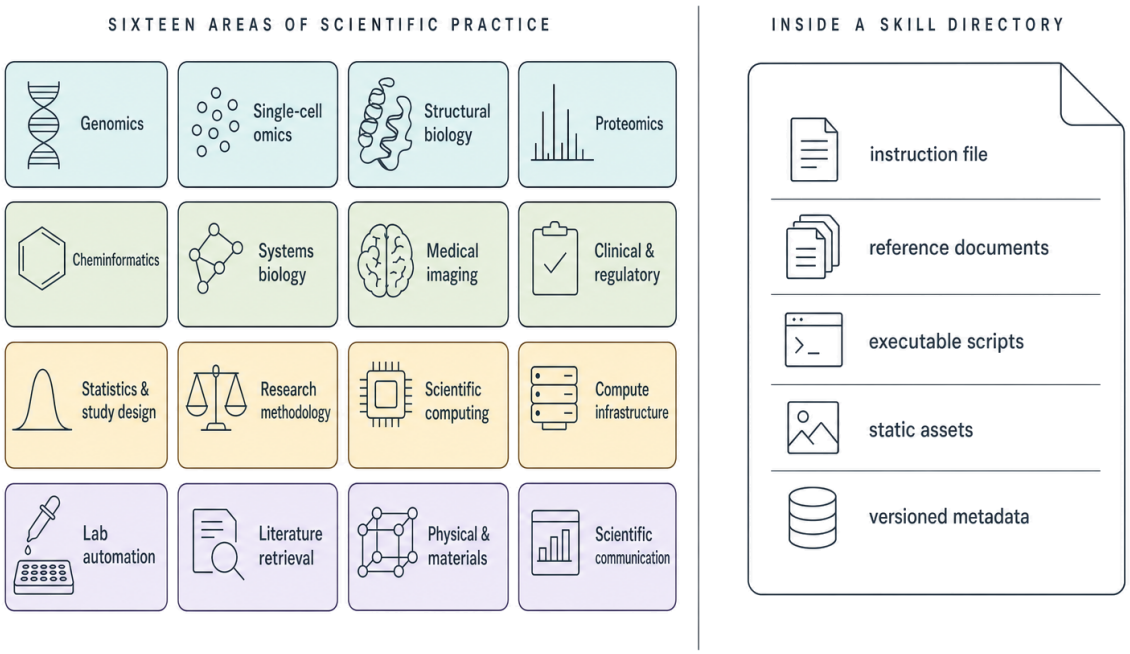
arXiv:2609.00065v1 [cs.CL] 30 Aug 2026

Scientific Agent Skills: A Library of Procedural Knowledge for Research Agents

Timothy Kassis*, Vinayak Agarwal, Yuhuan He, Darshil Patel, Aubrey M. Brueckner
K-Dense, Inc.

September 2, 2026



Abstract

A language-model agent asked to analyse an experiment will usually return working code. Whether the analysis is defensible is a different question. A defensible analysis depends on procedural choices: which test the field accepts, which identifier namespace is authoritative, and which caveats must accompany a result. We present Scientific Agent Skills, an open library of 163 such procedures in 16 areas of practice, including genomics, cheminformatics, medical imaging, study design and scientific communication. Each skill is a directory built around a versioned, human-readable instruction file. An agent loads the file only when a task calls for it; the directory often also contains reference material and runnable scripts. We report no task-level evaluation and no host selection rate. Openly licensed and available at `https://github.com/K-Dense-AI/scientific-agent-skills`.

*Correspondence: `timothy.kassis@k-dense.ai`

1 Introduction

A researcher who asks a coding agent to analyse an RNA-seq experiment will usually get working code. Whether they get a defensible analysis is a different question. For example, the agent may compare groups without correcting for multiple testing, average a column in which -999 encodes a failed well, treat a BED interval's zero-based start as one-based, or report a hazard ratio from a model that omits a treatment-confounding covariate. Each mistake can produce plausible code while invalidating the analysis under the stated conditions. The relevant conventions are documented: multiple-testing correction is standard [1], technical confounding affects high-throughput analyses [2], BED and related formats use explicit coordinate conventions [3], omitted covariates bias Cox-model treatment estimates [4], and spreadsheet conversion of gene symbols has propagated into published work [5]. An evaluation of agents on real scientific requests found that scientific accuracy trailed communication within every model and that overclaiming was the most common failure tag [6].

The recurring problem is procedural: agents need field-specific conventions that specify which test applies, which identifier namespace is authoritative, and which caveats must accompany a result. That knowledge is distributed across package documentation, reporting guidelines, standards, and the tacit practice of individual fields. Rediscovering it for every task is costly, so it can be recorded once. Scientific communities already record such knowledge for people. Reporting guidelines such as ARRIVE [7] and MIQE [8] exist because the same procedural mistakes recur across the published literature [9].

Agent Skills [10] are a convention for recording this procedural knowledge for agents. A skill is a directory containing a Markdown instruction file with a short YAML header, optionally accompanied by reference documents, executable scripts and static assets. An agent host scans the skills it has been given and holds each skill's name and short description in its system prompt. It reads the full instruction file only when a task appears to call for that skill. The convention specifies no runtime, API or service. Because the convention is file-based, a skill is portable across hosts and reviewable as text.

This paper describes Scientific Agent Skills [11], an openly licensed library of 163 skills that we build and maintain, covering research workflows in biology, chemistry, medicine, the physical sciences and scientific communication. We wrote most of them ourselves (Appendix B), so this is a description of our own artifact rather than an independent audit. Sections 2 and 3 describe what a skill is and what the library contains, so readers can judge whether it is relevant to their work. We then measure two properties of the documentation files: how well the short descriptions that remain in context are separated lexically (Section 4), and whether the skills named by the library's example workflows fit in a context window (Section 5). We recompute every quantity from the repository tree at a pinned release instead of taking it from our own documentation. Section 6 states what has and has not been measured. The appendices contain the full descriptive census, the measurement conventions, and the validation and scope ledgers.

This is a resource paper: its purpose is to introduce the library, not to validate the effect of installing any individual skill. We have published our own benchmarks of several skills at `https://www.k-dense.ai/benchmarks`, but those studies do not validate the collection. Each skill needs evaluation on tasks and outcomes appropriate to its domain, and broader independent work by the communities represented here remains necessary. The two measurements we report are properties of the documentation corpus: they say where selection is hard and what selection costs, not how often an agent selects well.

1.1 Related work

Recent systems write and run scientific code for autonomous chemistry [12], hierarchical multi-agent bioinformatics analysis [13], and end-to-end research automation [14]; others package biomedical research workflows over a shared tool layer [15]. Coding agents are also evaluated on repository-scale tasks [16] and scientific programming [17]. Error analysis of a data-driven discovery benchmark finds that most failures are programs that execute but are semantically incorrect, often because they misuse discipline-specific tools, rather than programs that fail to run [18].

Prior approaches put knowledge into a model's context through corpus retrieval [19], tool interfaces [20, 21], or routines accumulated by the agent itself [22]. Skills differ from retrieval because each skill is an authored procedure rather than a passage. They differ from tools because a skill provides no runtime to call. Retrieval can degrade in long contexts [23], so the tiered design in Section 2 affects how the library operates, not just how it is presented.

Recent work examines how skill libraries are built, catalogued and evaluated [24–26], how they behave as they grow [27], and what happens when an agent must retrieve its own skills from a large uncurated collection [28], and what happens when its tool layer already returns strict, schema-validated observations [29]. Both report benefits shrinking toward the no-skill score, in the first case once the agent must select the skill itself. Section 4 therefore measures how distinct the short descriptions that stay in context are. Other work treats agent-loadable instruction files as an attack surface [30–35]. Appendix E places the library's security material in this context. Appendix A relates skills to workflow engines and data standards, which address the same procedural problem for pipelines.

2 What an agent skill is

A skill is a directory. At minimum it contains `SKILL.md`: a YAML header followed by Markdown instructions written for an agent rather than a human reader. The header is constrained to six fields: `name`, `description`, `license`, `compatibility`, `allowed-tools` and `metadata`. Any other top-level key is a specification error. The specification recommends three conventional subdirectories: `references/` for additional documentation, `scripts/` for code, and `assets/` for templates and static resources, while permitting other files and directories. We adopt a stricter layout and allow only `SKILL.md` and those three subdirectories inside each skill (Appendix D). Appendix B describes the directory structure and instruction-file contents, and shows a skill in its shipped form.

For the skills we write ourselves, our authoring workflow usually begins with a language-model draft. We then review the text manually and test the documented workflow for basic functionality when that workflow's environment is available. We revise each skill as we learn when it works and when it does not, drawing on our own use, community feedback and outside contributions. These steps are authoring and maintenance, not domain validation; Appendix E states what the automated checks cover.

2.1 Progressive disclosure

The design keeps almost all library content out of standing context. Under the specification's disclosure model, an agent host reads the `name` and `description` of every installed skill into its system prompt at startup. What a host adds around those entries is host-specific, as discussed in Section 6. The instruction body is read when the agent judges the skill relevant; a reference document is read only if the instruction body points at it. The disclosure model has three tiers whose sizes differ by orders of magnitude (Figure 1).

Across the whole library, the resident tier is 14,246 tokens, a median of 67 tokens per skill. The complete instruction files behind the resident tier total 482,506 tokens, with a median of 2,857 tokens each. The reference documents behind *those* instruction files total 2,480,674 tokens across 1,033 files. As a result, 83.7% of the library's documentation remains unread unless an activated skill points to it. Holding the entire library available costs 0.48% of the 2,963,180-token corpus in standing context, or 7.1% of a 200,000-token reference window. This low resident cost allows a host to offer 163 specialised procedures without filling its context window with procedures the current task does not need.

Tiering defers cost; it does not remove it. A tier the agent never reads cannot inform its choice, so routing for the whole library relies on the resident tier alone. Reference material saves context only while it remains deferred during a real task. Sections 4 and 5 measure these constraints. Both measurements describe the documentation layout rather than agent traces: they show what selection costs and where it may be difficult, not whether any agent selects correctly. Appendix A states the tokenizer and the corpus definition the figures depend on.

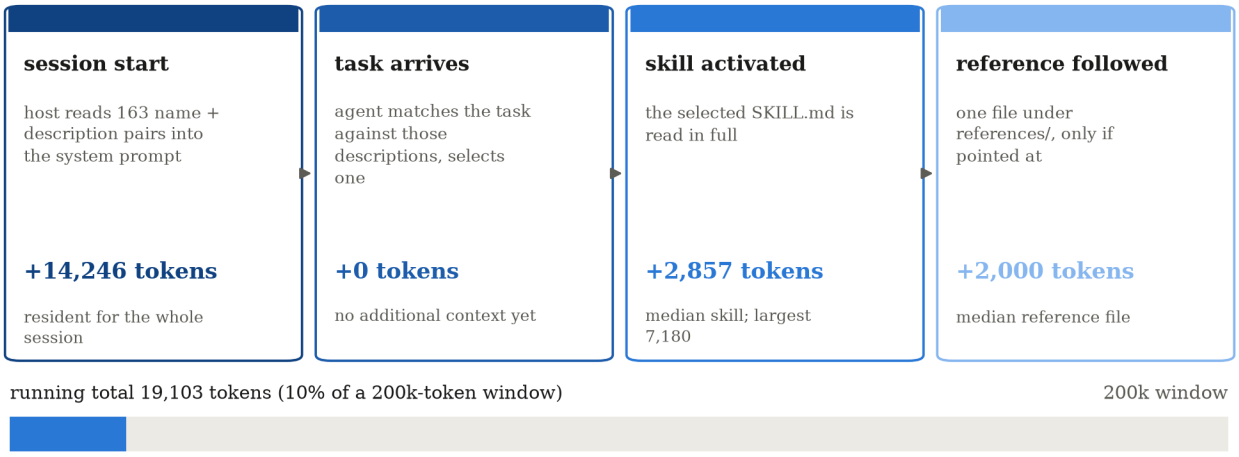


Figure 1: Measured token costs of the three disclosure tiers at tag v2.65.0, shown along a hypothetical task path. These are corpus measurements, not an agent trace. Selecting a skill uses descriptions already in standing context, so it adds no documentation tokens; Section 4 treats that step separately. The running total assumes a median skill and one median reference file. Section 5 costs the library's documented workflows instead.

3 What the library covers

The library is organised as one directory per skill under `skills/`, with no enforced grouping above that level. Our own documentation groups skills in three different, partly overlapping ways, none of which partitions the library. For reporting, we assign each skill exactly one primary category. This taxonomy is our reporting convention, not a property of the repository. Appendix C gives the full taxonomy and per-category counts. At this tag the 163 skills span 16 categories, the largest being scientific communication and figures, scientific computing and data engineering, and genomics and sequence analysis. 105 skills ship executable code under `scripts/` and 154 ship reference documents (Figure 2).

The library includes four recurring skill types across categories. *Package workflows* document how to use a specific scientific package correctly, such as Scanpy [36] for single-cell analysis or

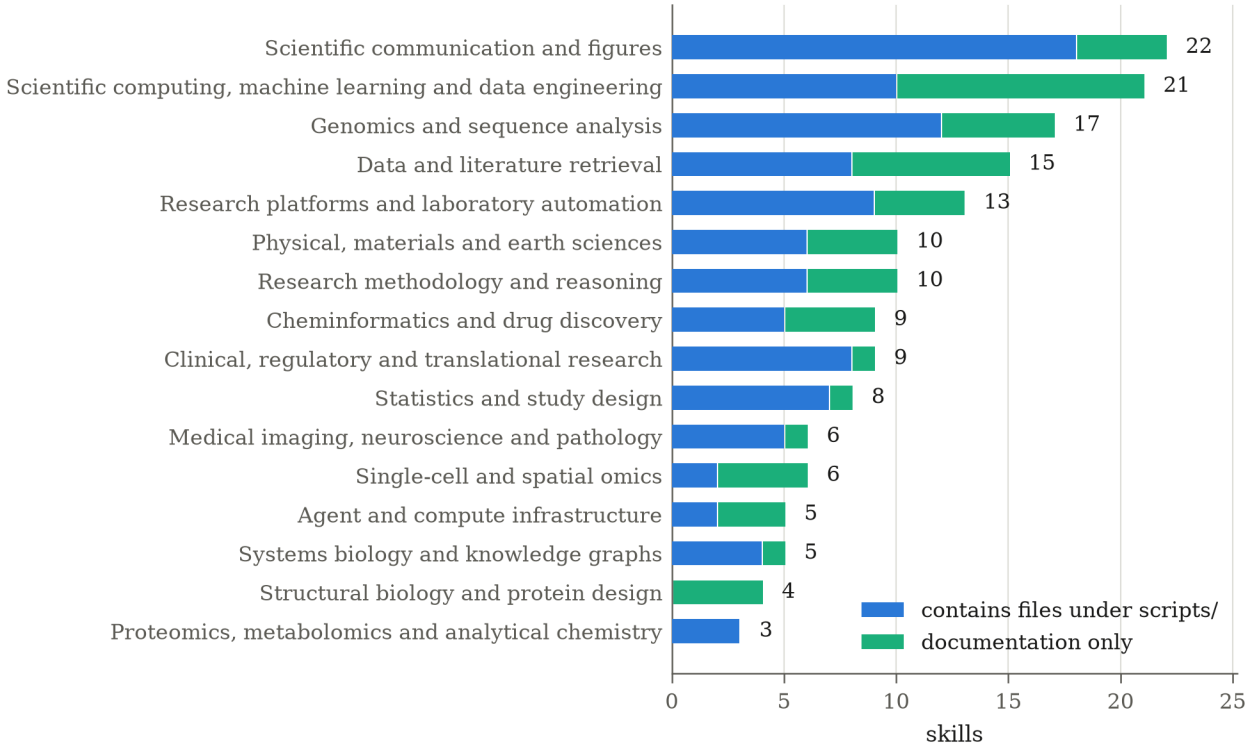


Figure 2: Skills per category, split by whether the skill ships files under `scripts/`. Every such skill has a corresponding test-suite directory. The existence of a suite does not establish that every helper or code path is exercised; a documentation-only skill leaves implementation to the agent.

PyDESeq2 [37] for differential expression; their content is closest to authoritative documentation. *Data retrieval* skills handle identifier resolution and query construction against named resources; one skill, `database-lookup`, alone documents 78 databases across 9 domains, of which 57 document no credential requirement and 18 require a key (Figure 3). *Research platforms and laboratory automation* skills address instrument and platform interfaces. They are the group most likely to require credentials: across the library 29 skills name one of 27 credential-like environment variables. *Method and judgment* skills encode procedure rather than tooling, such as study design and randomisation before data exist, sample-size justification [38], test selection, assumption checking and the interpretation of significance thresholds [39], and uncertainty and unit propagation. This group is the hardest to evaluate because its value is least like documentation lookup. Appendix C describes all four in more detail.

4 Can descriptions distinguish among skills?

Progressive disclosure keeps standing context small only if a 67-token resident description can distinguish one skill from 163 others. The library does not enforce this requirement, and no host reports a selection rate. We measure the lexical distance between descriptions and identify pairs for which we added explicit guidance.

We represent each description as a bag of lower-cased alphanumeric tokens that are at least 3 characters long. We remove a 27-word function-word list, weight the remaining terms by log frequency times inverse document frequency, and calculate cosine similarity for all 13,203 pairs. This is a lexical proxy for confusability. It locates pairs built from the same words; it does not establish

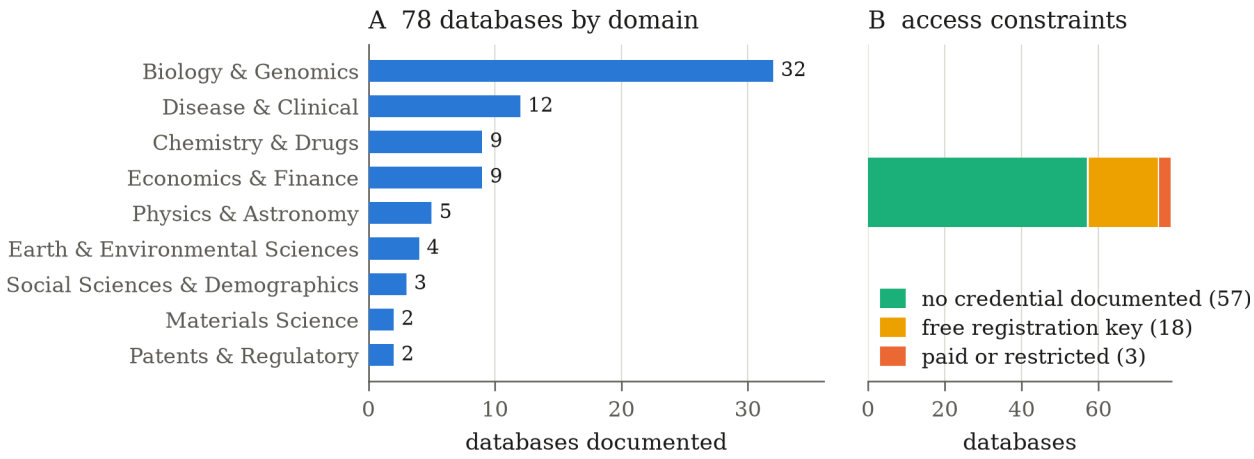


Figure 3: The 78 databases documented by the `database-lookup` skill at tag v2.65.0. **(A)** Databases by domain, using the skill's own grouping, which we checked against the reference files on disk. **(B)** Access constraints from the skill's own tables: 57 databases document no credential requirement, 18 require a free-registration key, and 3 are paid or otherwise restricted; each restricted database has a named free alternative.

that any host would confuse them, as Section 6 makes explicit. Because the preprocessing is our convention, Appendix G reports how far the ranking moves under three alternatives, including no stop list.

Most descriptions are lexically well separated. The median skill's nearest neighbour has a cosine similarity of 0.15, and the ninetieth percentile is 0.26 (Figure 4A), so a typical skill has no lexically close competitor. The difficult cases lie in the upper tail. The closest pair in the library, `datamol` and `rdkit`, reaches 0.40. The next most similar pairs also connect adjacent work in the same domain: two cheminformatics toolkits, two study-design skills, two quantum-computing frameworks and two plotting libraries (Figure 4B). Their lexical overlap reflects adjacent work, not authoring errors. In such cases, a 67-token description must distinguish the skills on its own.

We add cross-references to some similar descriptions. In all, 33 descriptions name another skill directly. Together, they contain 56 pointers connecting 44 distinct pairs, of which 12 are reciprocal. These pointers are routing hints in the resident tier: they tell the agent when to prefer a neighbouring skill. They are concentrated among similar descriptions. The mean cosine of a pair joined by a pointer is 0.185, compared with 0.013 for a pair without one, an enrichment of about 14 times. Pointers cover 17 of the twenty closest pairs, 26 of the fifty closest, and 33 of the hundred closest (Figure 4C). We found many of the closest collisions ourselves, but fewer pairs carry a pointer farther down the ranking.

Some highly similar pairs have no guard. At 0.36, `infographics` and `scientific-schematics` are the closest unguarded pair. They rank as high as the closest guarded pairs, not down in the tail, yet neither description mentions the other. Because each guard is authored per skill rather than derived from the corpus, coverage depends on whether the description author notices the collision. Running this measurement on each release would identify such pairs systematically.

Our continuous integration also computes a different but related signal. The third-party security scanner we run has a rule that reports skills with complementary descriptions, and it flags 301 distinct pairs. That rule keys on shared *function* words, so the pairs it flags have a mean cosine of 0.020, compared with 0.013 for unflagged pairs. It catches 0 of the twenty closest pairs in the library. The two measurements are not substitutes. Appendix E treats the scanner's output as the

heuristic it is.

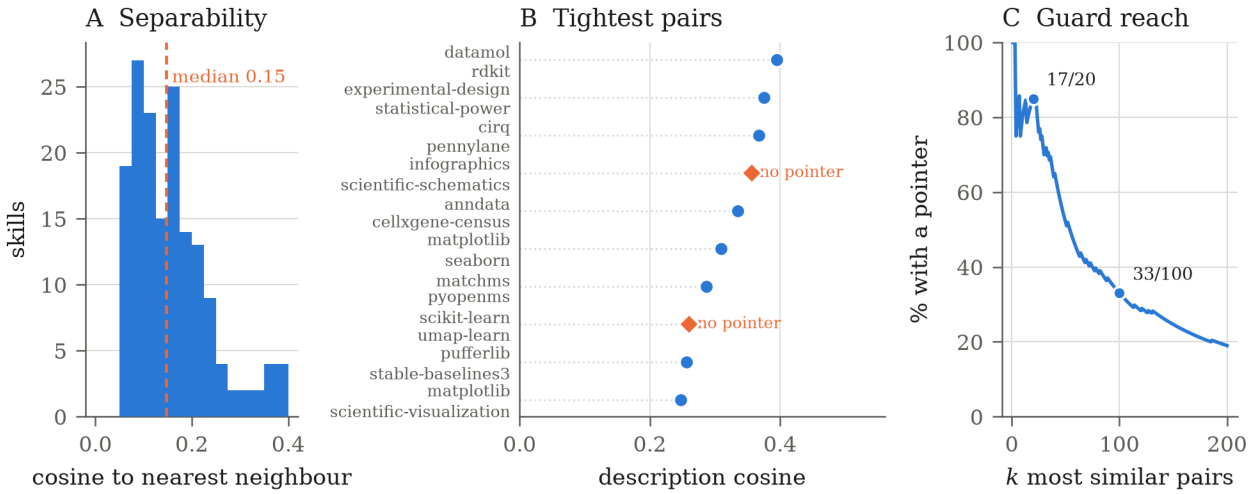


Figure 4: Lexical separability of the always-resident descriptions at tag v2.65.0. **(A)** Each skill's cosine to its nearest neighbour; most descriptions have no lexically close competitor. **(B)** The ten closest pairs, marked by whether either description points to the other. **(C)** The share of the k most similar pairs carrying such a pointer. Cosine is a lexical proxy for confusability, not a measurement of any host's selection behaviour.

5 Does what gets selected fit?

The corpus reveals selection *cost*, even though it cannot reveal selection accuracy. We document 46 worked examples in `docs/examples.md`; each names the skills it draws on, and together they contain 472 skill slots covering 162 of the 163 skills. They are our only statement of how many skills a realistic task is expected to need. The median example names 10 skills; the smallest names 5 and the largest names 16. We treat each example as a session to translate the per-skill token counts in Section 2 into context-window costs.

The instruction files fit comfortably. After including the resident tier, the median documented workflow costs 47,706 tokens, or 23.9% of a 200,000-token window, and the largest costs 62,476. 0 of the 46 workflows exceed the window (Figure 5A). Loading every reference file of every named skill changes the result. The median rises to 225,498 tokens, or 112.7% of the same window, and the largest rises to 466,131. 29 of the 46 workflows then exceed the window.

The gap between the two series measures how much context progressive disclosure can save on these documented workflows rather than on a hypothetical workload. The upper series loads material the design is meant to leave on disk, so it is an upper bound by construction. For 29 of the 46 workflows, the upper series exceeds the 200,000-token window, so those workflows still need references to be loaded selectively.

The remaining context window sets a budget for reference material. After loading a workflow's descriptions and instruction files, the remaining tokens are available for references. The median documented workflow can afford 85% of its own reference material, the lower quartile can afford 61%, and the most demanding workflow can afford 35% (Figure 5B). Of the 46, 17 can afford all their reference material; the others require selective loading, including 5 that can afford under half.

These figures count documentation text only and exclude scripts, assets, tool output and whatever framing a host wraps around each entry, so they understate the amount of context a session would hold. Each workflow total includes every resident description plus the complete instruction files, so

an activated skill's own description is counted twice. That extra copy is a median of 67 tokens; we retain it because it biases the total upward.

We recommend installing only the skills relevant to a project rather than the full library at once. Every installed skill adds an always-resident description and another routing candidate even if it is never used; limiting the installed set bounds both standing context and the number of candidates the host must distinguish. Whether a smaller installation improves selection accuracy remains untested.

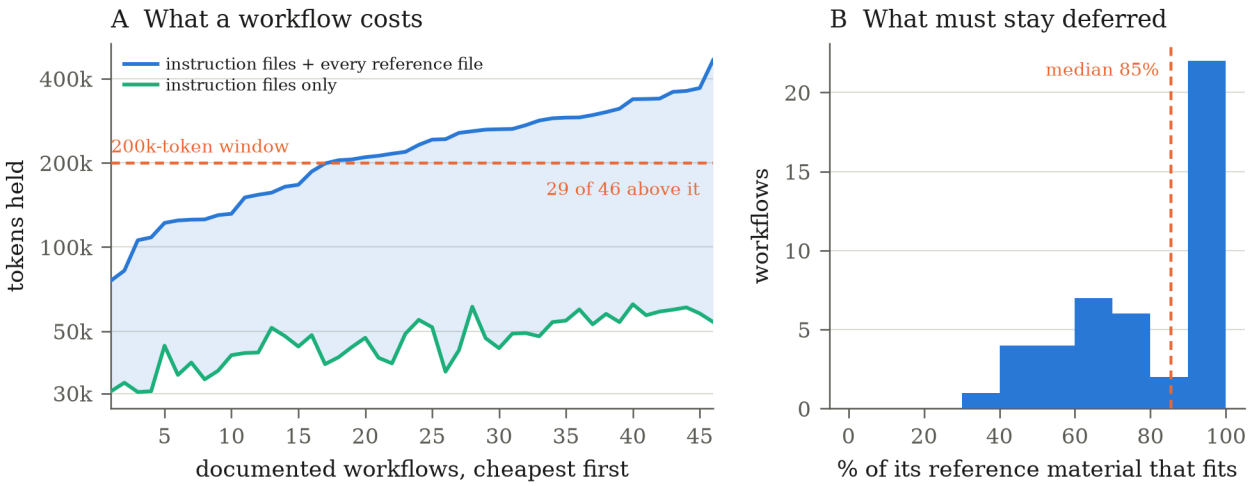


Figure 5: Token costs for the library's 46 documented workflows at tag v2.65.0, compared with a 200,000-token reference window. **(A)** Cost with the resident tier and instruction files, and after adding every reference file of every named skill; the shaded gap is the deferred tier. The upper series is an upper bound by construction. **(B)** The share of its own reference material each workflow could load before exhausting the window. Both panels count documentation text only, excluding scripts, assets, tool output and host wrappers.

6 Limitations

No task-level evaluation. We describe the library, its automated checks and two corpus properties, but do not show that giving an agent these skills improves its scientific work. A controlled study would hold the agent, model, tools and task fixed, vary only the installed skills, and score outcomes against answer keys based on published conventions rather than on the skill text. Matched evaluations with and without skills now exist for other collections [24, 25, 40]. A re-analysis of a controlled offensive-cybersecurity study finds the benefit collapsing in that domain [29]. Its authors propose that strict, schema-validated tool observations can supply the procedural correction that a skill would otherwise provide. Such studies also risk maintainer selection bias in their tasks, which statistical care cannot remove. If a capable model already completes the tasks without skills, there is no headroom left to show a benefit, and skipping that check can yield a confident null that reflects the task set rather than the skills.

Selection is measured on the corpus, not in an agent. Section 4 maps lexical confusability and authored guards, not how often a host selects correctly. That rate would require labelled task-to-skill pairs, a fixed host and a scored selection step. Lexical cosine is computable and auditable, but remains a proxy: descriptions can share vocabulary without confusing a model, or share none while competing for the same task. It maps lexical similarity but does not estimate an

error rate. One study finds that skill benefits can shrink toward the no-skill score when an agent retrieves skills from a large uncurated collection [28]; another reports that flat retrieval can degrade as libraries grow [27]. Their different library sizes and curation methods do not predict a rate here, but show why one must be measured.

A selected skill can still fail scientifically. Selection does not show that an agent followed the skill: it may omit or reorder steps, skip a stop condition, or load only part of the references. Even faithful execution can be wrong when the study violates the procedure's assumptions. The task or data may omit relevant details about study design, sampling, measurement, preprocessing, population or estimand, and a skill cannot apply conditions the host does not reveal. Skills and their dependencies can also become stale as standards, software, APIs, database schemas and reference data change. A pin records the available instructions, not their correctness or currency. Reproducing a run also requires the relevant data, software and environment details, plus database access dates where applicable. We measure neither step-level adherence nor contextual sufficiency, and do not audit every procedure or dependency against current domain guidance. Skills structure scientific judgment; they do not replace expert review.

Reach is not use. The installed-base estimate in Appendix H is the only number not derived from the pinned tree and the least reproducible: it uses a maintainer-only rolling window that GitHub does not archive. It counts clients that fetch the library, not scientists or sessions in which an agent used a skill. We include it as evidence of reach with a stated method and failure modes rather than using a company-reported figure with neither. It is not evidence of benefit, and nothing else in the paper rests on it.

Reported context costs are lower bounds and asymmetric. The budget figures in Section 5 count only documentation text and exclude host framing, scripts, assets and tool output whose context exposure depends on the host and task, so a real session uses more context than we report. The costs are asymmetric: an unselected skill's description consumes context on every turn, but its procedural content is read only when used.

Host portability is intended, not tested. Conformance to a shared layout does not show that hosts discover, select or execute a skill identically: wrappers, install locations, metadata, tool names and permission models differ. A compatibility claim would require a host-by-skill matrix that this paper does not provide.

Uneven review and incomplete CI coverage. 32 of 163 skills do not list us as their author, including 4 with no named author; the latter are unattributed, not known to be external. Our own security guidance warns that community contributions may receive less review than the skills we write, so the corpus is not uniformly vetted. CI also leaves conventions unenforced: the structural contract covers only 105 skills, leaving 2 broken local paths outside its gate (Appendix E).

Our measurements depend on stated conventions. Our choices determine the 16-category taxonomy, the lexical scope-candidate rule and adjudication, the heading-role regexes, the credential-name rule, the routing preprocessing and the workflow parse. Only the adjudicated scope positives are a lower bound. Heading roles and credential variables are lexical counts; the other items are analysis conventions, and every token figure depends on the fixed tokenizer. We describe each rule

rather than publishing its word lists, so these measurements can be approximated from the tagged tree but not reproduced exactly; Appendix G varies the routing rules and reports the effect.

A fixed snapshot of a changing library. The most recent 100 releases returned by the API span 311 days, about 9.7 releases per month, but that page is a recent window rather than the complete history. Every count describes tag v2.65.0 and will differ from later releases, so we state the pin and regenerate each number from the tree.

7 Availability and citation

Scientific Agent Skills is available at `https://github.com/K-Dense-AI/scientific-agent-skills` under the MIT licence. This paper describes tag v2.65.0, released 2026-08-29, at commit `f6fcafeb1cc8c82eca0160a18bc41c38427b8e0f`. At the time of writing (2026-08-29) the repository has 37,905 GitHub stars, and over the 14 days ending 2026-08-27 it recorded 161,210 clones from 15,682 distinct cloners and 61,901 page views from 21,268 distinct visitors. Appendix H estimates a pool of roughly 36,000 distinct cloning clients behind those counts. These values are proxies for reach. We report them with their window and method because the traffic endpoint is maintainer-only and its window is not archived. These metrics do not count people or agent sessions in which a skill was selected and used.

Work that uses these skills should cite this paper together with the release tag it used. The library's contents change between releases, so a citation without a tag does not identify what was run. A paper reporting a result obtained with a specific skill should also name that skill and its `metadata.version`, which is maintained independently of the repository version. For skills in the clinical, regulatory or imaging categories, it should also state the scope limits in that particular skill (Appendix F).

We do not release an analysis package with this paper. Every corpus-derived quantity is computed from the public repository tree at tag v2.65.0 with NumPy [41] and Matplotlib [42], so the input to every measurement is available even though our scripts are not. The convention behind each number is described where the number is reported, and Appendix A gives the tokenizer and corpus definition every token figure depends on. Recomputation would still not validate our labels or interpretation: Appendix E shows that a passing tree does not put every skill under the same gates.

Acknowledgements. The skills described here document a large body of open-source scientific software whose authors did the harder work. 4 document-format skills are vendored from Anthropic's public skills repository and are used under its terms. The frontispiece was produced by an image model from our prompt; the numbered figures are drawn in code from the measured data.

Competing interests. Timothy Kassis, Yuhuan He, Darshil Patel and Aubrey M. Brueckner are employees of K-Dense, Inc., which maintains Scientific Agent Skills. Vinayak Agarwal is a former employee of K-Dense, Inc. The authors are therefore describing an artifact developed and maintained by their current or former employer.

References

[1] Yoav Benjamini and Yosef Hochberg. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society Series B:*

Statistical Methodology, 57(1):289–300, January 1995. ISSN 1467-9868. doi: 10.1111/j.2517-6161.1995.tb02031.x.

[2] Jeffrey T. Leek, Robert B. Scharpf, Héctor Corrada Bravo, David Simcha, Benjamin Langmead, W. Evan Johnson, Donald Geman, Keith Baggerly, and Rafael A. Irizarry. Tackling the widespread and critical impact of batch effects in high-throughput data. *Nature Reviews Genetics*, 11(10):733–739, September 2010. ISSN 1471-0064. doi: 10.1038/nrg2825.

[3] UCSC Genome Browser. Format and coordinate conventions. `https://genome.ucsc.edu/FAQ/FAQformat.html`, 2026. Accessed 2026-08-30.

[4] Nan Xuan Lin, Stuart Logan, and William Edward Henley. Bias and Sensitivity Analysis When Estimating Treatment Effects from the Cox Model with Omitted Covariates. *Biometrics*, 69(4):850–860, November 2013. ISSN 1541-0420. doi: 10.1111/biom.12096.

[5] Mark Ziemann, Yotam Eren, and Assam El-Osta. Gene name errors are widespread in the scientific literature. *Genome Biology*, 17(1):177, August 2016. ISSN 1474-760X. doi: 10.1186/s13059-016-1044-7.

[6] Aubrey Brueckner, Darshil Patel, Yuhuan He, and Timothy Kassis. K-Bench: measuring model performance on real scientific agent requests, 2026. arXiv:2608.21601 [cs.AI].

[7] Nathalie Percie du Sert, Viki Hurst, Amrita Ahluwalia, Sabina Alam, Marc T. Avey, Monya Baker, William J. Browne, Alejandra Clark, Innes C. Cuthill, Ulrich Dirnagl, Michael Emerson, Paul Garner, Stephen T. Holgate, David W. Howells, Natasha A. Karp, Stanley E. Lazic, Katie Lidster, Catriona J. MacCallum, Malcolm Macleod, Esther J. Pearl, Ole H. Petersen, Frances Rawle, Penny Reynolds, Kieron Rooney, Emily S. Sena, Shai D. Silberberg, Thomas Steckler, and Hanno Würbel. The ARRIVE guidelines 2.0: Updated guidelines for reporting animal research. *PLOS Biology*, 18(7):e3000410, July 2020. ISSN 1545-7885. doi: 10.1371/journal.pbio.3000410.

[8] Stephen A Bustin, Vladimir Benes, Jeremy A Garson, Jan Hellemans, Jim Huggett, Mikael Kubista, Reinhold Mueller, Tania Nolan, Michael W Pfaffl, Gregory L Shipley, Jo Vandesompele, and Carl T Wittwer. The MIQE Guidelines: Minimum Information for Publication of Quantitative Real-Time PCR Experiments. *Clinical Chemistry*, 55(4):611–622, April 2009. ISSN 1530-8561. doi: 10.1373/clinchem.2008.112797.

[9] Tamar R Makin and Jean-Jacques Orban de Xivry. Ten common statistical mistakes to watch out for when writing or reviewing a manuscript. *eLife*, 8:e48175, October 2019. ISSN 2050-084X. doi: 10.7554/elife.48175.

[10] Agent Skills. Agent skills specification. `https://agentskills.io/specification`, 2026. Accessed 2026-08-30.

[11] K-Dense Inc. Scientific agent skills. `https://github.com/K-Dense-AI/scientific-agent-skills`, 2026. Version 2.65.0, commit f6fcafe; accessed 2026-08-30.

[12] Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, December 2023. ISSN 1476-4687. doi: 10.1038/s41586-023-06792-0.

[13] Orion Li, Vinayak Agarwal, Summer Zhou, Ashwin Gopinath, and Timothy Kassis. K-Dense Analyst: Towards Fully Automated Scientific Analysis, 2025. arXiv:2508.07043 [cs.AI].

[14] Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery, 2024. arXiv:2408.06292 [cs.AI].

[15] MIMS Harvard. Tooluniverse ai agent skills. `https://zitniklab.hms.harvard.edu/ToolUniverse/guide/skills_showcase.html`, 2026. Accessed 2026-08-30.

[16] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?, 2023. arXiv:2310.06770 [cs.CL].

[17] Minyang Tian, Luyu Gao, Shizhuo Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, Hao Tong, Kha Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Yanyu Xiong, Shengzhu Yin, Minhui Zhu, Kilian Lieret, Yanxin Lu, Genglin Liu, Yufeng Du, Tianhua Tao, Ofir Press, Jamie Callan, Eliu Huerta, and Hao Peng. SciCode: A Research Coding Benchmark Curated by Scientists, 2024. arXiv:2407.13168 [cs.AI].

[18] Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. ScienceAgentBench: Toward Rigorous Assessment of Language Agents for Data-Driven Scientific Discovery, 2024. arXiv:2410.05080 [cs.CL].

[19] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, 2020. arXiv:2005.11401 [cs.CL].

[20] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools, 2023. arXiv:2302.04761 [cs.CL].

[21] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models, 2022. arXiv:2210.03629 [cs.CL].

[22] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models, 2023. arXiv:2305.16291 [cs.AI].

[23] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. ISSN 2307-387X. doi: 10.1162/tacl_a_00638.

[24] Shuaike Shen, Wenduo Cheng, Mingqian Ma, Alistair Turcan, Martin Jinye Zhang, and Jian Ma. SKILLFOUNDRY: Building Self-Evolving Agent Skill Libraries from Heterogeneous Scientific Resources, 2026. arXiv:2604.03964 [cs.AI].

[25] Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia

Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks, 2026. arXiv:2602.12670 [cs.AI].

[26] George Ling, Shanshan Zhong, and Richard Huang. Agent Skills: A Data-Driven Analysis of Claude Skills for Extending Large Language Model Functionality, 2026. arXiv:2602.08004 [cs.SE].

[27] Yubo Li. Dynamic Agent Skills: A Lifecycle Survey and Taxonomy of Evolving Skill Libraries, 2026. arXiv:2607.10113 [cs.AI].

[28] Yujian Liu, Jiabao Ji, Li An, Tommi Jaakkola, Yang Zhang, and Shiyu Chang. How Well Do Agentic Skills Work in the Wild: Benchmarking LLM Skill Usage in Realistic Settings, 2026. arXiv:2604.04323 [cs.CL].

[29] Samuel Jacob Chacko, James Hugglestone, Chashi Mahiul Islam, and Xiuwen Liu. When Skills Don't Help: A Negative Result on Procedural Knowledge for Tool-Grounded Agents in Offensive Cybersecurity, 2026. arXiv:2605.20023 [cs.AI].

[30] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 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*, CCS '23, page 79–90. ACM, November 2023. doi: 10.1145/3605764.3623985.

[31] Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. SoK: Agentic Skills – Beyond Tool Use in LLM Agents, 2026. arXiv:2602.20867 [cs.CR].

[32] Yi Liu, Weizhe Wang, Ruitao Feng, Yao Zhang, Guangquan Xu, Gelei Deng, Yuekang Li, and Leo Zhang. Agent Skills in the Wild: An Empirical Study of Security Vulnerabilities at Scale, 2026. arXiv:2601.10338 [cs.CR].

[33] Zhiyuan Li, Jingzheng Wu, Xiang Ling, Xing Cui, and Tianyue Luo. Towards Secure Agent Skills: Architecture, Threat Taxonomy, and Security Analysis, 2026. arXiv:2604.02837 [cs.CR].

[34] Cisco AI Defense. Skill scanner. `https://github.com/cisco-ai-defense/skill-scanner`, 2026. Version 2.0.13; accessed 2026-08-30.

[35] Florian Holzbauer, David Schmidt, Gabriel Gegenhuber, Sebastian Schrittwieser, and Johanna Ullrich. Context Matters: Repository-Aware Security Analysis of the Agent Skill Ecosystem, 2026. arXiv:2603.16572 [cs.CR].

[36] F. Alexander Wolf, Philipp Angerer, and Fabian J. Theis. SCANPY: large-scale single-cell gene expression data analysis. *Genome Biology*, 19(1):15, February 2018. ISSN 1474-760X. doi: 10.1186/s13059-017-1382-0.

[37] Boris Muzellec, Maria Teleńczuk, Vincent Cabeli, and Mathieu Andreux. PyDESeq2: a python package for bulk RNA-seq differential expression analysis. *Bioinformatics*, 39(9):btad547, September 2023. ISSN 1367-4811. doi: 10.1093/bioinformatics/btad547.

[38] Katherine S. Button, John P. A. Ioannidis, Claire Mokrysz, Brian A. Nosek, Jonathan Flint, Emma S. J. Robinson, and Marcus R. Munafò. Power failure: why small sample size undermines the reliability of neuroscience. *Nature Reviews Neuroscience*, 14(5):365–376, April 2013. ISSN 1471-0048. doi: 10.1038/nrn3475.

[39] Ronald L. Wasserstein and Nicole A. Lazar. The ASA Statement on p-Values: Context, Process, and Purpose. *The American Statistician*, 70(2):129–133, April 2016. ISSN 1537-2731. doi: 10.1080/00031305.2016.1154108.

[40] Maksim Shaposhnikov, Nicolas Fortuin, Simon Stipcich, Maria I. Gorinova, Amy Heineike, and Rob Willoughby. A Framework for Evaluating Agentic Skills at Scale, 2026. arXiv:2606.17819 [cs.SE].

[41] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2649-2.

[42] John D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. ISSN 1521-9615. doi: 10.1109/mcse.2007.55.

[43] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4):316–319, April 2017. ISSN 1546-1696. doi: 10.1038/nbt.3820.

[44] Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, August 2012. ISSN 1367-4803. doi: 10.1093/bioinformatics/bts480.

[45] Björn Grüning, Ryan Dale, Andreas Sjödin, Brad A. Chapman, Jillian Rowe, Christopher H. Tomkins-Tinch, Renan Valieris, and Johannes Köster. Bioconda: sustainable and comprehensive software distribution for the life sciences. *Nature Methods*, 15(7):475–476, July 2018. ISSN 1548-7105. doi: 10.1038/s41592-018-0046-7.

[46] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J.G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A.C 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The

FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1): 160018, March 2016. ISSN 2052-4463. doi: 10.1038/sdata.2016.18.

[47] Arfon M. Smith, Daniel S. Katz, and Kyle E. Niemeyer. Software citation principles. *PeerJ Computer Science*, 2:e86, September 2016. ISSN 2376-5992. doi: 10.7717/peerj-cs.86.

[48] Greg Wilson, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K. Teal. Good enough practices in scientific computing. *PLOS Computational Biology*, 13(6): e1005510, June 2017. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1005510.

[49] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation, 2023. arXiv:2305.01210 [cs.SE].

[50] Agent Plugins. Agent plugins 1.0.0 specification. `https://agent-plugins.org/`, 2026. Accessed 2026-08-30.

[51] Peter J. A. Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel J. L. de Hoon. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, March 2009. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp163.

[52] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, June 2009. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp352.

[53] RDKit. RDKit: Open-Source Cheminformatics. `https://www.rdkit.org`, 2026. Accessed 2026-08-30.

[54] Michael I Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, 15(12):550, December 2014. ISSN 1474-760X. doi: 10.1186/s13059-014-0550-8.

[55] David Schmotz, Sahar Abdelnabi, and Maksym Andriushchenko. Agent Skills Enable a New Class of Realistic and Trivially Simple Prompt Injections, 2025. arXiv:2510.26328 [cs.LG].

[56] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents, 2024. arXiv:2406.13352 [cs.CR].

A Measurement conventions and the token ledger

Tokenizer and corpus. Every token figure in this paper counts the raw text of a file under the `o200k_base` encoding, with no host wrapper, chat template or system prompt around it. The count is a property of the file, so a host using a different tokenizer would obtain a different figure; the ratios between tiers are the stable part. The documentation corpus is tier two plus tier three: the 482,506 tokens in the 163 complete `SKILL.md` files and the 2,480,674 tokens in the 1,033 files under `references/`, for 2,963,180 tokens in all. The resident tier is not a third addend. A skill's `name` and

`description` live in the header of its own `SKILL.md`, so the 14,246-token resident tier is an excerpt of tier two rather than a quantity beside it, and adding it would count every description twice. Under `references/` we count text files only (`.md`, `.txt`, `.rst`, `.json`, `.yaml` and `.yml`); scripts and binary assets are counted as files but never as documentation tokens, which is why the reported costs are lower bounds. Editor and operating-system artifacts (`.DS_Store`, Python bytecode and similar) are excluded from every file count. The 200,000-token window used throughout is a round reference scale rather than a property of a particular model.

Reproducible practice in computational science. Skills address a familiar problem: known procedures are often not encoded in a reusable form. Workflow engines such as Nextflow [43] and Snakemake [44], curated software distributions [45], the FAIR principles for data [46], software citation practice [47], and guidance on computational craft [48] address the same problem. These mechanisms are complementary: workflow engines make an executable procedure reproducible, distributions make its dependencies installable, and data standards preserve its inputs and outputs. A skill tells an agent how to select, sequence and check those components, while still depending on the same environments and records.

A.1 The full token ledger

Figure A.1 shows what each tier costs, in aggregate and per skill. Figure A.2 shows the arithmetic behind tiering. We use a 200,000-token context window as a round reference scale, not as a property of any particular model. Holding the always-resident descriptions of all 163 skills costs 14,246 tokens, or 7.1% of that window. Holding the same library with complete instruction files inlined would cost 482,506 tokens. At most 89 skills fit when the shortest files are installed first; if each installed skill also contributes all its reference files, that upper bound falls to 38. The full documentation corpus would need 14.8 such windows. These counterfactuals measure documentation text only and exclude scripts, assets, tool output and host wrappers, although those resources may still consume context. The full corpus therefore does not fit in one reference window.

In practice, a session holds the resident tier plus content from any skills it activates. With everything installed and one skill activated, the median total is 17,103 tokens, or 8.6% of the reference window. This total includes every description and the complete instruction file for one skill. Because that skill's description is already resident, the total counts it twice, adding a median of 67 tokens. We retain that double count because it biases the estimate upward. If every activated skill were read together with its entire `references/` directory, the median session total would instead be 29,440 tokens. The heaviest skill would reach 231,263 tokens and exceed the window by itself. We compute the two median session totals from their respective per-skill distributions rather than by adding the component medians reported above. We cost the deferred tier file by file rather than directory by directory: the median reference file is 2,000 tokens and the largest is 189,746, so the choice of file can matter more than the choice of skill.

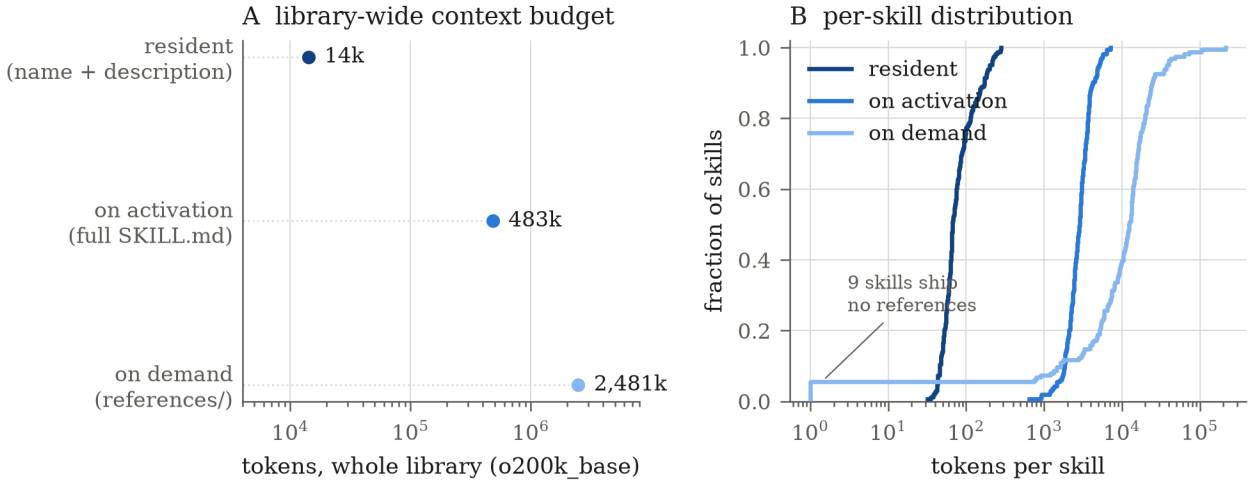


Figure A.1: The three disclosure tiers at tag v2.65.0. **(A)** What the whole library costs in each tier, on a log scale: the always-resident descriptions are 2.9% of the instruction bodies, and those bodies are in turn a fifth of the deferred reference material. **(B)** The same three tiers as per-skill distributions. The deferred tier is heavy-tailed: a handful of skills carry most of it, which is why the totals alone would be misleading.

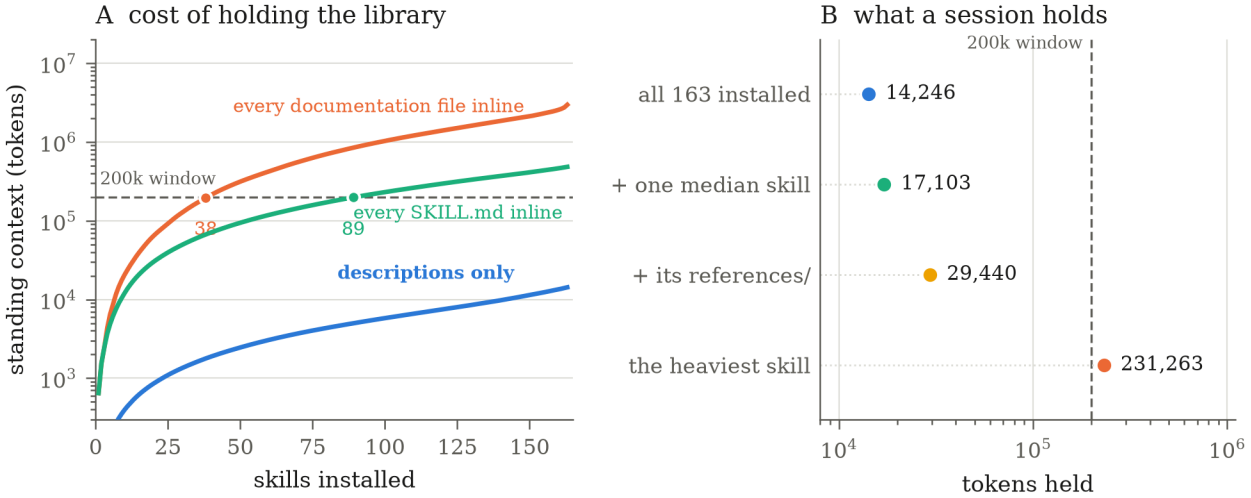


Figure A.2: The cost of the library at tag v2.65.0, against a 200,000-token reference window. **(A)** Standing context as skills are installed, cheapest first, under three representations of the same material; the marked points are the last skill that fits. Cheapest-first is the ordering most favourable to the two inlined variants, so those marks are upper bounds on how many skills either could hold. **(B)** What a session holds in practice. Both panels use a log scale; in panel B, read a value from the vertical position of its dot rather than the apparent length of its bar.

B What a skill contains

Skills differ substantially in what they ship alongside the instruction file: some bundle reference documents, scripts or assets, while others do not. Of 163 skills, 154 ship reference documents, 105 contain 651 files under `scripts/`, and 34 ship static assets. Those 651 comprise 527 Python files, 2 shell scripts and 122 XML or schema support files, so the directory total is not an executable-script count. The most common composition is a skill with both references and scripts (68 skills); 52 ship references alone, 31 ship all three subdirectories, and 3 skills each consist only of a single instruction file (Figure B.1).

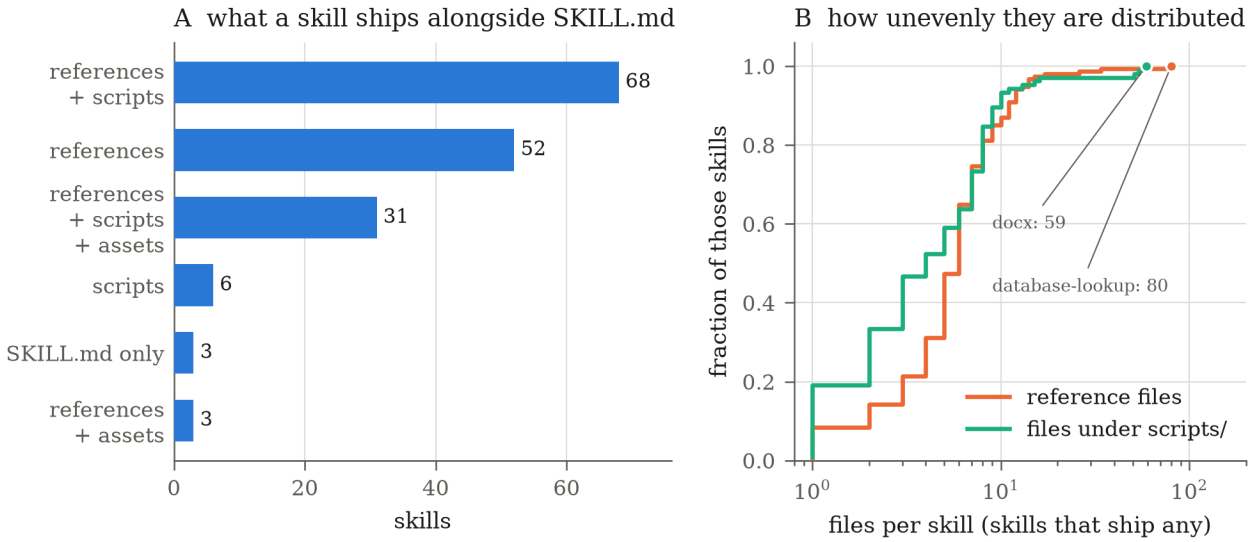


Figure B.1: What skills ship at tag v2.65.0. **(A)** The combinations of optional subdirectories. **(B)** Reference-file counts among skills with references and script-file counts among skills with scripts. Both distributions are heavy-tailed, so a few skills account for much of the library totals. The median skill with references ships 6 reference files; the median script-bearing skill ships 4 script files.

A documentation-only skill explains what to do but leaves implementation to the agent, so the output is only as reliable as the code the agent generates. Rigorous testing shows that this generation is less reliable than a passing example suggests [49]. A script-bearing skill can keep fragile or repetitive logic out of code generated during a session. Every such skill has a corresponding suite, but the existence of that directory does not by itself establish coverage of every helper or code path. Bundled scripts also add risk. An ecosystem-scale survey finds that skills with executable scripts are markedly more likely to carry a vulnerability than instruction-only skills [32]. Shipped code therefore requires the testing and scanning described in Appendix E. Our contributor guide identifies this as the purpose of `scripts/` and requires every script-bearing skill to carry a test suite (Appendix E).

B.1 The instruction file as a document

An agent always reads the instruction file once it selects a skill, so this file contains the skill's main guidance. The documents share broad structural patterns but no common template. Across the library they use 844 distinct level-2 headings, a median of 10 per file, but those headings serve a small set of recurring roles (Figure B.2A). 104 of 163 skills carry a section telling the agent when the skill applies, 126 give an explicit procedure or decision rule, 84 carry validation checks or caveats,

and 143 point at their deferred material. The grouping into roles is ours. It uses a keyword rule over headings, so a skill that answers the same question under an unmatched heading is not counted.

The median complete file is 285 lines against the 500-line recommendation (Figure B.2B), and carries 8 fenced code blocks. Across the library there are 1,489 such blocks, of which 760 are Python and 625 are shell, with 67 declaring no language at all (Figure B.2C). These counts show how often worked commands appear, not how much of the corpus is prose. They are consistent with the contributing guidance's preference for concrete workflows and commands over background explanation.

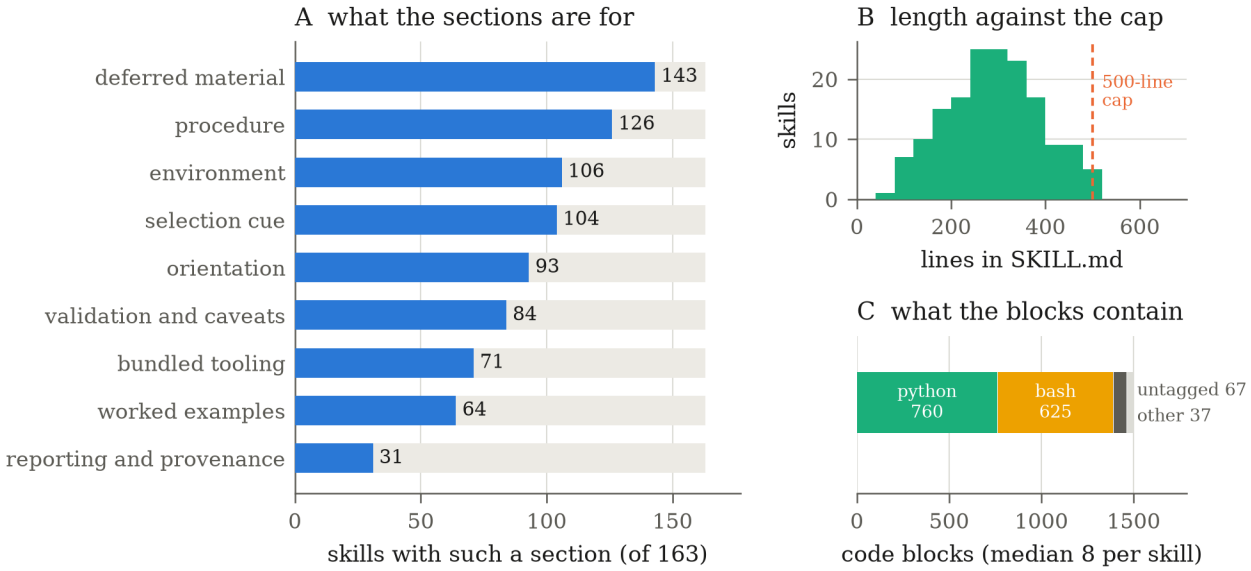


Figure B.2: Inside the file an agent reads on activation, at tag v2.65.0. **(A)** Level-2 sections grouped by the role they play; roles are not mutually exclusive. The grouping uses a keyword rule over headings rather than a semantic judgment. **(B)** File length against the repository's 500-line cap. **(C)** Languages declared by fenced code blocks; the segments account for all 1,489 blocks, including the 67 that declare no language.

B.2 Declared capability and provenance

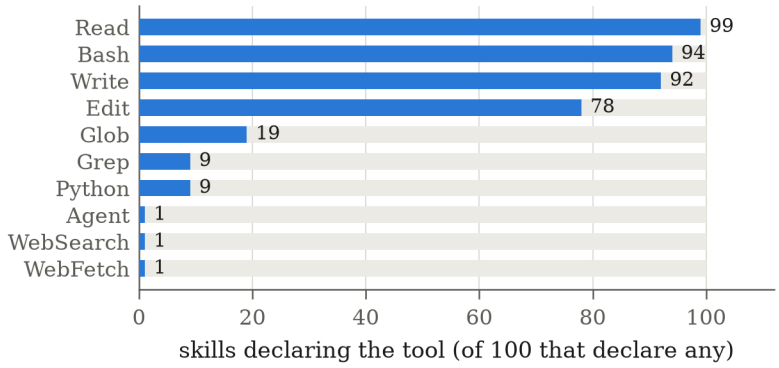


Figure B.3: The host capabilities skills declare in `allowed-tools`, for the 100 of 163 skills that declare any. The field is optional and advisory rather than a sandbox, but where it is present it is the library's own statement of what a skill assumes it may do.

100 skills declare an `allowed-tools` string, and the values are concentrated in four capabilities: 99 ask for file reading, 94 for a shell, 92 for file writing and 78 for editing (Figure B.3). Almost all expect an agent that can read and write files and run commands. A conversational assistant may not meet that expectation, which affects whether the library fits a given host. The field is optional, experimental and honoured only by hosts that implement it. These declarations characterise only the skills that use the field, not the whole library. The other 63 skills state no capability requirement. An undeclared field does not mean the skill needs no tools.

Skills are versioned individually in `metadata.version`, independently of the repository release, and all 163 carry a version string at this tag. We do not interpret the major-version distribution as a maturity measure: non-breaking revisions can be extensive, and the field records declared compatibility rather than review depth. The `license` field is optional and 4 skills leave it unset, so for those the repository's own MIT licence is the only statement covering them. Authorship is recorded for 159 of the 163 skills. We divide all skills into four groups (Figure B.4). 131 skills are attributed to us and 24 to outside contributors, across 17 distinct names; 4 are document-format skills vendored from another public skills repository, and 4 name no author. For the last group, readers lack the per-skill attribution needed to judge provenance. This split means the library is not a uniformly reviewed corpus, and our own security guidance says as much (Appendix E).

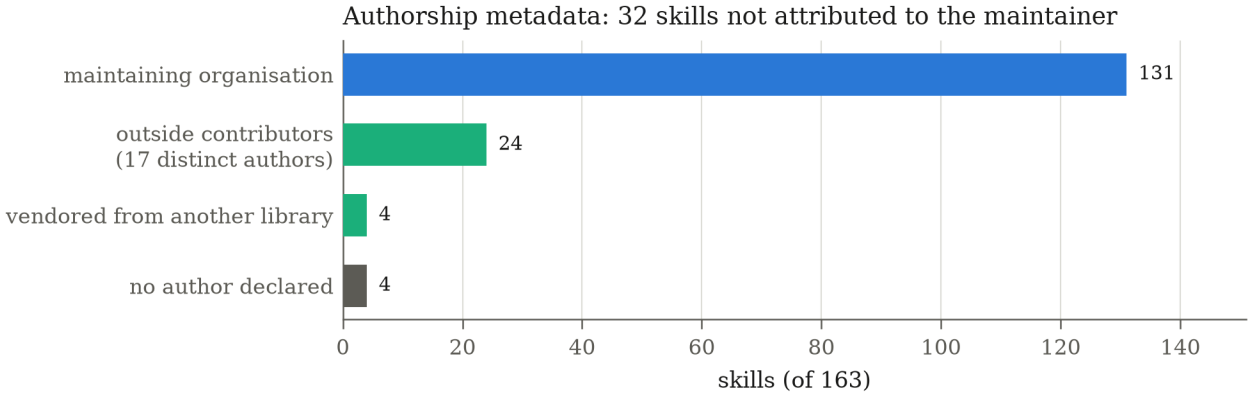


Figure B.4: Authorship metadata at tag v2.65.0, grouped rather than named. Per-skill attribution is in `metadata.skill-author`, which 4 skills leave unset. Version numbers are not shown because they do not measure review depth or maturity.

B.3 Portability

The format uses a file layout rather than a runtime interface so that different agent hosts can discover the same skill directory. This paper does not test a host-compatibility matrix, so portability here means conformance to that shared layout rather than identical behaviour across clients. We also package the library as an Agent Plugins 1.0.0 [50] bundle, with a root manifest and the `skills/` tree. Plugin-capable clients can discover that package, but distribution and installation remain client-specific rather than part of either standard.

Portability still depends on the host. Hosts differ in install paths, in discovery settings, and in which optional header fields they support, so a skill's behaviour is not fully determined by its own files. The clearest case is a host-specific mapping nested under `metadata`: at this tag 25 of 163 skills carry one. The reference validator accepts these mappings, but the specification text defines `metadata` as a mapping from string keys to string values. Because the validator and specification disagree, a host may consume the extension, ignore it or reject it. Our contributor guide further

warns that a failed eligibility check in that mapping can hide a skill from the agent, and that writing the mapping as a JSON string silently disables the gating and credential injection it was meant to provide. At this tag 116 of 163 skills declare a `compatibility` string naming their environment requirements. What they declare they may do is the `allowed-tools` field discussed above, not this string.

B.4 The skill directory, and a skill as it ships

Figure B.5 separates the agent payload from the repository infrastructure. Everything under `skills/<name>/` ships to the agent, while test suites, shared checks, generated diagrams and CI workflows do not. Listing 1 shows the verbatim header and opening of one `SKILL.md` as it ships at this tag. Of everything in the listing, only `name` and `description` are resident before selection.

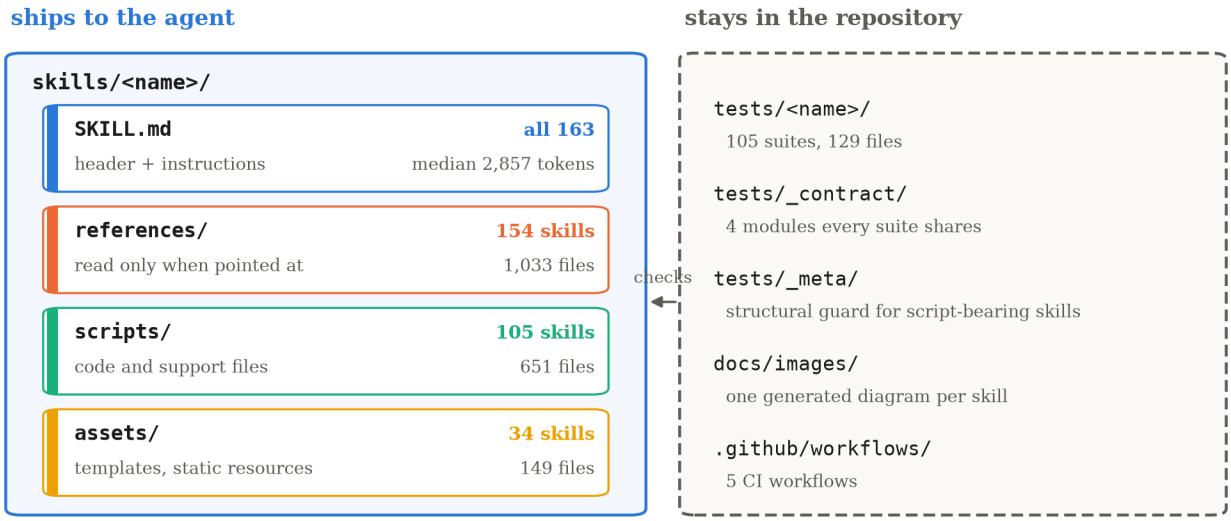


Figure B.5: The skill payload and repository apparatus at tag v2.65.0. Agent-loadable material lives under `skills/<name>/`; test suites, shared checks, generated diagrams and CI workflows live elsewhere in the repository. The tree follows that separation at this pin, although Appendix E shows that the rule excluding in-skill tests is enforced only for script-bearing skills.

The specification and repository guidance recommend keeping instruction files within 500 lines and moving longer material into `references/`. At this tag the longest complete file, including its YAML header, is 496 lines, and 0 files exceed the limit. The specification workflow checks all skills but would report an over-limit file only as a warning. The blocking structural check covers only script-bearing skills, an enforcement gap detailed in Appendix E. Of the 154 skills that ship reference documents, the median ships 6 and the largest ships 80.

```
---
name: pysam
description: Python/HTSlib workflows for genomic files. Use when reading, querying, filtering, or
    writing SAM/BAM/CRAM, VCF/BCF, FASTA/FASTQ, or tabix data with pysam, including pileup,
    coverage, indexing, and CRAM references.
license: MIT
allowed-tools: Read Write Edit Bash
compatibility: Requires Python 3.8-3.14 and pysam 0.24.0. Bundled scripts use local files. CRAM
    decoding may require the matching reference FASTA or an explicitly configured REF_PATH/
    REF_CACHE.
metadata:
  version: "2.0"
  skill-author: K-Dense Inc.
---

# pysam

## Overview

Use pysam for low-level, streaming access to HTSlib-supported genomic formats:

- ‘AlignmentFile‘ and ‘AlignedSegment‘ for SAM/BAM/CRAM
- ‘VariantFile‘, ‘VariantHeader‘, and ‘VariantRecord‘ for VCF/BCF
- ‘FastaFile‘ for indexed FASTA and ‘FastxFile‘ for sequential FASTA/FASTQ
- ‘TabixFile‘ for BGZF-compressed, tabix-indexed BED/GFF/GTF/custom tables
- ‘pysam.samtools‘ and ‘pysam.bcftools‘ for wrapped command dispatchers

Current upstream baseline: **pysam 0.24.0** (27 April 2026), wrapping

[... 306 further lines of instructions ...]
```

Listing 1: A skill in its shipped form: the verbatim header and opening of `skills/pysam/SKILL.md` at tag v2.65.0, with the instruction body truncated. The header carries all six permitted fields; its `name` and `description` are the only parts counted in resident context before selection.

C What the library covers, in detail

The library's documentation groups skills under partly overlapping headings rather than defining a single partition. For reporting, we assigned each of the 163 skills exactly one primary category by its dominant intent; these assignments yield the 16 categories in Table C.1 and Figure 2. The assignment is ours, not the library's. We checked that every skill appears in exactly one category, so the counts in Table C.1 partition the library. A reader who disagrees with a placement can reassign it from the skill names in the tagged tree. Some placements are debatable: we group skills whose purpose is to query a named database by that function rather than by the science of the data they return, and single-purpose analysis packages by the domain they serve rather than the numerical method they use.

For this discussion, we group skills into four cross-cutting kinds. They do not map directly to Table C.1: each can draw on several categories. We give no count for them because their boundaries are ours and differ from the partition reported in the table.

Package workflows. Many skills document how to use a specific scientific Python package: Scanpy [36] for single-cell analysis, Biopython [51] for sequence work, pysam over the SAM/BAM

Table C.1: Primary category assignment of all 163 skills at tag v2.65.0, with how many in each category ship executable scripts and how many ship reference documents. The assignment is ours and is not part of the library.

Category	Skills	With scripts	With references
Scientific communication and figures	22	18	18
Scientific computing, machine learning and data engineering	21	10	21
Genomics and sequence analysis	17	12	16
Data and literature retrieval	15	8	12
Research platforms and laboratory automation	13	9	13
Physical, materials and earth sciences	10	6	10
Research methodology and reasoning	10	6	10
Cheminformatics and drug discovery	9	5	9
Clinical, regulatory and translational research	9	8	9
Statistics and study design	8	7	8
Medical imaging, neuroscience and pathology	6	5	6
Single-cell and spatial omics	6	2	6
Agent and compute infrastructure	5	2	5
Systems biology and knowledge graphs	5	4	4
Structural biology and protein design	4	0	4
Proteomics, metabolomics and analytical chemistry	3	3	3
Total	**163**	**105**	**154**

formats [52], RDKit for cheminformatics [53], PyDESeq2 [37] for differential expression following DESeq2 [54], and comparable coverage in materials science, quantum computing, geospatial analysis and mass spectrometry. These skills are not substitutes for the packages' documentation. They focus on procedural choices that package documentation usually omits: which defaults are wrong for particular data, the order in which operations must occur, and which validity checks precede interpretation.

Data retrieval. A single `database-lookup` skill documents API access to 78 public databases across 9 domains, with one reference file per database. Its 80 files include 2 cross-cutting guides, and its coverage is concentrated in biology and genomics (Figure 3). Its instructions prioritise traceable retrieval over convenience. The instruction body requires an explicit retrieval contract before any call, reconciliation of retrieved counts against expected counts, and a record of endpoints and access dates alongside every result. It also treats API responses as untrusted input, instructing the agent not to follow directives embedded in returned text. Directives embedded in retrieved text form an indirect prompt-injection channel for any retrieval-integrated agent [30]. Databases whose records include user-contributed fields can expose agents to this channel. Dedicated skills cover resources that do not fit a generic pattern, while multi-database wrappers extend the coverage. Our README's headline figure of more than one hundred databases counts those wrappers, while 78 is the number we can verify by counting files.

Research platforms and laboratory automation. 13 skills target systems that hold laboratory data or execute protocols: electronic lab notebooks, sample registries, cloud genomics platforms, microscopy servers, liquid-handling robots and workflow engines. Several require authentication, but the declarations do not support a clean library-wide count of mandatory credentials. Across the library, 29 skills name one of 27 credential-like environment variables ending in `KEY`, `TOKEN`,

SECRET or PASSWORD in either the compatibility string or host-specific metadata. This is a lexical count of declarations, not of requirements: some variables are optional, permit higher rate limits, or provide alternatives to interactive login. Configuration variables such as URLs, ports and contact email addresses are excluded.

Method and judgment. A substantial group of skills encodes procedure rather than tooling. They cover study design and randomisation before data exist, sample-size justification [38], test selection, assumption checking and the interpretation of significance thresholds [39], uncertainty and unit propagation, hypothesis formulation bounded by available evidence, critical appraisal of claims, and manuscript and review preparation with explicit evidence provenance. Compared with documentation lookup, these skills offer the kind of checklist a careful collaborator would apply; that reliance on procedure and judgment also makes them the hardest to evaluate.

D Design rules, and what they cost

The following four repository constraints reflect design choices, not housekeeping rules.

We enforce narrow scope by declining skills. Our contributor guide (AGENTS.md) names categories of skill that we routinely decline: general software-engineering skills, general infrastructure with a scientific example attached, orchestrator skills that route to other skills, and second providers for a service an existing skill already reaches. Our stated reason is selection pressure. Every installed skill competes for the agent's attention on every task, so a broad skill imposes a cost on unrelated tasks, and an orchestrator overlaps every specialist by construction. The resident-tier calculation in Appendix A quantifies this selection pressure: the tier stays small because each description is short. Whether the descriptions are specific enough for reliable selection is not measured here.

Length limits are a disclosure mechanism. The 500-line limit on instruction files is intended to push long material into references/, where it adds no documentation tokens to a session until an agent follows a pointer to it. A skill that ignored the cap would move deferred tokens into the activation tier for every task that selects it.

We keep tests out of the shipped directory. A skill directory contains only what an agent loads, so tests, fixtures and generated artifacts live in a parallel tree (Figure B.5). This separation requires each suite to locate its skill through an explicit anchor path rather than by traversing a relative path. It also lets test coverage grow without enlarging what every host must scan.

One environment per skill, because the pins conflict. We do not install the skills' scientific packages into a shared environment because some of their requirements conflict. At this tag 10 skills need an interpreter older than the project's own, and our guidance identifies NumPy, Zarr, lxml and Transformers conflicts between specific pairs of skills. Test runs therefore build a throwaway environment per skill from a committed requirements file. The runner records 18 dependencies that either fall outside its uv/PyPI installation path or cannot be installed in the hosted environment. These include GitHub-only SDKs, conda-forge builds, CUDA packages and a package that needs a local MATLAB installation. Those dependencies can be installed in other environments; the ledger records a runner limitation rather than general unavailability. Another conflict involves module names rather than versions: 40 skills ship a scripts/_common.py, so collecting two skills into one

interpreter would resolve `_common` to whichever module was imported first and silently test the wrong file.[1] The test runner starts a separate process for each skill to prevent this.

The library inherits the dependency conflicts of the scientific software it documents. Per-skill environments keep those conflicts out of the project environment.

E How the library is validated

The term "validated" can refer to anything from "someone read it" to "its behaviour is tested". Figure E.1 shows which checks exist and which skills they cover. These scopes define what "validated" means here.

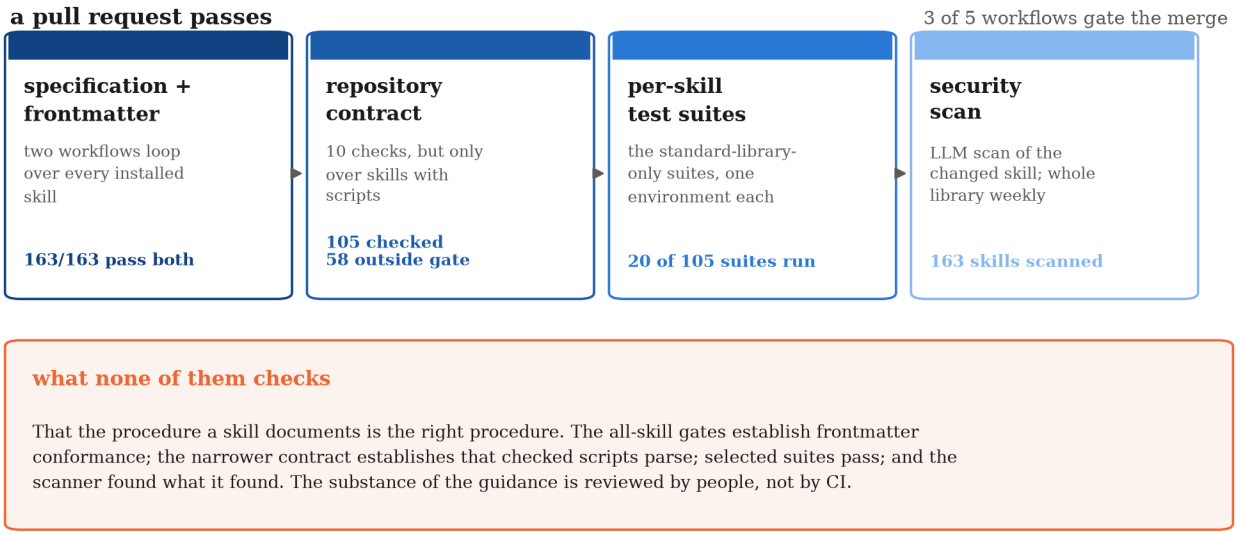


Figure E.1: The gates on a relevant pull request at tag v2.65.0. Frontmatter workflows loop over all 163 skills, whereas the 10-check structural contract loops over only the 105 skills with files under `scripts/`. Per-skill suites exist for all 105 script-bearing skills, but CI runs only the 20 suites whose tooling needs no scientific packages. The pull-request security scan covers changed skills; the library-wide report quoted in the text is the separate weekly run. None of these gates determines whether a documented scientific procedure is substantively correct.

Structural conformance. The reference specification validator and a repository-specific frontmatter check both loop over all 163 skills. Together they check names and descriptions, the six-field top-level set, `metadata.version`, `allowed-tools` string syntax and the repository's permitted host-block shape. That host-block shape still passes the reference validator, despite the mismatch with the specification text noted in Appendix B.

The separate `tests/_meta` suite imports a structural contract with 10 checks. They cover file length, tests and bytecode outside the skill payload, local-path resolution, Python parsing, banned dynamic execution, standard-library name collisions, contributor-local paths and shell-script validity, plus overlapping frontmatter checks. Its documentation says that it spans every skill, but at this tag

[1] Our contributing guide states a smaller figure for this; 40 is what the tree holds at this tag, and is an example of the drift Section 6 describes.

its loop is over the script-bearing set. It therefore gates 105 skills and leaves the other 58 outside those checks. An over-limit file would still reach the all-skill specification workflow, but only as a warning.

We ran the same structural functions over the full pinned tree for this description. They produce 2 findings: broken local paths in 2 documentation-only skills, all outside the CI loop. This shows that the narrower scope changes what CI catches. Table E.1 reports each check's applicable population, actual pull-request scope and state at the pin; it does not infer enforcement from a passing current tree.

Table E.1: Repository checks at tag v2.65.0. "Checked in PR" is the number of skills for which a failing result can block a relevant pull request; "Holds at pin" is our evaluation over the full population to which the rule applies. The broken paths occur in 2 documentation-only skills outside the structural contract's pull-request scope.

Rule	Applies	Checked in PR	Holds at pin
Fully enforced for its scope			
reference specification validator	163	163	163
repository frontmatter additions	163	163	163
bundled Python files parse	105	105	105
no banned dynamic-execution calls	105	105	105
no top-level standard-library shadowing	105	105	105
bundled shell scripts parse and are executable	2	2	2
script-bearing skill has a test suite	105	105	105
script-bearing skill has a requirements entry	105	105	105
Enforced only for script-bearing skills			
SKILL.md within the 500-line limit	163	105	163
no tests shipped inside a skill directory	163	105	163
no compiled bytecode shipped	163	105	163
documented local paths resolve	163	105	161
no contributor-local paths	163	105	163
Convention, not a pull-request gate			
generated workflow diagram present	163	0	162
listed in the documentation catalogue	163	0	162

Test coverage. The coverage guard asserts that every skill containing files under `scripts/` has a corresponding test directory and an entry in the test runner's requirements file. At this tag that correspondence holds exactly: 105 script-bearing skills and 105 suites across 129 test files, with 0 missing suites and no orphaned suites. We measure whether suites exist; their existence does not establish that every helper or code path has a behavioural assertion. The shared contract of 4 modules supplies reusable checks to per-skill suites. Its structural component parses all 527 bundled Python files with `ast`, checks the 2 shell scripts separately, and never imports skill code.

Suite existence, execution and coverage of a particular behaviour are different claims. Of the 105 suites, 20 run on pull requests that touch the workflow's relevant paths. The workflow selects suites whose bundled tooling needs no scientific packages, because the full sweep builds an environment per skill and several need CUDA, a JDK or a local MATLAB installation unavailable on the hosted runner. The remaining 85 are configured for the local isolated runner, and our contributor guide instructs maintainers to run them before a release; the pinned tree does not record whether that manual run occurred. 3 of the repository's 5 workflows have pull-request triggers: specification validation, the structural and selected per-skill tests, and a security scan of changed skills. The other 2 run on a merge to the default branch and on a weekly schedule.

The all-skill gates establish frontmatter conformance. The narrower structural gate establishes parsing and path properties for script-bearing skills. The selected suites establish only the behaviours they assert. None of these gates shows that following a skill's instructions produces correct science or evaluates the substance of the guidance.

Security scanning. Skills can direct consequential actions: a skill can instruct an agent to run arbitrary code, install packages, make network requests and modify files. We treat those capabilities as a direct risk. The instruction file is itself a prompt-injection vector, because it is text an agent is asked to follow [55], and susceptibility to injected instructions can now be measured directly for tool-using agents [56]. Surveys of public skill ecosystems report vulnerable skills at non-trivial rates, though the reported prevalence depends heavily on the detection method and on which repositories are swept [32, 35]. A lifecycle analysis of the standard argues that its most serious exposures are structural: there is no boundary between data and instructions, and the trust decision occurs once at install time [33]. Repository scanning cannot eliminate either exposure, and both limit what the checks in this section can establish. We scan every skill with a third-party scanner combining behavioural, trigger and LLM analyzers [34]. Pull requests that touch relevant paths scan changed skills. A weekly incremental scan covers the whole library, carrying previous findings forward for unchanged skills. We force a full rescan whenever the scanner version or the model changes, when a maintainer triggers one, and in any case at least every thirty days. We publish the results. The report committed at this tag (scanner 2.0.13, model claude-opus-5, generated 2026-08-24) covers 163 of the 163 skills; it flags 147 as safe and records 988 findings: 34 critical, 9 high, 241 medium, 703 low and 1 informational. 449 of those 988 are cross-skill findings, raised against a pair or a group of skills rather than against one (Figure E.2).

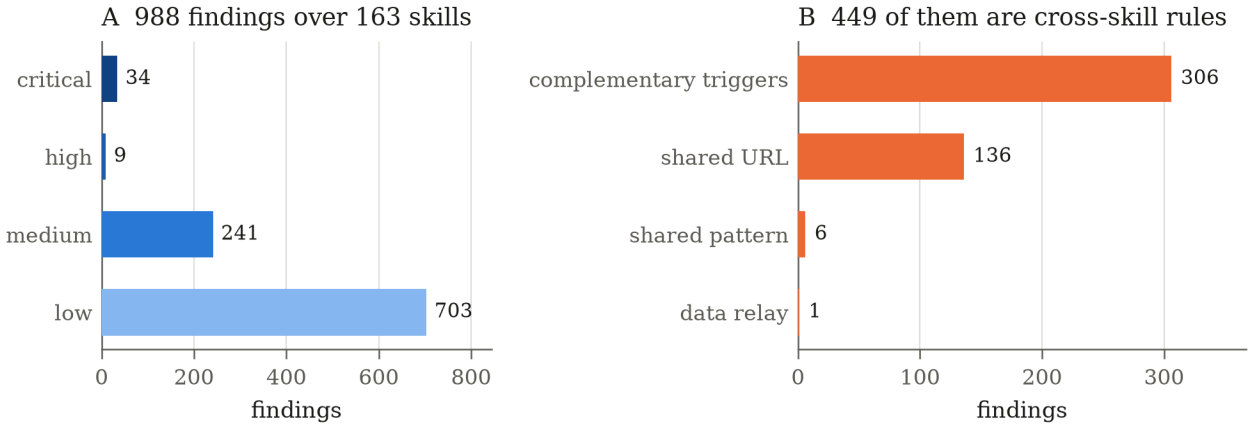


Figure E.2: Published scan output at tag v2.65.0, reported as scanner output rather than as a vulnerability census. **(A)** All 988 findings by severity. **(B)** The 449 cross-skill findings raised against groups of skills rather than individual skills, by rule. Panel B isolates the share of the report produced by library-level heuristics rather than findings attached to individual packages.

The scan counts are not a vulnerability census. 16 skills are not flagged safe, and 11 carry at least one critical finding. Of the 34 critical findings, 28 come from the 3 rules that fire when one package both reads an environment variable and makes a network call. Of the 11 skills with a critical finding, 8 name a credential-like environment variable in their own header. Our published triage document (`docs/security-triage.md`) identifies exactly this pattern, in which a skill reads its own API key and calls its own service, as a systematic false positive for those rules. Our `SECURITY.md` accordingly asks readers to verify a finding against the skill before acting on it.

The triage predates the report committed at this tag. It covers the scan of 2026-07-27, with 154 skills and 817 findings, rather than the current report's 163 skills and 988 findings. Its verdict that no critical finding survived verification therefore applies to the 33 critical findings, not to the 34 reported here; the remaining findings are untriaged at this pin. We count only those 3 rules as sources of this systematic false positive. A fourth rule also has "exfiltration" in its name, but the same triage document records one of its findings as genuine and already fixed, so we do not fold that rule into the false-positive count.

The cross-skill rules also fire on structural coincidence. The dominant rule flags skills with complementary trigger conditions, and one finding treats nearly every data-reading skill at once as a potential relay chain. We report the published, checkable counts but do not convert them into a vulnerability count, which the report does not support. The largest ecosystem survey finds that scanner verdicts used alone substantially overestimate maliciousness, while checking a flagged skill against its repository removes the great majority of suspicions [35]. Instead, the report documents our security practices. We publish scan output, describe a triage process and a route for contesting findings, advise users not to install every skill, and tell them to read a skill before trusting it.

F What the skills claim, and what they decline

Wrong answers delivered with confidence can cause harm in several domains covered here: clinical decision support, treatment documentation, diagnostic imaging, regulatory evidence and laboratory animal welfare. Our category descriptions in `README.md` place whole areas outside the scope of decision-making. The regulatory and standards category describes its output as "prepared for qualified review, never a certification, accreditation, or method-release decision". Imaging and pathology work is described as research-only. The healthcare-AI category covers retrospective validation and is "not patient-specific diagnosis, treatment, alarms, or deployment decisions". The clinical category carries no comparable disclaimer of its own. Instead, it names narrow deliverables: aggregate decision-support evaluation, draft report structures that must stay bound to their source facts, and formatting of treatment decisions a clinician has already written. Individual skills also state limits in their instructions. The clinical reporting skill, for instance, marks its output as a draft not for clinical use, requires a verified source-fact manifest before it produces any output, and instructs the agent to stop when source support or qualified review is missing. The standards-readiness skill states that it is not for compliance, certification or accreditation decisions and declines to reproduce clause text from the standards it prepares evidence against.

These limits are appropriate to the stated uses and should be reported without widening or understating them. In the instruction files loaded on activation, however, scope clauses are concentrated rather than uniform (Figure F.1). We first searched every instruction file with a broad lexical rule. We then manually adjudicated every reported positive, counting a skill only when we could point to an exact supporting clause in its instruction file. This procedure confirms an explicit scope limit in 10 of 163 instruction files. The limits are concentrated where the stakes are highest: 5 of the 9 skills in the clinical, regulatory and translational category, and 2 of the 6 in medical imaging, neuroscience and pathology. The confirmed set is a lower bound because the lexical rule may miss different wording. Manual adjudication prevents a keyword mentioned only in passing from counting as a positive. Our README frames categories as research-only, but only some skills in those categories restate the caveat. The agent loads the instruction file, not the README. A user or host installing a single skill by name need not encounter the category-level framing at all.

A paper that reports using a clinical or regulatory skill should state the limits in that specific skill because skills do not uniformly inherit the library-level limits.

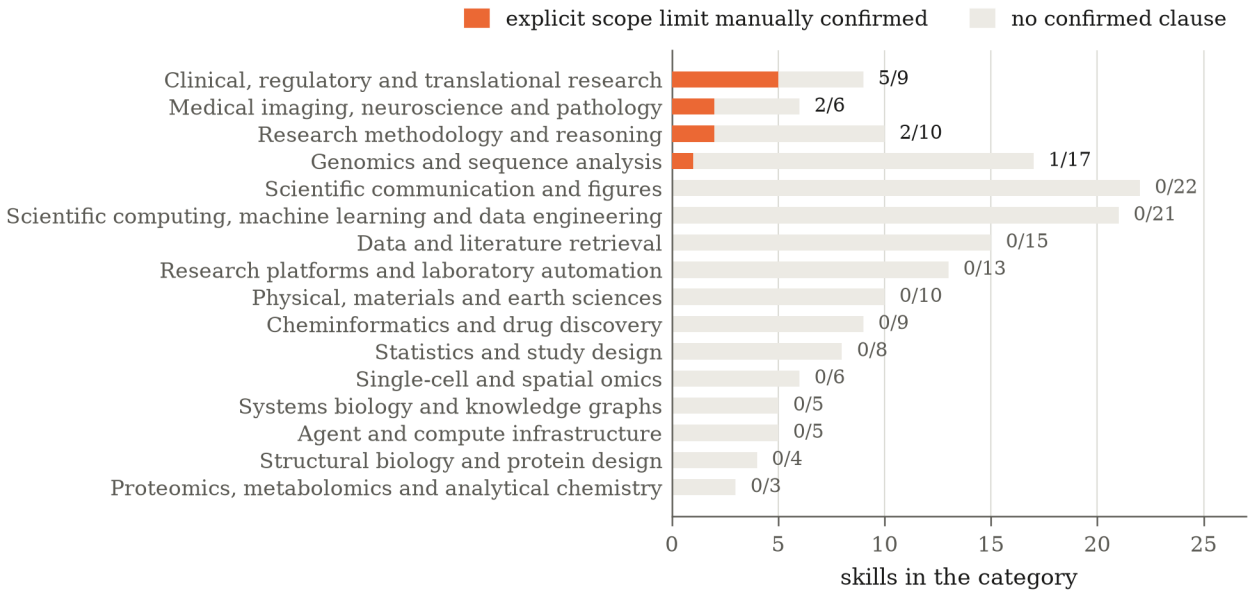


Figure F.1: Where a manually confirmed scope-limiting clause appears in the file an agent reads, by primary category at tag v2.65.0. Candidate files come from a lexical search, but each filled segment corresponds to an exact clause in that file. The confirmed set may miss different wording, and the figure shows concentration rather than a pass mark.

G Routing sensitivity

The routing measurement in Section 4 depends on how a description is turned into a bag of terms, and that choice is ours. The reported preprocessing lower-cases, keeps alphanumeric tokens of at least 3 characters, and removes a 27-word function-word list. We tested whether the finding depends on that list by recomputing the whole pairwise ranking under three alternatives: no stop list at all, a minimum token length of four, and a stop list extended with domain filler (`skill`, `data`, `analysis`, `workflow` and similar). Under each alternative, at least 9 of the original 10 closest pairs remain in that alternative's top-10 set. The named collisions in Figure 4B are therefore a property of the descriptions rather than of the filter applied to them.

We also compare the result with the security scanner's complementary-description rule. The rule identifies a largely different set of pairs. It flags 301 pairs with a mean cosine of 0.020, compared with 0.013 for unflagged pairs. Only 0 of the twenty closest pairs are among those it flags. Every input to this measurement is a `description` field in the tagged tree, and the preprocessing is fully specified above, so a reader who prefers a different scheme can substitute it and recompute the ranking, the 56 authored pointers and these comparisons without our code.

H Estimating the installed base

Every other number in this paper is computed from the pinned tree and can be recomputed by anyone who clones it. We keep the numbers in this appendix separate because they are not computed from the pinned tree. GitHub's repository traffic endpoint is readable only by accounts with push access. It returns a rolling 14-day window that GitHub does not archive. The result is therefore a dated snapshot that a reviewer cannot reproduce and that we cannot reconstruct after the window passes. We report it because the alternative is a company-reported user count with no stated

method. We state the estimate's assumptions so that readers can scrutinise them.

From 2026-08-14 to 2026-08-27, the repository recorded 161,210 clones from 15,682 distinct cloners and 61,901 page views from 21,268 distinct visitors. Clone volume is not a count of people. Active cloners generated an average of roughly 8 clones per active day. That repetition indicates continuous integration and mirroring rather than one-time installation. We therefore use the distinct-actor count to estimate the population from which the observed actors were sampled.

A single window observes only actors that cloned during that period. Summing the per-day unique counts gives $D = 20{,}234$. This exceeds the window-level unique count $P = 15{,}682$ because an actor active on several days is counted once in P but repeatedly in D. Their ratio, $k = D/P = 1.29$, is the mean number of active days among observed cloners. We use that conditional mean to estimate the sampling rate. We model each actor in a pool of size N as cloning independently on each day with probability p. An actor is then observed at least once with probability $q = 1-(1-p)^{14}$. Among observed actors, the zero-truncated mean number of active days is $14\,p/q$. Equating this expression with the measured k gives $p = 0.041$ and $q = 0.44$. Under the model, a fortnight observes fewer than half of the pool. The estimate is therefore $N = P/q \approx 35{,}690$.

Three limitations constrain what this estimate supports. First, the model assumes a homogeneous pool, whereas a build server may clone daily and a person installing the library may clone once. This heterogeneity inflates k, the mean active days among observed cloners, which inflates the implied observation rate q and so reduces the estimated pool N. The resulting bias lowers the estimate and is therefore conservative given the authors' interest in a larger value. Second, GitHub's unit is a distinct cloning actor, not a person. Shared egress can collapse a department into one actor, while one person's laptop and CI can count as two, and we cannot determine the net direction of those effects. Third, the estimate describes clients that fetch the repository, not agent sessions in which a skill was selected and used. No public signal supplies the latter quantity.

Two independent quantities bracket the result. The 15,682 distinct cloners observed directly provide a floor that requires no model. Applying the same estimator to page visitors gives a higher estimate of 51,089 because visiting is a weaker commitment than cloning. The repository's 37,905 stars are independent of both estimates and accumulate over its lifetime. The star count is within 6% of the clone-pool estimate, so it is a check rather than an input. This agreement does not confirm scientific use because both signals measure interest in a repository, not use of a skill.