Shows the reason for each result	yes	partly	partly	no*	no*	not stated	partly	not stated
Says when it cannot decide	yes	yes	yes	no*	no*	not stated	partly	no
Keeps corrections for later runs	yes	partly	yes	no*	no*	not stated	partly	not stated
Runs locally, without a network	yes	partly	yes	yes	partly	partly	no	no
Uses trained models	optional	no	no	no	no	yes	partly	no

Using Ingence

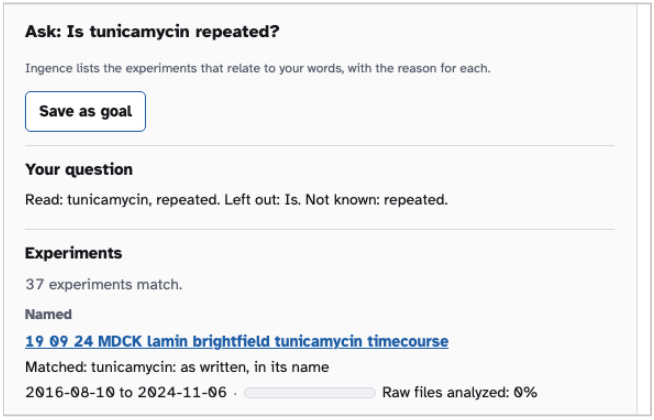
Ingence was designed to be used on a local dashboard, but can also be run entirely from the command line. Typing ingence opens the Dashboard in a browser on the user's own computer, served by Ingence at a local IP address that other computers cannot reach (Figure 2). The Dashboard's landing page asks for a directory path, a place to save the project and an experimental data range, and offers three reading depths: a *quick* read only scans names, sizes and dates, while *standard* (the default) also reads a sample of each file type's contents, and *deep* reads much more of each file, within a set time limit. The person can type the questions they want answered into the question box (Figure 2a), and during the scan the page shows each step and the experiments that match each question so far. Later, the same box at the top of every page takes a new question, such as "which experiments used DAPI", and returns the related experiments with the reason for each (Figure 2b). When the scan ends, the Dashboard offers up to five quick checks, each a question that already carries Ingence's guess, which the person can keep, change or skip.

The **Overview** holds the saved questions with their experiments, a summary of the project, its areas (by default, its top-level folders), its completeness (the raw files for which Ingence found an analysis), its repeats, its main themes, and **What Ingence assumed**: up to eight kinds of assumption, each with a count, two examples and an option to refute the assumption (Figure 2f). **Overlap** draws each experiment as a dot and joins two experiments that share words, follow the same workflow, use the same instrument, repeat one another, hold a copied file, or hold a file made from a file of the other; selecting a line shows its evidence. **Experiments** lists every experiment with its dates, its repeats and a bar from raw data to analysis, and opens each one's story, day by day, with the workflow Ingence suggests for it, such as ND2, then TIFF, then CSV (Figure 2c). A person corrects results where they appear: **This is wrong** records one answer for every assumption of that kind, a person can change or turn off a suggested workflow, and the answers take effect at the next **Re-run**. Words in bold name pages and buttons of the Dashboard.

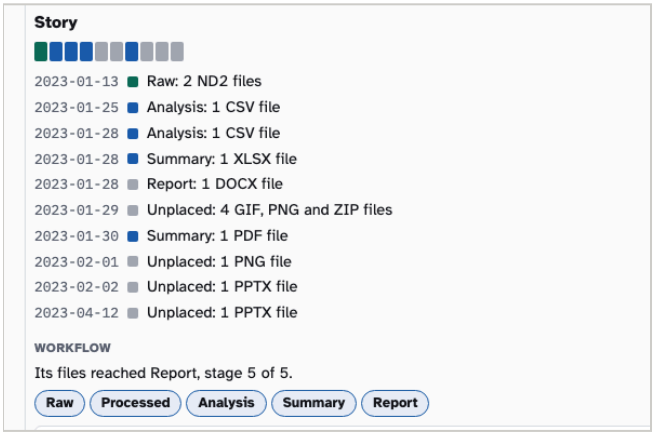
a The question box



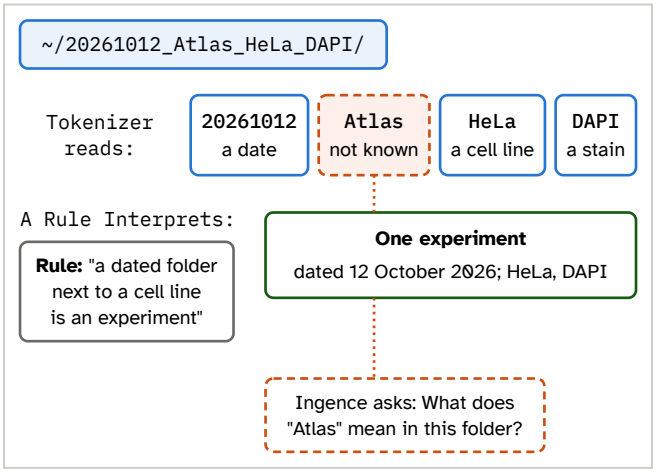
b An answer, with the reason for each match



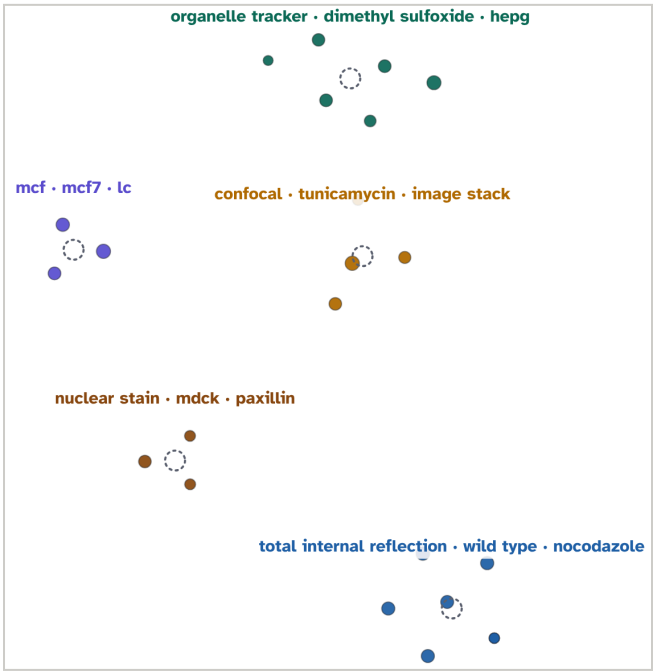
c One experiment's files over time, and its workflow



d How a folder name becomes a result



e Experiments grouped into themes by shared words



f What Ingence assumed, with a button to correct it



Figure 2. Dashboard screenshots using invented laboratory names. (a) The question box on the landing page. (b) The answer to one question, with the reason each experiment matched. (c) One experiment's story: its files by date and stage, and the workflow Ingence suggests for it. (d) How Ingence reads one folder name: each part is looked up in the Dictionary, a written rule reads the parts as an experiment, and a word the Dictionary does not know stays unknown. (e) The Map of groups: experiments placed by the words they share, with each theme labeled by its most distinctive words. (f) What Ingence assumed, with its examples and the button that corrects every assumption of that kind.

How Ingence works

Principles

Source data are read-only: Ingence writes only into its project, and it reads scripts as text without running them. Observations and interpretations are intentionally separate: an interpretation requires observations, but neither creates the other. Every result can be rebuilt from four inputs (Figure 1): the Index of observations, the Rulebook, the Dictionary and the Record, an append-only log of a person's answers. Each scan also records its settings and engine version. "No result" is an output, and Ingence keeps three such states apart: unknown, not yet read, and conflicting. Rules are the basis of every result, and every inference or conclusion, including low-confidence guesses, is attributable to the rule that formed it. More evidence can add an exception that shows a result as conflicting, but it cannot lower the tier of the rule that made the result. The engine runs on the user's computer and uses the network only when asked, for example to fetch a public archive's file list.

Ingence v0.4.0 adds a working principle: assume, show and correct. Where the evidence supports a result, a rule takes it and marks it as an assumption, and one answer corrects every assumption of that kind. An assumption is an ordinary result, made by a named rule from stated evidence at its own tier, and where there is no evidence the engine still gives no result. This principle came from development testing, where users asked for fewer questions. The Dashboard groups similar assumptions so a person can review them with examples instead of answering the same question repeatedly. The methods reference at ingence.org describes each step, rule and setting.

The scan

A scan begins with the census, which lists the files. Ingence records every folder and file with its name, size, kind and times, and assigns each file to a series by its folder, the shape of its name, its type and its size class. When Ingence cannot list a folder, it records the folder with the reason. File enumeration during the census preserves ordering and placement, and when a series is detected (from file name labeling), files are added to the census in order. Each scan writes its results into a new folder of its own, checks them, and uses the most recent scan. An interrupted scan never replaces its predecessor until it has successfully finished, and Ingence never edits an earlier scan.

Names, the tokenizer and the Dictionary

After the scan completes the census, the tokenizer (version 14) reads every file and folder name, as well as any research question a user types into the question box on the landing page. It cuts a name into words, numbers, quantities with units, dates, indexed labels such as r2, key-value labels such as sub-01, and identifiers such as accession numbers. In HEPES_buffer_500nM_2.5h_r2.csv, it finds two words, a concentration, a duration and a label, and it reduces quantities through their units, so 500nM equals 0.5uM. Most Dictionary terms include abbreviations, acronyms and word variations. The tokenizer flags a possible misspelling when a word of five or more letters is one edit away from a Dictionary spelling, or two edits for a word of seven or more letters, where an edit adds, removes or changes one letter or swaps two neighboring letters. It does not correct units, dates, or words outside of the Dictionary (Figure 2d).

The Dictionary says what a word can mean and offers some context around it. It is a set of meanings and word relations kept in YAML files, apart from the engine's code. Each meaning has a term, a kind (technique, reagent, instrument, unit, stage and so on) and its spellings. Ingence v0.4.0 holds 874 meanings with 3,831 spellings, organized into 15 general sections, such as connecting words, instruments, reagents and units, and eight fields: biochemistry, clinical, computational, electrophysiology, microscopy, programs, sequencing and spectrometry. Where one spelling has different meanings in two fields, the files around it choose: Ingence counts the file types and field-specific words in the nearest folder that holds at least 20 such files, and a field decides when it holds at least 60% of them and three times as many as the next field. Otherwise, the engine defaults to the user's specified field or, if the user named none, to the word's general meaning, marked as an assumption; failing both, the word stays unresolved. We encourage users to share their

Dictionary words, on the Ingence GitHub and especially within a laboratory or among groups whose work shares common terms. A person can export the words they taught Ingence as a pack, a small file of Dictionary words (ingence teach export-answers), and another person can install one or many packs into a project (ingence teach pack install); Ingence reports any spelling a pack shares with the included words, so a person can see how each tie is settled.

Evidence from contents

Files can provide evidence in addition to their names: metadata, the file type, when it was modified or created, whether the file might be raw data from an instrument, and which stage of the experiment it might belong to (raw, processed, analyzed, summarized or reported, depending on the collection's workflows). A table of file types, separate from the Dictionary, categorizes file types and suggests context for their origin. Ingence records how and where it made each observation from a file's bytes, and in some cases these observations supersede what a file's name suggests (Ingence reads an image saved as .csv as an image). This lets the engine place a TIFF file written by acquisition software in the recording stage and one written by analysis software at the processed stage, and read one written by a drawing program as a figure. For ND2, LIF and LI-FLIM files, the readers take the acquisition time from the file's own header. A FASTQ file named .fastq or .fq counts as recorded data, because a sequencer writes it; Ingence does not yet read the compressed spellings (.fastq.gz, .fq.gz) this way, which we return to in the Evaluation.

When names and metadata do not tokenize into observations that support inference, or when files carry uninformative names or look like duplicates, Ingence takes a deeper look at a file by reading the bytes it contains. The dive opens files read-only and reads what each format exposes: an instrument's header and acquisition clock, a TIFF's tags and first page, an Office document's properties, a PDF's metadata, a table's column names, a text file's encoding and frequent words, the members of an archive, and the read and write calls a script names. Ingence reads file contents into memory and never writes them anywhere, and it reads scripts and notebooks without running them. The dive runs on a pool of worker processes that read files' contents. One process hands out the work in a fixed order and writes every result, which keeps reading fast and keeps the results the same for any number of workers. Time limits keep a first scan short: at Standard, the dive spends at most 900 seconds of reading and reads at most 4 MiB of any file, and at Deep it reads up to 64 MiB of a file and stops after four hours. Ingence marks whatever a limit leaves as not yet read. On a laptop with 16 GiB of memory and 8 CPU cores, a deep scan of 9,021 files took about four minutes from start to finish, and the standard scan depth of 88,019 files about 24 minutes. The scan speed depends mostly on the computer running Ingence and, for storage reached over a network, on the connection's speed and bandwidth.

Rules and relations

A rule is a unit of logic that defines what files and folders are, how to interpret the engine's observations, and how these components relate to each other in the context of experimental research. Each rule is stored with a name, version, description, and its parameters and exceptions, alongside the Python function the engine uses to evaluate and apply it. Revisions of the Rulebook are append-only, so Ingence can rebuild an earlier scan. Ingence v0.4.0 (Rulebook version 5) uses 106 active rules, with 19 placeholders planned for a future update. Each time a rule is applied to a file or folder, Ingence records one of five outcomes: the rule reached a conclusion; it found nothing to read; it could not read its input, such as a damaged file or a format with no reader (a comprehension failure); its input was never collected, such as a file the dive did not reach (not yet read); or its evidence pointed more than one way (an interpretation failure). The first failure calls for a better reader and the second for a person's decision, so Ingence keeps them apart.

Rules belong to several logical categories that decide when they apply. Thirteen rules are entailed, meaning they are applied whenever their input observations are true, as with two equal whole-file hashes. There are 105 defeasible rules, which hold by default and are withdrawn when an exception concludes against them, and seven are abductive, proposing the best explanation of what was seen. Ingence v0.4.0 relies on abductive assumptions to reduce the need for user input: what wrote a TIFF file, the Dictionary field a folder belongs to, a corrected spelling, or a workflow's step, for example.

A person's answer is a fourth kind, called testimony, which applies only to the files or folders relevant to the person's answer. In v0.4.0, user answers act through the Record: an answer about a word becomes a Dictionary meaning that wins over the included ones, and correcting a rule's claim automatically propagates the correction across that rule's similar conclusions. A person who says a rule read one file wrongly answers for every file the rule read the same way.

Ingence stores its results as a graph: files, folders and experiments are its nodes, and each claim is an edge between two of them. The engine builds everything it asserts between two observations from eight relations (Figure 3a). The approach follows Winston, Chaffin and Herrmann, who told apart the meanings of the English phrase "a part of" by a few formal properties (Winston, Chaffin, and Herrmann 1987); Ingence tells its eight relations apart the same way. Ingence has no generic "related to", which would describe most relationships but carry little meaning. Seven properties declare each relation, among them whether it is directed or transitive and whether a fact may pass along it, and no two relations share all seven. Because every rule concludes with one of these relations, and each relation's properties are declared once, the engine stores and checks every claim by its relation rather than by the rule that made it. A new rule that reuses an existing function can therefore join the Rulebook as data, a sentence and parameters, without changes to the engine's code; only a rule that reads a new kind of evidence needs a new function. Figure 3b shows each relation on a small example, and Supplement S1 gives all eight formally, as they appear in v0.4.0.

"Related to" also says nothing useful about two identical files, and finding duplicates is a large part of what Ingence does. Ingence hashes only what a comparison needs to resolve. It computes a quick digest of the first and last 64 KiB of each file that shares its size with another, and it reads whole every file whose quick digest another shares, using the BLAKE3 hashing algorithm to form a digest of its bytes (O'Connor et al. 2021). Equal whole-file digests make two files identical, the one relation that lets facts about one file stand for another. Where an archive publishes a checksum for each file, Ingence reads it as declared evidence of identity. Where several basic observations agree, such as name, size, modification time and kind, Ingence proposes the files as copies for a person to confirm.

Stages, workflows and file origins

A file's stage is its place in the experimental workflow: recorded, processed or analyzed. Rules read it from an instrument's format, from the program a file's tags say wrote it, and from words in its name such as raw or processed. Once inference has read stages, Ingence counts each experiment's chain of stages and the file types at each step, and a chain seen in at least two experiments becomes a suggested workflow. A later rule places a file without a stage at the workflow step that lists its type, folder or name. When one file or folder could be in more than one stage, it is placed at none.

Workflows vary between laboratories, between people and between projects, so Ingence does not impose a fixed list of them. It suggests workflows from the collection's own files, and each suggestion stands as an assumption until the person changes it. On the Dashboard, a person can open any suggested workflow and change its steps or turn off the suggestion, and Ingence first shows what the change would do. A project's settings can also switch workflow suggestion off entirely.

Ingence also proposes which file was made from which, such as a table of measurements made from a movie, with the relation derives (Figure 3). Separate rules each read one kind of evidence: a script's read and write calls; a file that carries another file's name; one name repeated in side-by-side folders whose names give the roles of source and result; a file that a describing document names; a summary of a series; a time copied from a recording beside it; and the same file name with words for a later step added. The evidence also gives the direction: which file is the source and which was made from it. Each proposal covers one step between two files. Several steps across one workflow give a file's provenance, the sequence of steps that produced it. Ingence proposes such chains for a person to review, and its consistency tests reject any chain that loops back on itself.

a Eight relations connect files, folders and experiments

		Directed	Symmetric	Transitive	Acyclic	Series-dependent	Exhaustive	Substitutive
`contains`	this folder holds that entry	●	○	●	●	no	●	○
`embeds`	opening this file yields that one	●	○	●	●	no	○	○
`derives`	that was made from this	●	○	○ †	●	no	○	○
`identical`	the same bytes	○	●	●	—	no	○	●
`equivalent`	the same in a stated respect R	○	●	● ‡	—	no	○	○
`succeeds`	this comes after that in a series	●	○	●	●	yes	○	○
`references`	this names or points at that	●	○	○	○	no	○	○
`concerns`	both are about the same subject	○	●	○	—	no	○	○

† A `derives` edge claims one step only; a file's full provenance is found by following edges, not stored as one.
‡ `equivalent` is transitive only within a single respect R; sameness in one respect implies nothing in another.

b Example relationships

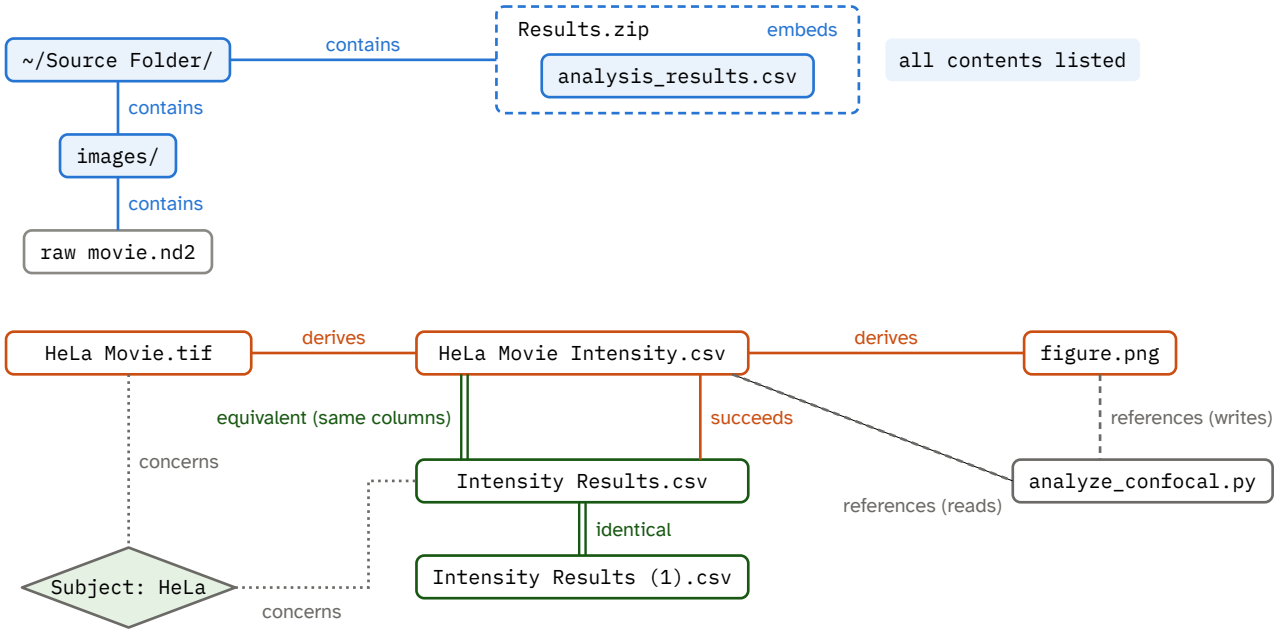


Figure 3. The eight relations Ingence uses to connect files, folders and experiments. (a) Each relation with its meaning and its seven formal properties; filled circles mark a property the relation has, open circles one it lacks, and a dash a property that does not apply. (b) An illustrative example of each relation among a few files: a folder contains files, an archive embeds a table, a table is derived from a movie, two copies are identical, and a script references the files it reads and writes.

Experiments, repeats, overlap and themes

Two rules propose which folders are experiments: one reads a folder whose name holds a date beside other words (Figure 2d), and the other a folder that holds an instrument's raw files together with files named after them. An experiment is complete when Ingence found an analysis for its raw files, and otherwise partial, raw only or analysis only. Each experiment carries a signature formed by its folders' descriptive words and the file types it contains. Ingence builds three views from these experiments. A *repeat* is a pair of experiments with the same signature that began on different

days, and the Experiments page lists each experiment's repeats. A line joins two experiments on the Overlap page and carries every kind of overlap the pair shares: words, a workflow, an instrument, a repeat, a copied file, or a file made from a file of the other. Its weight combines them, and rarer evidence counts more (Spärck Jones 1972).

A theme, the third view, is a group of experiments that share vocabulary (Figure 2e). Ingence weights each experiment's words so that rare, distinctive words count more than common ones such as date codes or file extensions, and condenses these weights so that experiments with similar vocabularies lie near one another (Deerwester et al. 1990). It then builds a second graph, separate from the graph of claims: its nodes are experiments, and an edge joins an experiment to its nearest neighbors where they are closer than chance would make them. A standard method for finding communities in a network, the Louvain method (Blondel et al. 2008), divides this graph into groups of experiments that are more connected to each other than to the rest. Ingence splits each group into its connected parts and labels it by its most distinctive words, as the **Map of groups** page (under **Advanced**) shows. Because the method makes random choices, Ingence runs it from eight seeds. A seed is a fixed starting number for those choices: the same seed always gives the same answer, and different seeds show how much the answer depends on chance. This approach also takes into account the level at which groupings are suggested, through the groupings' *resolution*, a parameter that sets the scale of the groups it looks for (Reichardt and Bornholdt 2006): a low resolution favors a few broad themes, and a high one many narrow themes. Ingence tries seven resolutions, from 0.3 to 3, and measures at each how well the eight seeds agree, using adjusted mutual information (Vinh, Epps, and Bailey 2010). This measure compares two ways of dividing the same experiments into groups; it is 1 when they are identical and near 0 when they agree no more than chance would. A fixed rule keeps the resolution whose seeds agree most, among those that give a readable number of themes. Seed agreement measures stability, not correctness: it tells a person how firmly the collection's words support a theme, not whether the theme is right.

Questions, and AI agents

Ingence generates many inferences, and their number grows with the size of the data collection. Exploring these inferences must follow the user's research goals, or the engine loses its use. To focus what the Dashboard shows and asks the user, we built a question box, where users type plain English questions about their data collection, or keywords they would like to explore within it. The question box reads user input with the same tokenizer and Dictionary as every name, and widens a class word to its members, for example a nuclear stain to DAPI and Hoechst. An experiment is **Named** when its names hold every word of the question, and it is in the **Same theme** when it shares a theme with at least two named experiments. Each answer shows the words it matched and how, so a person can judge every match at a glance (Figure 2b). Because the question box reads questions through the Dictionary, it answers best where the Dictionary knows a collection's words, and it will improve as people share packs of their own.

During testing, many people chose to interpret Ingence's findings with an AI agent. Ingence does not need one, but we wanted agent use to be safe and simple. A user can export a summary of the project (ingence export brief) at one of four levels, each adding to the one before: counts only; Dictionary words and theme labels, with short aliases in place of experiment names; the experiments' real names and the words the Dictionary does not know; and each file's path inside its data folder. At no level does the summary include a file's contents, a full path, user or computer names, or the user's notes. The summary also stays within a length the user sets, 4,000 tokens by default (a token is the unit in which agents count text, about four characters), and leaves out experiments first to fit. The website describes each level.

Some agents use the Model Context Protocol (MCP), an open standard for connecting agents to programs. For these, the user adds ingence mcp to the agent's list of servers. The agent then starts Ingence itself and asks it questions over standard input and output, with no network. Ingence answers from the project's last scan, at the level the user chose, through nine read-only tools, such as an overview of the project, the list of experiments, or one experiment's story, and a tenth at the level that includes file paths. No tool can change the project, and an agent's answers enter a project only when the user imports them, marked as written by an agent and reviewed by the user.

Implementation and privacy

Ingence is written in Python and installs as one package, ingence. Typer builds its command-line interface from ordinary Python functions, and Rich formats what the terminal shows. Ingence stores observations and results in DuckDB (Raasveldt and Mühleisen 2019), an analytical SQL database that runs inside the program with no server, and in Apache Parquet files (Apache Software Foundation 2024), a column-oriented format for large tables, which it reads and writes through PyArrow. Pydantic checks that settings and records hold the expected types and values, and PyYAML reads the Dictionary and the Rulebook, which are YAML, a plain-text format for structured data. Whole-file digests use BLAKE3, a cryptographic hash function that, like SHA-256, gives each file a 256-bit fingerprint, and that runs faster on modern processors by hashing in parallel. The base install reads PDF text and metadata with pypdf, Excel workbooks with openpyxl and the containers of legacy Office files (.doc, .xls, .ppt) with olefile, and reads text, delimited tables, TIFF and the Office Open XML formats with Python's standard library. Themes use NumPy arrays (Harris et al. 2020), SciPy's sparse matrices (Virtanen et al. 2020) and scikit-learn (Pedregosa et al. 2011) for the reduction of word weights and the agreement measures, with a seeded Louvain implementation written in the engine. The standard library's HTTP server serves the Dashboard on the computer's loopback address, which other computers cannot reach. Optional extras add image readers (tifffile for TIFF pages and tags, Pillow for common image formats, imagehash for near-duplicate image fingerprints, pylibCZIrw for Zeiss CZI files), nbformat for Jupyter notebooks, datasketch for estimating how much two texts overlap, pyahocorasick for finding many terms in one pass over a text, and file-type detection with Magika (Fratantonio et al. 2024) and python-magic. Magika, a small, self-contained deep-learning model that runs on the user's computer to detect diverse file types, is the only trained component Ingence can use, and only when its optional detection extra is installed; it chooses which reader opens a file and doesn't author any results on its own.

Before each public release of Ingence, an automated test suite checks the engine. A test runs one part of Ingence on a small constructed input and fails if the result differs from what it must be. Examples from the suite include "a dive refuses a root it cannot reach and writes nothing", "results come back in the order the tasks were given, whatever the worker", "one hostile file kills one worker and the rest are read", "there are exactly eight primitives" and "the MCP tools have a schema each and no write tool". Browser tests open the Dashboard in a headless browser and work through its pages as a person would. The release build passed 7,700 suite tests and 640 browser tests; one suite test failed under load and passed when run alone.

Ingence doesn't collect or send telemetry; the author never sees or knows of any instance of it being run. It sends nothing over a network except when the user asks it to install a package, fetch a public archive's file list or install a pack from a web address. The engine does not check for updates; the website's What's new page and the package's release page on PyPI announce new releases. Privacy safeguards follow the same line. The Dashboard answers only on the computer's loopback address and refuses requests that come from other web pages. It has no accounts, so a person should run it on their own computer rather than on a shared server. Summaries for agents replace experiment names with aliases at the default level and leave out words the Dictionary does not know, which keeps personal names and project words out of what an agent reads. A user can share how their laboratory names files without sharing the names: ingence teach shapes prints each name's shape, such as {DATE}_{WORD}_{WORD}, with none of its words. An exported pack leaves out answers that apply to a single folder, since those name the project's folders. The included clinical words mark words that typically appear in identifiers as sensitive, but no rule yet flags a file that may hold identifiers, so users should not rely on Ingence to find them. A project holds file names and short extracts from files, so users should treat a project folder with the same care as the data it describes.

Evaluation

We tested Ingence v0.4.0 in three settings: on research servers of several laboratories at our institution during development, on an invented laboratory whose generator writes the ground truth, and on three public archives whose

depositors declared how their files relate. In each scored test, we compared Ingence's results with a ground truth that the engine never reads, written by the generator or declared by an archive's depositors, which gives two performance scores: the share of Ingence's results that were correct, and the share of the ground truth that Ingence found. Read together, they show whether Ingence was careful but missed much, or found much but also made many wrong proposals. Supplement S2 gives every count with its 95% confidence interval (Wilson 1927), and the full results of each test.

Development on laboratory servers

Throughout development, we also ran Ingence on research servers of several laboratories at our institution, since several of the people who tested it work in more than one laboratory. Where they could, the people who made those collections answered Ingence's questions, corrected its assumptions, and told us in development meetings what their collections hold. For collections whose depositors had left, other members of their laboratory, the author or Claude Opus 5.5 supplied the answers, and Claude also proposed meanings for words Ingence could not identify. The author or someone familiar with the work reviewed every answer, whatever its source. Testing results, user input and answers to the engine's questions entered Ingence the way any user's answers do, through the Record and through packs, and the engine runs fully without them. These runs shaped the rules, but no one wrote down the answers for those folders before Ingence read them, so we do not report them as scored results.

This input is a form of teaching. Ingence's knowledge is explicit: the Dictionary lists the words it knows, and each rule is a Python function that tests for specific kinds of evidence. However broadly they were written, the engine has no trained model, so it cannot recognize a word or a pattern that no entry or rule describes. We made the Dictionary and the Rulebook modular, portable and easy to edit so that people can add the words and patterns their own collections use, tailoring Ingence to their work as development tailored it to ours.

An invented laboratory

The Ingence codebase includes a generator, separate from the engine, that creates the data collection of an invented laboratory over several years. We built it from habits observed in real laboratory storage, and during development we used it to compare engine builds against each other. Generated collections include invented people and names, instrument files in the shapes of their formats, naming habits that change over time, files saved again as _v2 and _final, folders copied whole into backups, broken links, abandoned experiments, analyses whose raw data are gone, and a program installed inside a data folder (Figure 4a). Beside the collection, the generator writes its ground truth: what each file is and how files relate. The generator is part of the released package: ingence demo writes a small laboratory and opens it on the Dashboard, and python -m ingence.evaluation sample-lab regenerates any size and scores the engine against its ground truth (Supplement S3). Because the author of Ingence's rules also wrote the generator, and the invented collection served as development data, the scores are necessarily inflated. The generator's value is as a test that never changes: a fully known collection on which any change in engine behavior shows at once, and on which two versions can be compared exactly. A fresh install of the released package gave the same small-size results, rule for rule.

The generator makes its random choices from a seed, so the same seed always writes the same files, byte for byte, with the same ground truth and a datasheet (Gebru et al. 2021) (Supplement S3). At seed 1, its three sizes hold 430, 9,021 and 88,019 files. On the middle size (Figure 4b), Ingence found almost every copy (573 of 576), date in a name (751 of 768) and installed program (8 of 8), with no wrong answers, and 4,857 of its 4,882 stage results were correct. The rules that propose pairs for a person to review proposed far more than the ground truth holds: 1,151 of 9,582 proposed pairs of related experiments were correct, as were 1,562 of 5,974 proposals that one file was made from another. These are suggestions for a person to decide, and they show where the proposing rules most need tightening. One source of variability is worth stating: the same counts also need the same folder path, because at another path one rule keeps a different subset of the files a script names, and on the middle size the number of correct proposals that one file was made from another moved by 7% (119 of 1,681).

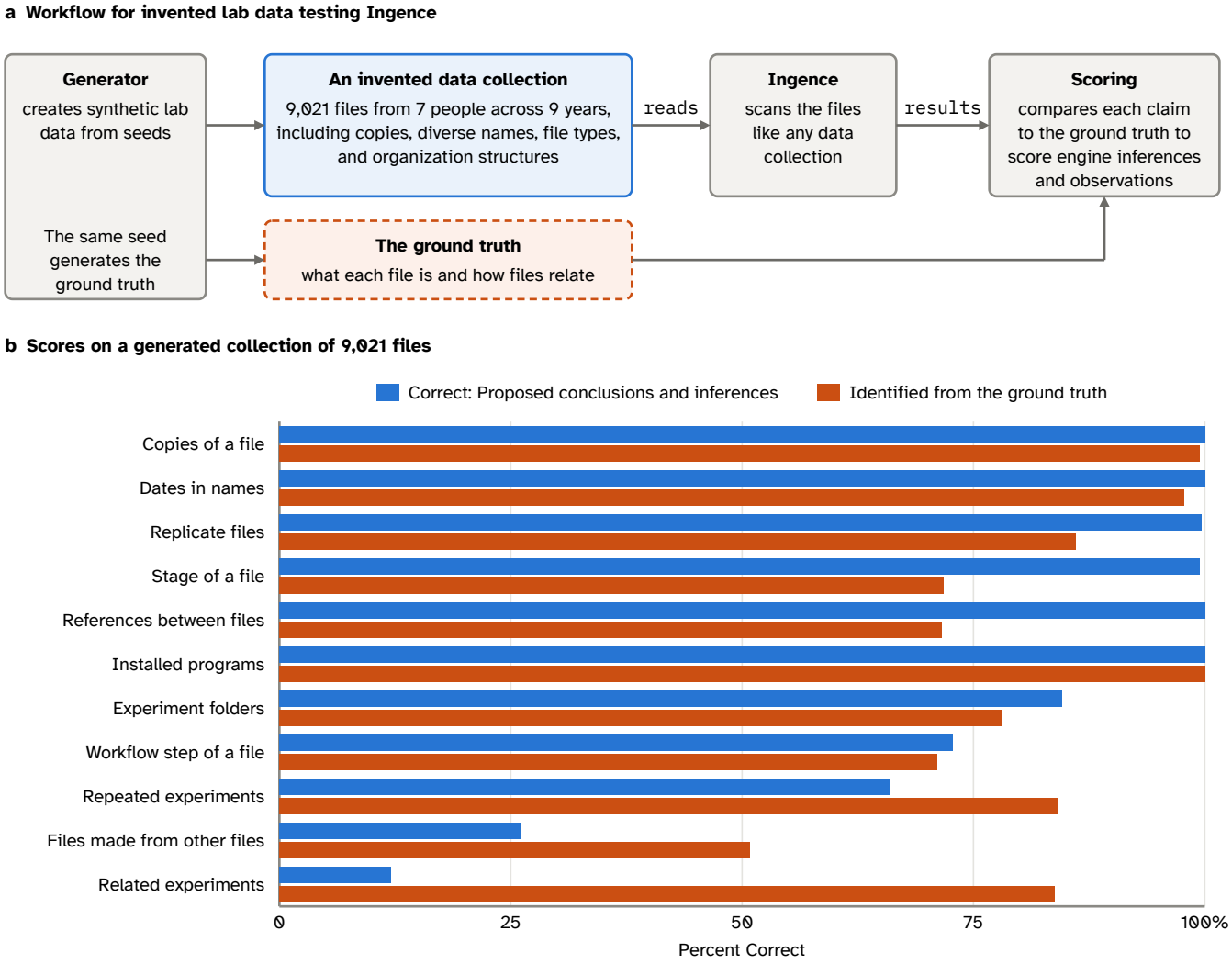


Figure 4. Engine performance scored on an invented laboratory. (a) A script generates a synthetic laboratory's data collection and its ground truth, which Ingence never reads, and a scorer compares Ingence's results with the ground truth. (b) Scores on a generated collection of 9,021 files (middle size, seed 1). Blue bars show the percentage of Ingence's results that were correct; orange bars show the percentage of the ground truth that Ingence found. The kinds of result include file associations (copies, replicate files, references between files, and related and repeated experiments), file derivations (files made from other files), and readings of single files and folders (dates, stages, experiment folders, workflow steps and installed programs). Two small kinds are not shown: files whose extension does not match their contents (23 of 23 correct and found) and empty files (13 of 14). Supplement S2 gives every count, the small size and 95% confidence intervals.

We also tested the question box on the invented laboratory. A command in the package (python -m ingence.evaluation questions) asks 30 questions of the kind a scientist writes, drawn from the generator's own record of each project, and scores the experiments the question box returns against the experiments the ground truth says each question is about (Supplement S3). On the small size, all 52 experiments returned as Named were correct, 52 of the 72 the ground truth expected. On the middle size, 124 of the 173 Named experiments were correct (124 of the 150 expected), and 39 of the 49 wrong ones were folders holding a file copied in from another experiment, whose name carries the question's words. A third tier, which widened answers to experiments associated with the question's words, returned far more wrong experiments than correct ones (126 of 5,968 correct on the middle size), so v0.4.0 ships with it switched off.

Public archives

Laboratory servers and generated collections cannot show how Ingence reads terms or parse file organization it has never seen, so we also tested it on public data repositories, whose large and diverse collections often come with annotations and descriptions that can serve as ground truth. We read three archives with different conventions (Figure 5): the

BioImage Archive, whose depositors name files as they choose; OpenNeuro, where every dataset follows one naming standard; and ENCODE, where each file carries three kinds of name. In each, Ingence read the archive's file lists and metadata as it would read a data collection, without downloading data, and we scored two things. Correct associations: did Ingence put files that belong together in the same group? Correct derivations: did it say which file was made from which, in the right direction? An archive is a harder test than a laboratory's collection. Many of Ingence's rules read the context that a laboratory's organization provides, such as experiment folders, dated names and raw files beside their analyses, and an archive arranged to hold millions of files from many groups provides less of it. These scores therefore likely understate how Ingence reads a single laboratory's collection. They also depend on how many derivations each archive records, since an archive can test only the derivations it declares.

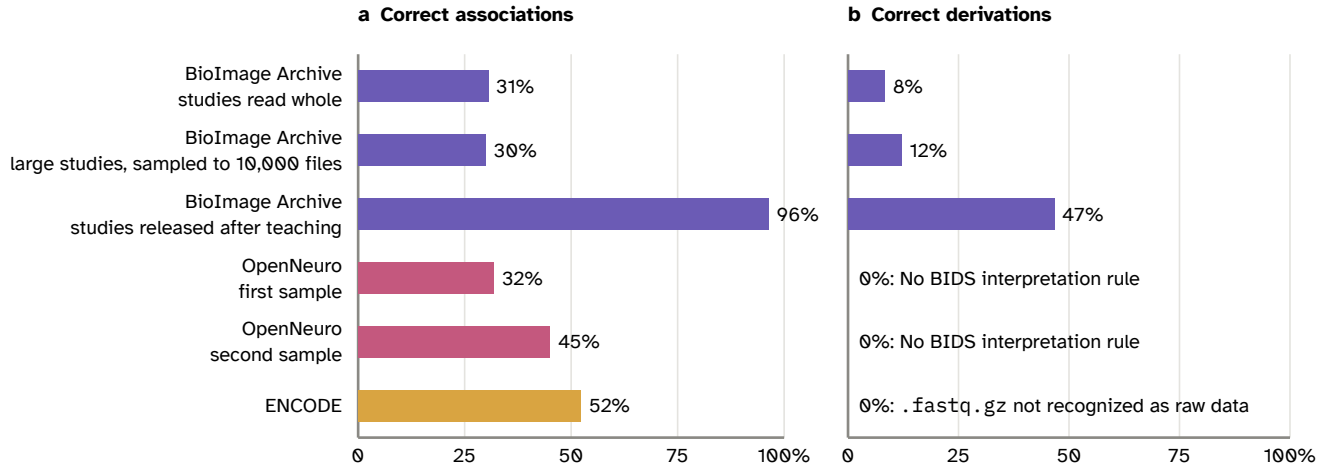


Figure 5. Engine performance on three public archives, scored against what each archive declares. (a) Correct associations: the share of files that belong together that Ingence placed in one group. For the BioImage Archive, a derived file and the file it was made from; for OpenNeuro, an acquisition's image and the files that describe it; for ENCODE, the files of biological replicates. (b) Correct derivations: the share of declared pairs of a file and the file it was made from that Ingence proposed, in the declared direction. The archives record derivations differently: some BioImage Archive deposits name each derived image's source, OpenNeuro holds derived files only in separate derivatives/ folders, and ENCODE declares the inputs of every processed file. BioImage Archive rows: studies of up to 10,000 files, read whole; larger studies, sampled to 10,000 files; and studies read in a later run, made after the earlier runs had been used to improve Ingence. OpenNeuro rows: two samples of about 100 datasets each, the first also used during development. Supplement S2 gives every count.

The BioImage Archive (Hartley et al. 2022) is a public repository for microscopy data. Most deposits state which files form a dataset and how and why the data were produced, and most accompany a published study. Some deposits also name, for each annotation or derived image, the image it was made from; these declarations are the archive's only record of derivations, and many studies make none. On 6 October 2026, the archive's catalogue listed 5,151 entries declaring 53,939,556 files. Ingence read the file lists and study metadata of 18 projects holding 3,344,943 files, reading studies containing up to 10,000 files entirely, and sampling up to 10,000 files of studies which contained more. Ingence received each file's path, size and the depositor's descriptive columns; the declared source of each derived file was withheld from the engine and used only for scoring. Because these studies and their testing influenced development, we performed a similar analysis using the release-build engine on 13 studies and 20,688 files, deposited after the first scan.

Ingence never reversed a declared derivation: every declared pair it proposed, 7,803 in the October studies and 860 in the later ones, pointed in the direction the depositor declared. In the studies read whole, it grouped 31% of declared derived files (14,305 of 46,838) with the file they were made from, and proposed 8% (3,909) as made from that file (Figure 5). Most missed pairs follow layouts common in machine-learning and screening datasets, such as a derived image in a side-by-side folder whose name Ingence does not read as a role (groundtruth/X.tif beside rawimages/X.tif). Ingence's experiments matched the declared datasets where each dataset is its own folder, but a reading based on folders cannot recover a dataset defined by a condition or imaging method spread across folders (Supplement S2). In the later studies, which we also scored during development, Ingence grouped 96% of declared derived files with their source and proposed

47% as made from it, but this reflects how two depositors named their files rather than a general rate: one rule, which reads a file named after an acquisition beside it, made every one of these proposals, and they came from two studies. Overall, the BioImage Archive tested Ingence's indexing at scale more than its ability to associate files, since the archive's organization carries little of the context its rules read.

Before choosing which studies to score, we ran scripts built on Ingence's census across the whole archive, to plan the sample and to test indexing on a very large collection. Fetching the metadata and file lists of every study through the archive's public web interfaces surfaced findings about the archive itself, from metadata alone (Supplement S4). Of the 1,388 studies with both a catalogue file count and a file list, 200 had counts and lists that disagreed. Grouping files of 1 MB or more by name and size, the first step Ingence takes when it looks for copies, found 147,537 sets of files that share a name and size across studies, all within 54 groups of studies; reading samples and then whole files proved some of these copies byte-identical, among them a study deposited twice under two spellings of its accession. In addition, 245 files in seven studies answered "not found" at the addresses their file lists gave. These observations informed how the census handles errors and the default settings for grouping experiments into themes.

OpenNeuro (Markiewicz et al. 2021) is a public archive for neuroimaging and electrophysiology data, collected using techniques like magnetic resonance imaging (MRI), positron emission tomography (PET), and electroencephalography (EEG). In October 2026, its public storage held 2,493 dataset folders. Every deposit must pass validation against the Brain Imaging Data Structure (BIDS, Gorgolewski et al. 2016), a community standard for how files and folders are named and arranged. Under BIDS, a path such as sub-01/ses-02/func/sub-01_ses-02_task-rest_run-1_bold.nii.gz states a file's subject, session, task, run and type, and a file beside it with the same name parts carries the acquisition's settings. Derived files appear only in datasets that include a separate derivatives/ folder, where BIDS names each one after the acquisition it came from. We built two samples of 102 and 89 datasets, taking every 20th dataset in the archive's list, with the second sample starting ten places after the first so the two share no dataset. Ingence received each file's path, size, checksum and upload time, as the archive lists them. Because BIDS writes the answers into the paths themselves, a script computed the ground truth from those same paths using the BIDS rules, and Ingence never read its output; the test asks whether Ingence's general rules read structure that BIDS states explicitly.

Ingence correctly matched an acquisition's image with the files that describe it for 32% of such pairs in the first sample (46,480 of 145,664) and 45% in the second (49,960 of 110,883) (Figure 5a), and most of the pairs it proposed were correct (94% and 79%). BIDS marks repeated acquisitions with a run label, such as run-01 and run-02. Ingence grouped only 9% and 3% of these repeats, because it does not assume what a label like run means: it asks the user once, and one answer then applies to every dataset. It read few subject or session folders as experiments, because the experiment rules look for instrument formats and dated names, which BIDS doesn't provide. It found none of the declared derived files (Figure 5b): the listings included the derivatives/ folders, but no current rule reads BIDS's convention of naming each derived file after its source, and the pairs Ingence did propose as made from one another, mostly files named after an acquisition beside them, are not derivations under BIDS. Future development will include rules that read standardized annotation and metadata formats, including BIDS, and frameworks such as ASTRA (Agentic Schema for Transparent Research Analysis) (Lightcone Research 2026) and ARC (Annotated Research Context) (DataPLANT 2026).

The Encyclopedia of DNA Elements (ENCODE) is a consortium project to identify the functional elements of the human and mouse genomes. Its portal shares more than 23,000 functional genomics experiments annotated using a standardized system (Kagda et al. 2025). For each processed file, such as an alignment made from sequencing reads, ENCODE declares the files it was made from, which makes it the most complete record of derivations among the three archives. Each file also carries three kinds of name: the archive's accession number, the name its submitters gave, and the name of the processing pipeline, which directly tests how Ingence's rules use names to find associations and derivations. We read the metadata of transcription-factor ChIP-seq, RNA-seq and ATAC-seq experiments in three cell lines, each set chosen by a rule fixed before fetching: 1,356 files from 44 K562 experiments and controls, of which 29 were kept for scoring and the rest used to develop and teach the engine, and 2,126 files from 52 HepG2 and GM12878 experiments

and controls, chosen to confirm the K562 results. Ingence received each file's name, under each kind of name in turn, its size and its experiment's folder; the declared inputs of each file were withheld and used only for scoring. As a second, separate collection, we also read 657 files listed by 30 series in the Gene Expression Omnibus (GEO), the National Center for Biotechnology Information's archive of functional genomics data (Barrett et al. 2013).

Accession numbers carry no meaning that Ingence can read, and it made no results and no errors under them. With no words added to its Dictionary, Ingence recognized 45% of the words in the names ENCODE's submitters gave (1,794 of 3,999) and 40% of the words in GEO's names (280 of 705); a word counts as recognized when the Dictionary holds a meaning for it. We then taught Ingence a pack of 20 words from the names of the K562 experiments used in development, each word reviewed by a person, and the share of recognized words rose by 36 and 32 points in the HepG2 and GM12878 experiments. Under submitted names, Ingence found half of ENCODE's pairs of biological replicate files (168 of 321, Figure 5a), but only about half of the pairs it proposed were correct (168 of 341). Ingence found none of the derivations ENCODE declares (Figure 5b), for a specific reason. ENCODE's provenance begins with sequencing reads, which archives distribute as compressed .fastq.gz files, and Ingence's file-type table counts a FASTQ file as raw data only when its name ends in .fastq or .fq. Ingence therefore placed no read file at the start of a workflow, and no provenance could start from one. Instead, a rule new in v0.4.0, which reads two files with the same name, one with words for a later step added, as a source and its product, proposed 199 pairs that ENCODE does not declare. Each was a pair of files processed side by side from the same input, not one made from the other, so all 199 were wrong, although none proposed that a source was made from its own product. In an earlier build that counted read files as raw data, 472 of 478 proposed pairs matched ENCODE's declared provenance. Counting the compressed spellings is a small update planned for v0.4.1.

Collections differ in vocabulary, naming habits, instruments and workflows from one laboratory to the next, and even between people in one laboratory, so a score on one collection says little about another. The scores on the invented laboratory and the public archives show where the rules read well and where a convention is missing. User-supplied answers, applied to every assumption they clarify, are the best improvement for Ingence's performance, and on the laboratory collections we tested, the answers people gave shaped the rules more than any single score. Future work aims to make the engine more adaptable to diverse research collections through open-source collaboration: shared Dictionary packs, rule scripting, and testing on more community data repositories.

Discussion

This report describes Ingence v0.4.0, a read-only engine that reads a research data collection and proposes how its files relate. It lists every file, reads a sample of contents, reads every name with one tokenizer and Dictionary, and turns what it observes into results with written rules, each built from one of eight relations and each naming its rule and evidence. Where evidence supports an inference, Ingence reports it; where evidence is missing, it gives no result and says why. The Dashboard centers experiments as the basic research unit, with their dates, completeness, repeats and overlap, and every answer a person gives enters the Record and applies at the next scan. We scored the engine against ground truth it never reads, from an invented laboratory and three public archives.

The scores show an engine that reads well what names and metadata state plainly, and reads less where a collection follows a convention its rules don't recognize. In the invented laboratory, it found copies, dates and the stages of files reliably, and in the BioImage Archive and ENCODE, it never reversed the direction of a declared derivation. Where it could not read names at all, as with ENCODE's accession numbers, it gave no result and made no error, and its wrong answers came mostly from the rules that propose pairs for a person to review. Each gap it showed, from BIDS's derivatives/ folders to compressed sequencing files and words not yet in the Dictionary, comes from Ingence's familiarity with specific words and patterns of data organization, not from how the engine is designed, and a pack, a parameter or a new rule can

close it. Every research collection is different, and no engine, with or without a language model, can know a laboratory's abbreviations, habits and workflows before someone with prior knowledge defines them.

Ingence was designed with the understanding that it will be wrong. Its goal is not to be correct after its first scan, but to help users see and explore large data collections in a way that makes sense. A result that names its rule and evidence takes seconds to check. Where Ingence shows a wrong assumption, it counts the relevant files and gives examples, and one answer corrects everywhere that assumption appears. All user feedback and input is recorded in the versioned Record and kept for every later scan. Through this loop, engine errors are easier to correct and become inputs for learning. This is the motivation for using rules as units of logical interpretation, rather than building a training engine that learns and writes its own: a person can read a rule and debate its result, anyone can rebuild a scan exactly from its inputs, and what one person teaches Ingence may apply everywhere else similar data is found. Collaboration between the user and the engine creates a portable record that can be passed between lab members and colleagues, saving the time they might otherwise spend reorienting themselves or others to a data collection and its research goals.

The same design makes Ingence useful beside AI agents rather than in place of them. Many institutions do not allow an agent to crawl their research servers, and even where one may, nothing guarantees that it only reads. Ingence reads a collection once, read-only and locally, and then gives a person's own agent the structure it would otherwise have had to find by exploring: experiments, dates, completeness, themes and the evidence behind each, at the level of detail the person chooses. The agent can help interpret and correct what Ingence found, its suggestions return only as answers a person has reviewed, and it never touches the laboratory's storage.

Future releases will follow from what these evaluations found, from user feedback and, we hope, from community input. Directions include recognizing more file types, faster inference that uses less memory, more careful reading of the text inside files, and rules that read standardized metadata formats. People who tested Ingence also asked for links from notebook entries to the raw data they describe, instrument settings read from file headers, and one view of a whole laboratory assembled from personal projects. Most of this work fits the existing structure: a new word or naming convention is an entry in a pack, a new reading of evidence the engine already observes is a rule added to the Rulebook as data, and only a new kind of evidence needs a new reader and function. Each addition is versioned, so it can be measured against the release before it.

Ingence improves fastest when the people who know their collections share what they taught it. A laboratory can export the words it taught as a pack, and another laboratory can install it; each pack names its author and license and carries a digest of its words, so a shared vocabulary can grow across the laboratories of one field. These contributions reach us through the website's contact form and the GitHub repository's issues and discussions. Because the code, the Dictionary and the Rulebook are all open source, any group can also adapt Ingence for their own purposes and research.

Supplemental Material, Code, and Data Availability

Supplemental material is available on Zenodo, under a CC0 license at 10.5281/zenodo.23756686. Supplement S1 gives the eight relations with all seven properties and the formal argument. Supplement S2 holds the documents fixed before each test, with their digests and outcomes, the full results of each test, and the sources behind Table 1. Supplement S3 is the invented laboratory's datasheet, with the commands that regenerate and score it. Supplement S4 holds the code we used to fetch, read and score the public archives, and the findings about the BioImage Archive from its metadata, with their table. The website generates the rule catalogue and the settings for each version at ingence.org/reference.

Ingence is open source under the MIT license. The source code is maintained on github.com/andrewbazley/Ingence, and the package is on PyPI as ingence (pypi.org/project/ingence/). Detailed documentation, usage reference, and API are available at www.ingence.org. The invented laboratory data collection, the scoring and figure scripts, and the code we used to read the public archives, are all available on Zenodo, under a CC0 license at 10.5281/zenodo.23756686.

Acknowledgments

We would like to thank the Astera Institute for supporting this work through the Hidden Science Competition and their commitment to open science. We thank Liam Holt for his guidance and support, and the members of his and other laboratories who contributed to Ingence development through access to their data collections and feedback on Ingence runs. We would like to thank the BioImage Archive, OpenNeuro and ENCODE for their service in making data archives publicly available, and for their contribution to Ingence testing.

We developed Ingence's code, tests and documentation, drafts of its website and Dashboard design, this manuscript's figures, and parts of its text with Anthropic's Claude models through Claude Code and Claude Design. During testing, Claude Opus 5.5 proposed meanings for unidentified words and supplied answers where a collection's depositor had left, which a person reviewed. The author directed this entire work and reviewed and tested all of what these tools produced.

References

Allan, Chris, Jean-Marie Burel, Josh Moore, Colin Blackburn, Melissa Linkert, Scott Loynton, Donald MacDonald, et al. 2012. "OMERO: Flexible, Model-Driven Data Management for Experimental Biology." *Nature Methods* 9 (3): 245–53. https://doi.org/10.1038/nmeth.1896.

Apache Software Foundation. 2024. "Apache Parquet: File Format." Format documentation. https://parquet.apache.org/docs/file-format/.

Barrett, Tanya, Stephen E. Wilhite, Pierre Ledoux, Carlos Evangelista, Irene F. Kim, Maxim Tomashevsky, Kimberly A. Marshall, et al. 2013. "NCBI GEO: Archive for Functional Genomics Data Sets–Update." *Nucleic Acids Research* 41 (D1): D991–95. https://doi.org/10.1093/nar/gks1193.

Blondel, Vincent D, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. "Fast Unfolding of Communities in Large Networks." *Journal of Statistical Mechanics: Theory and Experiment* 2008 (10): P10008. https://doi.org/10.1088/1742-5468/2008/10/P10008.

Chan, Joel, Matthew Akamatsu, David Vargas, Lukas Kawerau, and Michael Gartner. 2024. "Steps Towards an Infrastructure for Scholarly Synthesis." https://doi.org/10.48550/arXiv.2407.20666.

DataPLANT. 2026. "ARC: Annotated Research Context." Framework documentation. https://arc-rdm.org/.

Deerwester, Scott, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. "Indexing by Latent Semantic Analysis." *Journal of the American Society for Information Science* 41 (6): 391–407. https://doi.org/10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9.

Fratantonio, Yanick, Luca Invernizzi, Loua Farah, Kurt Thomas, Marina Zhang, Ange Albertini, Francois Galilee, et al. 2024. "Magika: AI-Powered Content-Type Detection." arXiv preprint arXiv:2409.13768. https://doi.org/10.48550/arXiv.2409.13768.

Gebru, Timnit, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. 2021. "Datasheets for Datasets." *Communications of the ACM* 64 (12): 86–92. https://doi.org/10.1145/3458723.

Gorgolewski, Krzysztof J., Tibor Auer, Vince D. Calhoun, R. Cameron Craddock, Samir Das, Eugene P. Duff, Guillaume Flandin, et al. 2016. "The Brain Imaging Data Structure, a Format for Organizing and Describing Outputs of Neuroimaging Experiments." *Scientific Data* 3: 160044. https://doi.org/10.1038/sdata.2016.44.

Greenblatt, Ryan, Ajeya Cotra, and Hjalmar Wijk. 2026. “Brief Independent Investigation of the OpenAI / Hugging Face Hacking Incident.” METR, 26 August 2026. https://metr.org/blog/2026-08-26-openai-hugging-face-incident-investigation/.

Groth, Paul, Yolanda Gil, and Sara Magliacane. 2012. “Automatic Metadata Annotation Through Reconstructing Provenance.” In *Proceedings of the 3rd International Workshop on Semantic Web in Provenance Management (SWPM 2012), Co-Located with the 9th Extended Semantic Web Conference (ESWC 2012), Heraklion, Greece, 28 May 2012*. Vol. 856. CEUR Workshop Proceedings. https://ceur-ws.org/Vol-856/paper_4.pdf.

Halchenko, Yaroslav, Kyle Meyer, Benjamin Poldrack, Debanjum Solanky, Adina Wagner, Jason Gors, Dave MacFarlane, et al. 2021. “DataLad: Distributed System for Joint Management of Code, Data, and Their Relationship.” *Journal of Open Source Software* 6 (63): 3262. https://doi.org/10.21105/joss.03262.

Halevy, Alon, Flip Korn, Natalya F. Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. “Goods: Organizing Google’s Datasets.” In *Proceedings of the 2016 International Conference on Management of Data*, 795–806. https://doi.org/10.1145/2882903.2903730.

Harris, Charles R., K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, et al. 2020. “Array Programming with NumPy.” *Nature* 585 (7825): 357–62. https://doi.org/10.1038/s41586-020-2649-2.

Hartley, Matthew, Gerard J. Kleywegt, Ardan Patwardhan, Ugis Sarkans, Jason R. Swedlow, and Alvis Brazma. 2022. “The BioImage Archive – Building a Home for Life-Sciences Microscopy Data.” *Journal of Molecular Biology* 434 (11): 167505. https://doi.org/10.1016/j.jmb.2022.167505.

Kagda, Meenakshi S., Bonita Lam, Casey Litton, Corinn Small, Cricket A. Sloan, Emma Spragins, Forrest Tanaka, et al. 2025. “Data Navigation on the ENCODE Portal.” *Nature Communications* 16: 9592. https://doi.org/10.1038/s41467-025-64343-9.

Levitas, Daniel, Soichi Hayashi, Sophia Vinci-Booher, Anibal Heinsfeld, Dheeraj Bhatia, Nicholas Lee, Anthony Galassi, Guiomar Niso, and Franco Pestilli. 2024. “ezBIDS: Guided Standardization of Neuroimaging Data Interoperable with Major Data Archives and Platforms.” *Scientific Data* 11: 179. https://doi.org/10.1038/s41597-024-02959-0.

Lightcone Research. 2026. “ASTRA: Agentic Schema for Transparent Research Analysis.” Specification, version 0.0.14. https://astra-spec.org/.

Markiewicz, Christopher J, Krzysztof J Gorgolewski, Franklin Feingold, Ross Blair, Yaroslav O Halchenko, Eric Miller, Nell Hardcastle, et al. 2021. “The OpenNeuro Resource for Sharing of Neuroscience Data.” *eLife* 10: e71774. https://doi.org/10.7554/eLife.71774.

Muniswamy-Reddy, Kiran-Kumar, David A. Holland, Uri Braun, and Margo Seltzer. 2006. “Provenance-Aware Storage Systems.” In *2006 USENIX Annual Technical Conference (USENIX ATC 06)*. Boston, MA: USENIX Association. https://www.usenix.org/conference/2006-usenix-annual-technical-conference/provenance-aware-storage-systems.

Murta, Leonardo, Vanessa Braganholo, Fernando Chirigati, David Koop, and Juliana Freire. 2015. “noWorkflow: Capturing and Analyzing Provenance of Scripts.” In *Provenance and Annotation of Data and Processes: 5th International Provenance and Annotation Workshop, IPAW 2014, Cologne, Germany, June 9–13, 2014. Revised Selected Papers*, edited by Bertram Ludäscher and Beth Plale, 71–83. Lecture Notes in Computer Science. Springer. https://doi.org/10.1007/978-3-319-16462-5_6.

O’Connor, Jack, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O’Hearn. 2021. “BLAKE3: One Function, Fast Everywhere.” Specification. https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf.

OECD.AI. 2025a. “Google Gemini CLI AI Flaws Lead to Data Loss and Security Risks.” AI Incidents Monitor, incident of 25 July 2025. https://oecd.ai/en/incidents/2025-07-25-03b9.

OECD.AI. 2025b. “Replit AI Coding Tool Deletes Live Production Database and Fakes Data.” AI Incidents Monitor, incident of 19 July 2025. https://oecd.ai/en/incidents/2025-07-19-1eb1.

OpenAI. 2026. “OpenAI – Hugging Face Incident Technical Report.” OpenAI, 26 August 2026. https://cdn.openai.com/pdf/67869394-cb91-4c12-888c-5cbd85c7814c/OpenAI-Hugging-Face%20Incident-Technical-Report.pdf.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, et al. 2011. “Scikit-Learn: Machine Learning in Python.” *Journal of Machine Learning Research* 12: 2825–30. https://jmlr.org/papers/v12/pedregosa11a.html.

Raasveldt, Mark, and Hannes Mühleisen. 2019. “DuckDB: An Embeddable Analytical Database.” In *Proceedings of the 2019 International Conference on Management of Data*, 1981–84. https://doi.org/10.1145/3299869.3320212.

Reichardt, Jörg, and Stefan Bornholdt. 2006. “Statistical Mechanics of Community Detection.” *Physical Review E* 74 (1): 016110. https://doi.org/10.1103/PhysRevE.74.016110.

Shah, Raunak, Koyel Mukherjee, Atharv Tyagi, Sai Keerthana Karnam, Dhruv Joshi, Shivam Pravin Bhosale, and Subrata Mitra. 2023. “R2D2: Reducing Redundancy and Duplication in Data Lakes.” *Proceedings of the ACM on Management of Data* 1 (4): 268:1–25. https://doi.org/10.1145/3626762.

Skluzacek, Tyler J., Kyle Chard, and Ian Foster. 2022. “Automated Metadata Extraction: Challenges and Opportunities.” In *2022 IEEE 18th International Conference on e-Science (e-Science)*, 495–500. https://doi.org/10.1109/eScience55777.2022.00088.

Skluzacek, Tyler J., Ryan Wong, Zhuozhao Li, Ryan Chard, Kyle Chard, and Ian Foster. 2021. “A Serverless Framework for Distributed Bulk Metadata Extraction.” In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, 7–18. https://doi.org/10.1145/3431379.3460636.

Spärck Jones, Karen. 1972. “A Statistical Interpretation of Term Specificity and Its Application in Retrieval.” *Journal of Documentation* 28 (1): 11–21. https://doi.org/10.1108/eb026526.

Vinh, Nguyen Xuan, Julien Epps, and James Bailey. 2010. “Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance.” *Journal of Machine Learning Research* 11: 2837–54. https://jmlr.org/papers/v11/vinh10a.html.

Virtanen, Pauli, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, et al. 2020. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python.” *Nature Methods* 17: 261–72. https://doi.org/10.1038/s41592-019-0686-2.

Wilson, Edwin B. 1927. “Probable Inference, the Law of Succession, and Statistical Inference.” *Journal of the American Statistical Association* 22 (158): 209–12. https://doi.org/10.1080/01621459.1927.10502953.

Winston, Morton E., Roger Chaffin, and Douglas Herrmann. 1987. “A Taxonomy of Part-Whole Relations.” *Cognitive Science* 11 (4): 417–44. https://doi.org/10.1207/s15516709cog1104_2.

Zwiers, Marcel Peter, Stefano Moia, and Robert Oostenveld. 2022. “BIDScoin: A User-Friendly Application to Convert Source Data to Brain Imaging Data Structure.” *Frontiers in Neuroinformatics* 15: 770608. https://doi.org/10.3389/fninf.2021.770608.