OPINION PAPER

Reproducibility in the Era of AI Agents

Jimmy Lin
University of Waterloo
Canada
jimmylin@uwaterloo.ca

Abstract

Reproducibility efforts should yield easy-to-use, well-packaged, and self-contained software artifacts. However, the goal is to enable AI agents, *not* fellow human users, to reproduce empirical results. Gone are the days where a user engages with code and documentation directly. Increasingly, this task is delegated to agents, and even in cases where user intervention is required, interactions are mediated by agents. Given this shift in the target audience, developers building software artifacts in support of reproducibility need to optimize for the "agent experience" alongside the "developer experience". In modern parlance, this means making software *legible* to agents. This paper explores the implications of this shift in mindset and attempts to explain what's changed and how we should adapt. Generalizing on my own recent attempts to "agentify" the Anserini IR research toolkit, I share lessons learned that hopefully illustrate this shift in emphasis with current agentic technology.

1 Introduction

This paper attempts to argue a simple point:

> The goal of reproducibility efforts is to enable *AI agents* to reproduce empirical results.

The key difference from commentary about reproducibility written before is a shift of emphasis from human users to AI agents.[1]

We are witnessing a profound shift in how software is designed and built. For decades, software engineering has optimized for the developer experience: readable code, intuitive APIs, and clear documentation for human programmers. But as agents increasingly participate in the software lifecycle, we must also design for the agent experience. This means creating systems that are discoverable, navigable, and usable by agents—not as an afterthought, but as an important design goal at the outset. The term that has emerged recently to encompass all these aspects of agent–software interaction is *legibility*. We should increasingly be optimizing for the "agent experience" alongside traditional concerns for the "developer experience" that have long dominated our thinking. This important trend not only affects the entire software industry, but also impacts how researchers think about reproducibility.

[1] Hereafter, just *agents*; I also refer to users and humans interchangeably.

Today, reproducibility is typically operationalized as a software repository associated with empirical results that conveys an implicit promise: *I'm giving you everything needed to recreate my results, and you should be able to do so from my repo.* Most often, the repository is released under an open-source license and hosted on GitHub, separate and distinct from the source of the results, be it a paper, blog post, tweet, etc.

However, this state of affairs is rapidly changing. Yesterday, reproducibility was mainly a manual exercise: a human navigates to a repository and attempts to reproduce the purported empirical results based on the documentation and code available there. This process is time-consuming, error-prone, and highly inefficient—but (thankfully) is increasingly displaced by automated processes driven by agents. Today, a human simply steers the agent to a repository and prompts the agent to reproduce the purported empirical results. Ideally, this process is "one shot"—the agent is successful without user intervention—but even if human input becomes necessary, subsequent efforts are likely agent-mediated in a collaborative dialogue. Gone are the days when a human attempts reproduction "by hand" (without agent assistance), by reading documentation, debugging scripts, etc. Developers of reproducibility artifacts need to adjust to this reality.

I have been passionate about reproducibility throughout my academic career. I delight in not only sharing "cool things" that my research group has accomplished, but teaching others and helping them recreate what we've done. I've written before in Lin [2022]:

> Reproducibility efforts should yield easy-to-use, well-packaged, and self-contained software artifacts that allow others to reproduce and generalize research findings.

While I believe this statement remains true, the reference to "others" takes on a very different meaning today from what I originally intended a few years ago. Back then, I imagined the audience of reproducibility efforts to be students, other researchers, industry practitioners, etc.—in other words, *other humans*. This is likely not the case anymore, and thus a more accurate rephrasing of the above assertion might be the following:

> Reproducibility efforts should yield easy-to-use, well-packaged, and self-contained software artifacts that allow *agents to reproduce empirical results.*

The key difference is an emphasis on agents as the audience. For now, agents should be responsible for reproducibility, but the initiative for figuring out what to do next (e.g., generalization) should continue to reside with users (see discussions in Section 6). The other wording change is mostly incidental: "empirical results" more accurately capture the goal of reproducibility than "research findings", in that some research findings may not be empirical in nature. Also, we often wish to reproduce results that are not primarily research in nature.

The product of reproducibility efforts remains software artifacts, but their designs are very different today than in the pre-agentic era. Quite explicitly, reproducibility artifacts should be *legible* to agents, and developers should be considerate of the "agent experience" alongside the traditional "developer experience". What exactly does that mean? My goal is to articulate this viewpoint and to explore its implications. Concretely, I share lessons from my own attempts to "agentify" the Anserini IR research toolkit developed over the past several years by my research group with the help of external collaborators, and I hope that my recommendations will serve as a helpful starting point for other developers on this journey.

Before proceeding, three caveats: First, since reproducibility involves building software artifacts, most best practices from modern agentic engineering apply, although I will explicitly point out divergences. Second, my focus is on reproducibility in the context of information retrieval, but many of the lessons I share are broadly applicable to adjacent areas such as natural language processing, data mining, applied machine learning, etc. Finally, these reflections capture a snapshot in time: best practices in the agentic era are only beginning to emerge, and elements of what I discuss here might not stand the test of time.

2 Preliminaries

What do I mean by reproducibility? I use this term as defined by the ACM in its Artifact Review and Badging Policy,[2] summarized in context with closely related terms, as follows:

- **Reproducibility** can be characterized as "different team, same experimental setup". Specifically, "this means that an independent group can obtain the same result using the author's own artifacts." That is, the product of a reproducibility effort is an artifact that is shared, and using the artifact, another group can recreate the same empirical results.
- **Replicability** can be characterized as "different team, different experimental setup", and operationally, "this means that an independent group can obtain the same result using artifacts which they develop completely independently". Importantly, replicability does not involve sharing artifacts.
- **Repeatability** can be characterized as "same team, same experimental setup", i.e., "a researcher can reliably repeat her own computation". This represents a much lower bar, although an interesting variation is within-group repeatability—for example, can one member of my research group obtain the results reported by another member?

Reproducibility is the focus of this paper, but I use the terms above in the way prescribed by the ACM policy.[3] I argue that reproducibility represents the gold standard we should strive for—whether by users or agents.

Repeated from the introduction, the implied (mostly social) contract around reproducibility today is that a human would be able to reproduce the empirical results with the associated repository. The degree of success can be characterized in terms of the following questions:

- **Is it possible?** Can I actually reproduce the results? Just because the repository exists doesn't mean it contains relevant content [Leech et al., 2024]; the code might be broken or incomplete. Just because relevant content exists doesn't mean that the code runs, and just because the code runs doesn't mean that the results are reproducible [Voorhees et al., 2016]; the code might generate results that are very different from those reported.
- **Is it easy?** Just because I can reproduce the results doesn't mean that it is easy. Often, a thousand paper cuts lie on the path to successful reproductions. These refer to mostly minor annoyances that are fixable, but require effort—for example, references that are hard-coded to a specific environment, inconsistent model names, code on a path commented out

[2] ACM Artifact Review and Badging Policy

[3] Confusingly, a previous version of the ACM policy swapped the definition of reproducibility and replicability.

by default, incompatible software dependencies, etc. To "make it easy", I have previously articulated the aspirational ideal of what I call "two-click reproductions" (2CRs) [Lin, 2022], where a user can reproduce a reported result (for example, from a paper) with only two clicks: one click to copy a command-line invocation from a source (for example, a documentation page) and another click to paste the command into a shell.

- **Is it interesting?** Just because an empirical result is reproducible doesn't necessarily make it interesting. For example, the authors might only release inference outputs for popular benchmark test collections without sharing the underlying model, in which case the technique cannot be evaluated on arbitrary input. The authors might only share the weights of the model and not the training recipe, in which case generalizing the approach to another domain might be difficult. I have previously described this as the "scope" of the reproduction [Lin, 2022]: it is important to articulate what a reproducibility artifact *does*, and just as important, what it *doesn't*. This scoping influences whether a reproduction is interesting and what can be done to further build on the results.

These questions become important when I discuss differences between users and agents. Beyond reproducibility, we may wish to generalize the results, transfer them to different domains, reuse the techniques in different contexts, etc. I'll return to discuss "what happens next" in Section 6.

What do I mean by agents? Stripping away unhelpful analogies and deliberate attempts at making things sound more complicated than they actually are, agents are simply models using tools in loops. More precisely, from Anthropic engineers: "LLMs using tools based on environmental feedback in a loop".[4] Thus, it is productive to decompose agents into their basic components:

- The model — for example, an LLM such as OpenAI's GPT or Anthropic's Claude.
- Tools — mechanisms that allow the model to interact with the environment, for example, `ls`, `grep`, query a database, read and write data to files, etc.
- The agentic harness (or just the harness) — the part that "does the looping". In more technical terms, the harness performs orchestration: it sends information to the model, executes tool calls, relays observations back to the model, and helps decide the next action. The harness is also in charge of providing instructions and context to the model, handling errors, respecting budget limits and other business logic, and deciding when to stop.

Agents represent a major advance over static and predefined workflows, even those that already take advantage of LLMs. Their design is well suited to handle open-ended situations with complex and possibly conflicting goals, often based on ambiguous or incomplete input. Agents do not merely execute actions in a fixed sequence, but are afforded the flexibility to traverse different execution paths to accomplish tasks in an autonomous manner.

3 From Users to Agents

In the era of agentic AI, reproducibility is for agents, not users. The question is *not*, "can a user reproduce these results", but "can an agent do it"? But what exactly does it mean to design

[4] Anthropic's engineering note on building effective agents

for the "agent experience" and make artifacts more *legible* to agents, beyond catchy slogans? To better operationalize this shift in mindset, let's start by discussing some key differences:

Agents are challenged in different ways. Needless to say, agents don't get tired, annoyed, or bored. Many details that derail (or at least slow) human-driven reproducibility efforts are "not a big deal" for agents—examples include complex configurations, custom environments, long dependency chains, hard-coded filesystem references. For the most part, an agent can solve these issues today, e.g., rewrite paths based on the exact location of a checkout. Previously, a user might have skimmed an installation walkthrough and concluded that the reproduction was "not worth the effort". Today, less so: it's easy to let the agent crank on it tirelessly until all possible avenues have been exhausted.

However, agents are challenged in other ways: they run into token budgets, permission issues, hallucinated fixes, context loss, difficulty knowing when to stop, and countless other issues. Thus, "tireless" isn't necessarily "reliable". For developers, designing for agents means handling different challenges. On the one hand, the "overall shape" of the software artifact is more important than the exact details (which an agent can fix), thus relieving the developer of certain burdens. On the other hand, an agent might become confused when overwhelmed with too much information that a user can simply ignore, thus creating new challenges.

Agents care less about the implementation language or framework. Typically, a user is only comfortable with a handful of programming languages and frameworks, which restricts the space of possible reproduction efforts, i.e., a user is less likely to engage with a software artifact written in something unfamiliar. Agents have a larger repertoire and are less constrained in this manner (since they are trained with large amounts of diverse data).

For developers, this suggests that picking the right tool for the job is more important than building in a particular software ecosystem just because it is popular and widely adopted. This is potentially liberating for developers, as the agent can usually write adapters and glue code automatically to support interoperability.

Agents embrace idioms, conventions, and standards. The models that underlie agents are typically post-trained with tool calls that adopt common idioms, conventions, and standards used across the industry. Thus, the agent has a large repertoire of things it already knows how to do, right out of the box.

For developers, this means that artifacts should respect existing idioms, conventions, and standards. This idea of always adopting best practices is developed into specific recommendations on building tools for agents, covered in Section 4.1.

Agents currently have limited context. There are two main limitations of agents today with respect to context: First, they cannot reason about anything that is not provided to the model directly, which means that they lack the context needed to accomplish specific tasks out of the box. Second, an agent's ability to understand user intents and accurately follow instructions degrades as inputs to the model accumulate and the model's context window fills up.

Thus, developers need to walk a fine line: They need to supply procedural knowledge and instructions about workflows to actually make agents useful, which is operationalized in agent skills

(see Section 4.2). However, developers also need to carefully manage context to not overwhelm the agent with too much information (see Section 4.3).

Agents are rapidly increasing in capabilities. Any claims about the limitations of agents capture only a snapshot in time. Models and harnesses are rapidly improving, and any deficiencies will be remedied (sooner than you think!), including the limitations described above. Nevertheless, it remains worthwhile to design artifacts based on the agent's affordances to increase reliability, token efficiency, etc., although best practices will certainly evolve over time.[5]

4 Designing for Agents

Building on the narrative that reproducibility should target agents, not users, this section provides concrete recommendations for developers of reproducibility artifacts. How exactly should we design for agent legibility to create a good "agent experience"?

I share my own experiences "agentifying" software artifacts that my research group and our collaborators have developed, as part of attempts to generalize design principles that can be broadly applied. These lessons represent a snapshot in time, consistent with the current capabilities and limitations of agents today. It is not clear if these recommendations rise to the level of "best practices", or if they will even withstand the test of time (measured in months, likely), as new models might render my advice obsolete and even counter-productive. Nevertheless, I hope that my musings are helpful at present.

Before proceeding, some necessary context: For this exercise, I focus on what information retrieval researchers call the "core" retrieval problem (or, just *ad hoc* retrieval). I'll wholesale lift the definition from Lin [2021]:

> Given an information need expressed as a query q, the task is to return a ranked list of k documents[6] $\{d_1, d_2 \ldots d_k\}$ from an arbitrarily large but finite collection of documents $\mathcal{D} = \{d_i\}$ that maximizes a metric of interest, for example, nDCG, AP, etc. These metrics vary, but they all aim to quantify the "goodness" of the results with respect to the information need.

The retrieval task is also called top-k retrieval (or ranking), where k is the length of the ranked list.[7] Generically, a retrieval model is a software artifact that addresses the core retrieval problem, and one focus of many researchers is to build better models.[8] This process is primarily empirical,

[5] But as a counterpoint, see discussions at the end of Section 5: maybe it doesn't matter.

[6] Consistent with parlance in information retrieval, I use "document" in a generic sense to refer to the unit of retrieved text, even though in truth it may be a passage, a web page, a PDF, or some arbitrary span of text.

[7] As an aside, there is emerging debate on the importance of ranked retrieval in the era of agentic AI, as search is increasingly performed by agents, not users. The problem as defined here focuses primarily on one-shot retrieval, which is increasingly inconsistent with agentic search where iteration and feedback are first-class citizens [Chen et al., 2025; Li et al., 2026b; Hsu et al., 2026]. However, these developments are mostly incidental to the focus of reproducibility in this paper.

[8] In truth, modern retrieval is broken into multi-stage architectures, most recently (re-)articulated by Nogueira et al. [2019] in the transformer era; cf. Pradeep et al. [2021]. However, the general design is roughly 20 years old [Matveeva et al., 2006; Wang et al., 2011]. In industry, Microsoft Bing calls these stages "L0", "L1", "L2", etc., and for a recent "lost in translation" moment, see a related post on X.

as the most widely accepted way to demonstrate model effectiveness involves sharing evaluation results on benchmark test collections. It would be desirable for the community to have widely accessible artifacts that reproduce such comparisons.

In this context, my research group at the University of Waterloo—with the help of external collaborators—has built two popular IR toolkits, Anserini [Yang et al., 2017, 2018; Lin et al., 2025] and Pyserini [Lin et al., 2021], over many years:

- Anserini is built around the open-source Lucene search library, the most widely adopted solution for tackling search problems in deployed real-world applications (typically, via platforms such as Elasticsearch). Our goal in building a *research* toolkit around Lucene is to facilitate a two-way exchange between academia and industry [Devins et al., 2022].
- Pyserini provides Python bindings to the capabilities offered in Anserini and integration with neural retrieval models built on industry-standard packages such as Hugging Face Transformers, PyTorch, and Faiss. We built this toolkit to support developers today who are more comfortable with Python than Java.

Anserini and Pyserini aim to reproduce competitive first-stage retrievers on standard benchmarks (e.g., MS MARCO, TREC DL, BEIR, etc.). We have achieved some success promoting reproducibility in the community by supporting students, researchers, and practitioners. However, our toolkits were designed primarily with the "developer experience" in mind—how to best enable users (i.e., other humans) to reproduce empirical results. As a test case, I have been "agentifying" Anserini over the past several months—reimagining what the toolkit would look like if it were optimized for the agent experience. I am happy to share what I have learned.

At the highest level, designing for agents means designing from the perspective of agents. As Barry Zhang, an engineer at Anthropic, says, "Put yourself in the model's shoes". I have crystallized this into three directives:

- Build useful tools;
- Formulate skills around tools;
- Manage context carefully.

I will elaborate on these three ideas below in more detail, but they capture a modular bottom-up approach that contrasts with a monolithic top-down design captured in many reproducibility efforts. Many reproducibility artifacts are designed with a single `run_all.sh` script that attempts to "do everything". While having a single point of entry clarifies purpose, the approach in my opinion is often brittle and error-prone, and in many ways represents the opposite of designing for agents. Instead, I advocate a bottom-up approach around tools and teaching models how to use those tools, detailed below.

4.1 Build Useful Tools

Given that agents are basically models with tools in loops, it makes sense to give agents the best possible tools. In the context of reproducibility efforts, the tools correspond to the primitive building blocks that the models assemble to perform various tasks. They answer the question: What are the things you can do?

Tools provide entry points into the capabilities offered by the artifact. I argue that they should be designed according to the Unix Philosophy, which dates back to the 1970s: "Make each program [tool] do one thing well." Consider three good ideas:

1. Use idioms, conventions, standards, and existing best practices whenever possible.
2. Enable tools to be self-documenting and reflective.
3. Encapsulate as much detail as possible.

Whenever possible, *don't* reinvent the wheel, per idea (1). For example, design tools around command-line interfaces (CLIs) or MCP servers;[9] if the tool is a REST endpoint, make it compliant to the OpenAPI Specification. And within each type of tool, use standard idioms to the extent possible (more below). To illustrate concretely, consider the following snippets:

```
$ java -cp anserini-fatjar.jar io.anserini.cli.PrebuiltIndexRegistry --help

Options for PrebuiltIndexRegistry:

  --list                                List available prebuilt indexes.
  --filter [regexp]                     Filter prebuilt indexes by regular expression.
  --type [flat|inverted|impact|hnsw]    Filter prebuilt indexes by type.
  --help                                Print this help message and exit.

$ java -cp anserini-fatjar.jar io.anserini.cli.TopicsRegistry --help

Options for TopicsRegistry:

  --list                    List available topics.
  --filter [regexp]         Filter topics by regular expression.
  --get [topics]            Enumerate topics.
  --help                    Print this help message and exit.
```

The `PrebuiltIndexRegistry` tool lists prebuilt indexes available out of the box in Anserini. Our toolkit provides many such indexes (currently, hundreds) for commonly used document collections (MS MARCO, BEIR, etc.) so that agents do not need to build indexes from scratch and can search them right away. Indexes span several standard types: inverted indexes (for BM25), impact indexes (for learned sparse models such as SPLADE [Formal et al., 2021; Lassance et al., 2024]), and flat or HNSW vector indexes (for learned dense models such as BGE [Xiao et al., 2024]). This tool answers the question: What can I search out of the box? Similarly, the `TopicsRegistry` tool lists topics (queries) that are commonly used in retrieval experiments (e.g., the dev set of MS MARCO, DL19/20, etc.). This tool answers the question: What can I search for out of the box?

Both tools attempt to put into practice ideas (1) and (2). The concept and nomenclature of a registry is widely known and commonly used across the software industry. All Anserini CLIs expose a `--help` option that describes what the command does, which is a widely adopted convention. The help message describes available options in the form of strong action-oriented verbs paired with succinct yet unambiguous descriptions. The two CLIs generate output in JSON,

[9] There has been much debate about the relative merits of CLIs versus MCP servers; ask your agent to summarize the arguments from each side if you're interested. Here, I use CLIs, but Pyserini provides MCP bindings as well [Ge et al., 2026].

which allows agents to examine and further manipulate results using standard tools such as `jq`. By following these ideas, we are taking advantage of standard post-training recipes that models have been subjected to—they already know how to use tools, so it makes sense to structure Anserini CLIs to be as familiar as possible.

Per idea (3), these tools hide many details. When needed, Anserini downloads a copy of the requested prebuilt index from a known location (servers at the University of Waterloo and elsewhere) and caches it locally. Similarly, Anserini knows where to find and download topic files, so that the user isn't bothered with having to manage such details.

Next, Anserini provides another building block, a CLI called `SearchCollection`, that enables both users and agents to perform an end-to-end retrieval run. The command to perform BM25 retrieval on the MS MARCO passage corpus using the dev queries is as follows:[10]

```
$ java -cp anserini-fatjar.jar io.anserini.search.SearchCollection \
  -index msmarco-v1-passage \
  -topics msmarco-v1-passage.dev \
  -output run.msmarco-v1-passage.dev.bm25.txt \
  -bm25
```

The command generates a standard TREC runfile that can be directly evaluated with `trec_eval`, also packaged in Anserini:

```
java -cp anserini-fatjar.jar trec_eval -c -M 10 -m recip_rank msmarco-v1-passage.dev \
  run.msmarco-v1-passage.dev.bm25.txt
```

There are similar commands for learned sparse representations such as SPLADE and learned dense representations such as BGE. The `SearchCollection` tool builds on the above two tools in that `PrebuiltIndexRegistry` supplies the options for `-index` and `TopicsRegistry` supplies the options for `-topics`. This characterizes how the building blocks compose.

Per idea (3), the high-level goal is to reduce friction for users who wish to reproduce a particular empirical result—in this case, a widely reported baseline on a popular benchmark dataset. The complexity of modern IR evaluation methodology means that there are a gazillion tiny details to keep track of: Where do I get the index? Where do I get the topics? For anything fancier than BM25, where do I get the model? What versions of each, exactly? Is *this* the right version that goes with *that*? What are the scores I should be expecting?

Sorting through these details is not intellectually challenging, but can be confusing for a novice (e.g., a student just getting into IR) or even a seasoned IR researcher who's never worked with a specific collection before. As a simple example, there are often different versions of a particular set of topics (queries): what everyone calls the 6,980 queries in the MS MARCO passage ranking development set is actually only a subset of the "real" full development set [Bajaj et al., 2018]. My other favorite example is TREC-COVID, which has no fewer than a dozen different sets of relevance judgments [Roberts et al., 2020]. All of them are useful, but for answering different research questions. Which one do you use?

In Lin [2022], I advocated for the aspirational ideal of "two-click reproduction" (2CRs), where the goal is to relieve the user of all these burdens via simple, self-contained commands that can

[10]The astute reader might wonder why we have mixed single dash (`-`) vs. double dash (`--`) conventions: it's due to legacy code that has not yet been refactored.

be copied and pasted into a shell to reproduce an empirical result (i.e., one click to copy, another click to paste). Perhaps this bar is too high, since agents can independently fix broken code, reconcile inconsistencies, etc.[11] Perhaps it suffices to provide agents with good tools, and then let them figure things out on their own.[12] This is where agent skills come in.

4.2 Formulate Skills Around Tools

Tools are necessary but not sufficient. A collection of well-designed tools answers the question: What are the things you can do? However, reproducibility usually requires more than invoking a single tool. The agent must decide which tools to use, in which order, with what parameters, and how to interpret the outputs. Such context is not usually baked into the model, but must be supplied externally. This is where skills come in.

Skills, or more precisely, agent skills, are a lightweight, open format for extending agent capabilities with specialized knowledge and workflows.[13] At its core, a skill is a folder containing a `SKILL.md` file in markdown format that contains instructions telling an agent how to perform one or more specific tasks. Beyond these instructions, a skill can also bundle scripts (i.e., code in a programming language like Python), detailed reference materials (typically, additional markdown files), and other resources. Skills were originally developed by Anthropic but later released as an open standard to capture organization- or context-specific procedural knowledge in a portable, reusable, and model-agnostic manner. They have recently emerged as an active area of academic study [Zheng et al., 2025; Huang et al., 2025; Xia et al., 2026; Li et al., 2026a; Alzubi et al., 2026; Si et al., 2026; Ouyang et al., 2026; Zhou et al., 2026].

A skill is not documentation in the traditional sense and shouldn't be thought of that way. Although the main parts of a skill are written in natural language, skills are not primarily meant to be consumed by users, and in most cases, are not even written by humans (see Section 5). Rather, a skill packages procedural knowledge for an agent: given a task, here is the sequence of actions that usually works, here are the outputs to expect, here are the sanity checks to perform, and here are the common failure modes to watch for.

It might be tempting to think of skills as "merely" prompts, but a more accurate characterization is that skills represent executable artifacts. They don't merely document or describe in a static manner; they directly accomplish the tasks they were designed for (as steered by a user). Skills *are* programs, in the sense that a program is just a sequence of instructions, that once fed into a computer, executes to perform some task. The markdown in a `SKILL.md` file *is* code, in that code is merely the textual representation of a program—in this case, not in a programming language, but in natural language. In the same way that a Java Virtual Machine takes binary class files generated from Java source code to execute a program, an agent takes a skill written in natural language and executes its specifications using a model. Ensuring that skills are portable across different agents (models, harnesses, etc.) is not conceptually different from verifying the executable behavior of a program across different operating systems (e.g., Windows vs. MacOS), different architectures (e.g., x86 vs. ARM), etc.

[11] As an example, `anserini-fatjar.jar` is not literally correct, since jar files usually contain versions; an agent has no trouble figuring this out and rewriting the filename to reference the correctly versioned jar.

[12] Perhaps, instead of 2CR, the gold standard should be 1SPR (one-shot prompt reproducibility).

[13] Agent Skills home page

In the context of Anserini, skills contain compact, task-oriented instructions that teach an agent how to use the available tools (the CLIs discussed in the previous section) to accomplish certain reproducibility tasks. Conceptually, the tasks are grouped into two high-level categories:

- *Reproductions from prebuilt indexes.* Anserini provides indexes out of the box that eliminate the need for an agent to obtain the underlying document collection and then to run the indexing pipeline. As described in the previous section, Anserini knows where to download prebuilt indexes for common benchmarks used in information retrieval research, and provides a search tool for reproducing the empirical results directly.
- *Reproductions from raw document collections.* For users who wish to start from scratch with the document collections, or for cases where convenient prebuilt indexes are not available, Anserini provides a driver program that reproduces empirical results end to end. This encompasses a multi-step pipeline that includes downloading the document collection (if publicly available), building the index, executing one or more search runs, evaluating the output runfiles, and finally verifying against expected results.

Currently, Anserini implements two main skills: `anserini-cli` teaches the agent how to use the CLIs discussed in the previous section, and `anserini-reproduction` teaches the agent how to perform the two types of reproductions above.

While skills should be conceptualized as executable artifacts and markdown should be treated like code, the "interpreter" (i.e., model + harness) is a non-deterministic execution engine that behaves very differently from the von Neumann machines that we are accustomed to. This is neither good nor bad, but simply a fact, and we need to develop different intuitions about these machines. Through mostly trial and error, I have accumulated some experience in writing effective skills, which I would distill into the following recommendations:

1. They are task-oriented, not tool-oriented.
2. They provide executable recipes, not prose descriptions.
3. They provide operational guidance, including validation and recovery steps.
4. They are opinionated, but not overly prescriptive.
5. They are modular and composable.

A common temptation is to write a skill that simply describes what each tool does. This is not very helpful, because well-designed tools should already be self-documenting (see discussions in the previous section). The agent can invoke `--help`, inspect examples, and infer many details from the command-line interface itself. Neither should the skill be a complete reference manual. Instead, a useful skill should be organized around tasks that the agent is likely to perform, e.g., how to reproduce a BM25 baseline with a particular index and a set of topics.

Skills should encode procedural and workflow knowledge. Running through the example above: the agent must first identify the test collection and topics, typically starting from an imprecise natural language label from the user. The user might specify MS MARCO, which the agent translates into `msmarco-v1-passage.dev`, inferring that the user likely meant the passage corpus (as opposed to the document corpus) and the dev subset comprising 6,980 queries. The agent needs to confirm that a compatible (prebuilt) index exists (another mapping exercise from an imprecise natural language descriptor into an unambiguous internal symbol), then run the appropriate search command, plugging in all the parameters discovered above. Finally, the generated run must be

evaluated using a tool such as `trec_eval`, and the resulting metrics must be compared against reference scores; only then can the reproduction attempt be considered successful.

The sequence above is conceptually obvious to experienced IR researchers, but they do not necessarily know how each step maps to Anserini commands. Students learning IR may have a basic understanding of these steps, but their knowledge might be shaky. However, an agent encountering the Anserini toolkit for the first time has no idea what to do—but yet is tasked with executing successful reproductions. Skills bridge this gap, providing the context that teaches the agent how to use the tools, so that it can assist users from diverse backgrounds. Using skills, the agent learns how to translate a high-level task into the right sequence of tool calls.

Thus, a major focus of skills is providing operational guidance in an opinionated manner. Validation instructions allow an agent to confirm that something worked as expected (e.g., in the case of reproducibility, metrics computed from a runfile match reference values). Recovery instructions tell an agent what to do if something goes wrong and how to fix commonly encountered errors, including information on frequent "gotchas". Operational guidance is often provided in the form of sample tool invocations: usage examples allow the agent to map from tasks to the combination of parameters that elicits the desired outcome.

In my experience, skills should be opinionated but not overly prescriptive. They should provide firm guidance on how to accomplish a task, the opposite of the TIMTOWTDI ("There Is More Than One Way To Do It") philosophy espoused by some programming languages. To not confuse an agent by overloading its context window (see the next section), skills should promote a single "idiomatic" or "obvious" way to accomplish something. However, by not being overly prescriptive, well-written skills afford agents the flexibility to handle unanticipated situations and edge cases. They provide the starting point for agents to implement clever workarounds as needed.

Finally, skills should be modular and composable, much like complex programs in any programming language. Most software engineering best practices remain applicable when building skills: there should be separation of concerns and well-defined abstractions that allow skills to compose in a priori unanticipated ways. As a concrete example, the `anserini-cli` skill is encapsulated and separated from the `anserini-reproduction` skill in that the former might still be useful in contexts beyond reproducing empirical results.

In summary, tools and skills play complementary roles. Tools expose capabilities; skills teach agents how to compose those capabilities into meaningful workflows to accomplish specific tasks. Together, they provide the foundation for agent-friendly reproducibility. However, we need to be careful: skills encode knowledge that resides outside the agent's harness and model, in the same way that a program and the computer it runs on are distinct. This separation introduces a new challenge. We need to provide sufficient context without overwhelming the agent with too much information. This brings us to the third directive: manage context carefully.

4.3 Manage Context Carefully

At the core of an agent is its "agentic loop", whereby the harness repeatedly sends information to the model, orchestrates tool calls, and observes model outputs. The model only has access to what's provided in its input—typically referred to as the context—nothing else. In particular, the model does not have direct access to the entire codebase, the complete documentation, previous execution histories, etc. It only reasons over the context that has been provided to it by the

harness. This context is finite, and as it grows, the model's ability to follow instructions, attend to details, process tool output, and consider previous interactions can degrade. Although modern harnesses provide mechanisms such as compaction and memory, they remain imperfect and have yet to solve the underlying challenges. At present, we should not expect agents to manage context reliably on their own. Rather, we need to help them by structuring context so that the right details are available when needed, without overwhelming the model with everything at once.

The general design principle is known as progressive disclosure, which is frequently discussed by Anthropic engineers and prominently referenced in the agent skills documentation. Operationally, progressive disclosure means organizing information so that the agent first sees only the minimum necessary orientation, with clear pointers to additional details that can be loaded on demand. This is the opposite of writing a single massive README that attempts to explain everything at once. While this approach may be tolerable for humans, who can skim, search, and ignore sections as necessary, agents often lack the ability to figure out what's important and what's not.[14] While progressive disclosure is a good idea for humans, it is absolutely essential for agents.

In practice, this means that a skill should begin with a compact description of its purpose, the tasks it supports, and the immediate decision points the agent needs to resolve. Details should be pushed into separate files that the agent can load only when needed. As a concrete example, the `anserini-reproduction` skill clearly explains the distinction between reproductions from prebuilt indexes and reproductions from raw document collections at a high level, then points to reference documentation for each specific configuration. The agent should not need to load the details of every test collection just to reproduce a single BM25 baseline on MS MARCO.

Another way to think about this approach is that skills should be structured for bounded reasoning, guided not only by the question of "What does the agent need to know?" but also "When does the agent need to know it?" and "How will the agent know where to find details?" For proper context management, the absence of information (i.e., hiding irrelevant details behind a clear pointer) is just as important as the presence of information.

Tying everything together: Tools provide the primitive building blocks. Skills teach agents how to compose those building blocks into workflows that accomplish specific tasks. Context management ensures that the relevant knowledge is available to the agent when it is needed. Together, they define in my experience the three pillars of a good "agent experience".

5 Reproducibility For Agents, With Agents

The discussion so far has focused on agents as the consumers of reproducibility artifacts. But if agents are the intended audience, then agents should help us build the artifacts that they themselves will consume. In other words, we can use agents to help make software artifacts more legible to agents. However, based on my experiences and the present state of technological

[14] Progressive disclosure is not a new idea. In human–computer interaction, it is commonly associated with work by Carroll and Rosson dating back to the 1980s, later popularized by Jakob Nielsen; however, both efforts focused primarily on interface design. Similar intuitions are conveyed in advice commonly given to writers and teachers in the so-called "spiral approach": introduce a concept first in its simplest form, then return to it later with additional details and nuance, iterating until all relevant knowledge has been presented. Applicable both in writing and in teaching, the common principle is that understanding should be staged, and details should not be presented all at once (to a reader, to a student, to an agent, etc.).

development, crafting reproducibility artifacts (still) requires human–agent collaboration. Let me try to articulate this division of labor and effective interaction patterns I have observed.

Building good tools still requires domain knowledge, which I believe should remain human-driven, at least for some time. A developer must decide what the right abstractions are, which operations deserve codification into stable entry points, which parameters should be exposed, which details should be hidden, and what counts as a valid output. These decisions are not merely matters of syntax or packaging, but rather reflect a deep understanding of the task itself. In the case of Anserini, for example, deciding that prebuilt indexes, topic registries, search execution, and evaluation commands should be exposed as separate tools requires knowledge of how information retrieval experiments are typically conducted. An agent can help implement such tools, but the conceptual design should remain the product of human thinking.[15]

Skills are different. Since skills are written for agents, they should be shaped by the way agents understand the markdown files and how the tools are actually used. What I have found helpful is to engage in iterative cycles of human–agent dialogue. I begin a session by teaching the agent how to use the tools, articulating exactly what it needs to learn: what this tool is for, when to use it, what parameters to specify, etc. I ask the agent to run the tool and guide it to interpret the output: what's expected, how to check reference values, etc. At the end, I ask the agent to capture everything it has learned into a skill for itself. The approach combines the best of both worlds: human input and agent self-reflection.

Then, I iterate. In a new session, I ask the agent to reproduce a result from a clean checkout using the skill it just documented. I observe the execution trace to understand how the skill is triggered, what files the agent reads, what commands it tries, how it discovers the right parameters, etc. Even more important are the failure modes: where the agent gets stuck, where the trajectory veers off course, etc. These observations are then used to improve the artifact in consultation with the agent itself. I propose modifications and ask the agent if it's a good idea. For example:

- Does it make sense to add a note about this common failure scenario?
- Does it make sense to split reference material into smaller files?
- Does it make sense to repeat a directive multiple times?

Surprisingly, the agent does *not* blindly adopt my recommendations, but engages in well-reasoned technical discussions. For example, it might push back and say: no, I think the markdown is currently concise enough to keep as one file or that redundancy is fine because it is worthwhile to repeat an important point in multiple sections. After these discussions, I ask the agent to update its skill accordingly. I see no reason to second-guess the agent, and I almost never modify skills directly. Finally, I start over and run the reproduction process again. If the new agent succeeds more directly, I can conclude that artifact has become more legible. Rinse, repeat.

In fact, this may be the most important practical lesson from my recent experience: Do not guess what agents need. Ask agents to perform the task and watch what happens. The transcript is diagnostic data: steer the agent based on what it does and ask it to update its own skill. Furthermore, I have found it productive to ask the agent to be self-reflective, for example, to tell

[15]It is certainly the case that an agent has knowledge about empirical IR methodologies and experimental workflows, and can be helpful in surveying what similar projects have done. However, there are many alternative implementations of the same concepts, and I believe that users should remain in the loop to weigh alternatives and to exercise judgment (at least for now).

me what's confusing, what's missing, etc. Asking the agent to diagnose deficiencies in the current implementation can be helpful in making the artifact more legible.

Finally, we need to answer the question: which agent? Specifically, which models and harnesses are we going to support? Developers need to consider these questions in the iterations described above: this is no different than deciding which operating systems and architectures a reproducibility artifact will support (see discussions in Section 4.2). There will always be a tradeoff between developer effort and generalizability across agents, and achieving the desired balance is an exercise in scoping the reproducibility effort.

At this point, we have closed the loop. As developers, our goal is to build software artifacts that are more legible to agents, allowing them to reproduce empirical results with greater ease. It is only natural that this is accomplished *with* agents, and in some cases, *by* agents directly.[16]

6 Looking Ahead

I'll conclude with a look ahead, starting with the obvious question I have yet to directly address: What's different about reproducibility? In other words, since reproducibility is about building software artifacts, doesn't everything simply inherit from agentic software engineering in general? Is there anything different or special about reproducibility? Well, yes...

It's not a marketplace. For general software artifacts, there are typically alternatives that do the same or similar things, since software exists in a market and alternative implementations naturally arise to compete. In contrast, since many reproducibility efforts are associated with research papers that are evaluated in terms of novelty (among other criteria), the authors often hold a monopoly on the manifestation of a particular innovation (at least in the beginning).

This difference affects user behavior. In the "general marketplace", if a software artifact is difficult to use or not legible to agents, the user is likely to abandon it and try some other option. Attempts to reproduce research results typically do not have alternatives, as there is initially only one implementation available (from the authors). This makes it more likely that a user persists in working with not-so-easy-to-use reproducibility artifacts.

Agents potentially change this calculus, as they can work tirelessly until all avenues are exhausted. Giving agents something to work from, even broken or incomplete code, is likely better than nothing.[17] This is a win for science, allowing researchers to stand on the shoulders of others, even if the existing shoulders are a bit shaky.

Performance, scalability, security, etc. are less important. I suspect this will generate controversy, but for reproducibility, making it work is more important than making it performant,

[16] A natural question arises: What happens if we *don't* embrace these practices? What if we don't design reproducibility artifacts for agent legibility? Interestingly, the consequences aren't particularly dire. In my experience, even current agents have a remarkable ability to "figure things out" (steered by users if necessary). The downside, of course, is more token and resource usage. Or we can just wait for the next model, which may obviate many of my recommendations here anyway. The scary thought is that hygiene is primarily intended for humans, and as agents become increasingly capable, all these seemingly important issues matter less and less.

[17] Assuming non-malicious and non-adversarial intent from the developers.

scalable, secure, etc. This means that janky wrappers, repeated copying and munging of data, inefficient algorithms, security vulnerabilities, etc. are all okay. With agents, reimplementations can occur with minimal user supervision, at the (mere) cost of consuming more tokens and other resources. The original deficient reproducibility artifact can serve as a reference for a completely new implementation that has better architecture, is more scalable and secure, etc. Again, something is better than nothing (I argue)—and this is a distinguishing characteristic of reproducibility: an artifact can be useful even if it does not satisfy any of the other desiderata we typically associate with high-quality software.

Slop is okay. To further dial up the spice level, I claim that slop is okay as long as the agent is able to successfully reproduce the desired result. We can treat reproductions as black box abstractions: if the input and output behavior match the expected results, then the effort succeeds. The source paper provides integration tests for the reproducibility artifact. Building on the previous point, it doesn't matter if the code is janky, slow, insecure, etc.

But how do we know if the artifact is actually implementing the purported innovation? We don't, but this is where the characterizations in Section 2 come into play: is it possible, is it easy, and is it interesting? If an artifact reproduces empirical results in an uninteresting way (e.g., trivially hard-coding input–output relationships), this will surely come to light when the user attempts to generalize or to build on the results, which leads to...

My final argument, the boss fight:

Reproducibility is a major unlock in the skill tree of human civilization.

Bet you didn't see such a bold claim coming. In Lin [2022], I had diagnosed the challenge of reproducibility as primarily an issue of culture. Allow me to articulate why I was previously wrong.[18] The tl;dr went like this: reproducibility is an ideal that no researcher would dispute in the abstract, but when aspirations meet the cold hard reality of the academic grind, reproducibility often loses out. The solution, I believed, was better alignment: instead of treating reproducibility efforts as altruistic, we should appeal to self-interest; cf. Gent [2013]. If we believe in reproducibility as benefiting ourselves, in enabling more reliable experiments, faster iterations, more publications, etc., then we are more likely to engage in reproducibility efforts. Let's start with better repeatability by building solid infrastructure, and then add in the element of sharing: this naturally yields reproducibility. Thus, in my diagnosis, self-interest plus expectations of reciprocity (which motivates sharing) would lead to more reproducibility artifacts.

However, I have since realized that this line of argumentation builds on a false premise, that software artifacts for reproducibility are second-class citizens, "supplemental" to the thing that actually "counted"—which for many is a publication in a top-tier venue. What if this weren't the case? What if the software artifact itself was the thing that actually counted? Then, obviously, the artifact would "know" how to reproduce the results it purports to encapsulate. This, of course, is not a new idea, and tracks the vision of an "executable paper" dating back decades [Claerbout and Karrenbach, 1992; Strijkers et al., 2011; Howe, 2012; Gent, 2013; Dittrich and Bender, 2015;

[18] In fact, I was given the opportunity to express an opinion [Lin, 2018], and then later to recant it [Lin, 2019], in this very venue.

Chirigati et al., 2016]. With agents, such a vision is within reach [Liu et al., 2026]! Of course, this will require changes in bean counting, much to the dismay of academic ~~bureaucrats~~ leaders, but given the growing obsolescence of academia and the rise of alternative measurements of impact such as arXiv citations and GitHub stars, we are already well on our way to this new reality. The untenable future of academic peer review further speeds us along this path.[19]

Looking further ahead, it is clear to me that we are merely at a convenient save point in the game of civilization. While the path ahead remains shrouded in the fog of war, the endpoint of the journey is clear. Assuming everything "goes right",[20] in the near future, we will let agents loose in vast datacenters and ask them to solve the biggest challenges facing humanity: curing cancer, generating abundant clean energy, mitigating climate change, etc. The implementation is yet unknown, but the input/output behavior of these abstractions is clear: problem statements and energy go in, solutions come out. One manifestation of this vision was already laid out by Dario Amodei back in 2024;[21] here I am not saying anything new.

Closer to the reality of tomorrow, let me return to a point from the introduction. I had argued that, at least today, reproducibility lies in the domain of agents, whereas the initiative of figuring out "what to do next" resides with users. Beyond reproducibility, user–agent collaborations can engage in attempts to replicate (i.e., independently implement artifacts from scratch), generalize (e.g., transfer to other domains, datasets, etc.), extend (e.g., improve on the innovations), and so on.[22] That's today. It is inevitable in the near future that agents will drive all these efforts and propose novel ideas on the path to recursive self-improvement [Si et al., 2024; Tao, 2025; Bubeck et al., 2025; Lyu et al., 2026; Lu et al., 2026; Ghareeb et al., 2026; Gottweis et al., 2026].

What happens then? Progress will remain constrained by the diffusion of innovation—in this case, *from agents to other agents*. I see two possible scenarios: The future world of AI may be dominated by a very small number of very large players running many copies of a small number of agents (even if the agents assume different roles to collaborate). As an alternative, we may see the emergence of a rich AI ecosystem with many diverse agents, large and small, general and specialized, created by a moderately large number of organizations from different countries with wide-ranging motivations, datasets, budgets, etc., all collaborating and competing to tackle challenges and solve problems.

Even setting aside all the concomitant dangers associated with centralization of power, in the first scenario, we wouldn't need reproducibility: we'd simply clone the agents. As soon as one agent discovers an innovation, it can easily diffuse across the entire fleet, since they're mostly copies of each other. However, the second scenario plays out very differently: in a diverse ecosystem, an agent that has discovered a secret still needs to communicate and teach the other agents in service of the larger goal—that is, make its findings reproducible.

I vastly prefer the second scenario with a rich ecosystem of diverse agents. In it, agentic reproducibility is the unlock.

[19] I recognize the self own here: what I'm advocating *is*, in fact, a change in culture.

[20] Alignment, safety, etc.; see Amodei's essay on the adolescence of technology.

[21] Amodei's essay on machines of loving grace

[22] Countering the arguments made by Drummond [2009]: with agents, reproducibility, replicability, generalizability, transferability, reusability, etc. all start to blend together, and the only substantive difference is the starting point: is there a reproducibility artifact that can provide a reference implementation? I agree, of course, that it is desirable to recreate empirical results with completely independent artifacts, but surely having a reference is better than not having one?

Acknowledgments

This research was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada. I'd like to thank Charlie Clarke, Yijun Ge, Lingwei Gu, Nour Jedidi, Neng Li, Amine Mhedhbi, Ronak Pradeep, and Sahel Sharifymoghaddam for helpful comments on previous drafts.

References

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. EvoSkill: Automated skill discovery for multi-agent systems. *arXiv:2603.02766*, 2026.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *arXiv:1611.09268v3*, 2018.

Sébastien Bubeck, Christian Coester, Ronen Eldan, Timothy Gowers, Yin Tat Lee, Alexandru Lupsasca, Mehtaab Sawhney, Robert Scherrer, Mark Sellke, Brian K. Spears, Derya Unutmaz, Kevin Weil, Steven Yin, and Nikita Zhivotovskiy. Early science acceleration experiments with GPT-5. *arXiv:2511.16072*, 2025.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. BrowseComp-Plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv:2508.06600*, 2025.

Fernando Chirigati, Rémi Rampin, Dennis Shasha, and Juliana Freire. ReproZip: Computational reproducibility with ease. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD 2016)*, pages 2085–2088, 2016.

Jon F. Claerbout and Martin Karrenbach. Electronic documents give reproducible research a new meaning. In *SEG Technical Program Expanded Abstracts*, pages 601–604, 1992.

Josh Devins, Julie Tibshirani, and Jimmy Lin. Aligning the research and practice of building search applications: Elasticsearch and Pyserini. In *Proceedings of the 15th ACM International Conference on Web Search and Data Mining (WSDM 2022)*, pages 1573–1576, 2022.

Jens Dittrich and Patrick Bender. Janiform intra-document analytics for reproducible research. *Proceedings of the VLDB Endowment*, 8(12):1972–1975, 2015.

Chris Drummond. Replicability is not reproducibility: Nor is it good science. In *Proceedings of the 4th Workshop on Evaluation Methods for Machine Learning at ICML*, 2009.

Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. SPLADE: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pages 2288–2292, 2021.

Yijun Ge, Zibo Guo, Sahel Sharifymoghaddam, and Jimmy Lin. MCP servers for Pyserini and RankLLM: Enabling agentic retrieval-augmented generation. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2026)*, 2026.

Ian P. Gent. The recomputation manifesto. *arXiv:1304.3674*, 2013.

Ali Essam Ghareeb, Benjamin Chang, Ludovico Mitchener, Angela Yiu, Caralyn J. Szostkiewicz, Dmytro Shved, Gavin J. Gyimesi, Jon M. Laurent, Samantha M. Wright, Muhammed T. Razzak, Andrew D. White, Silvia C. Finnemann, Michaela M. Hinks, and Samuel G. Rodriques. A multi-agent system for automating scientific discovery. *Nature*, 10.1038/s41586-026-10652-y, 2026.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, Anil Palepu, Keran Rong, Ryutaro Tanno, Khaled Saab, Fan Zhang, Jacob Blum, Andrew Carroll, Kavita Kulkarni, Nenad Tomašev, Dina Zverinski, Ivor Rendulic, Elahe Vedadi, Florian Hasler, Luka Rimanic, Marina Boia, Ivan Budiselic, Ben Feinstein, Mathias Bellaiche, Tom Sheffer, Jan Freyberg, Jeremy Ratcliff, Ottavia Bertolli, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Yuan Guan, Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R. D. Costa, José R. Penadés, Gary Peltz, Yossi Matias, James Manyika, Demis Hassabis, Yunhan Xu, Pushmeet Kohli, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Accelerating scientific discovery with co-scientist. *Nature*, 10.1038/s41586-026-10644-y, 2026.

Bill Howe. Virtual appliances, cloud computing, and reproducible research. *Computing in Science & Engineering*, 14(4):36–41, 2012.

Tz-Huan Hsu, Jheng-Hong Yang, and Jimmy Lin. Rethinking agentic search with Pi-Serini: Is lexical retrieval sufficient? *arXiv:2605.10848*, 2026.

Xu Huang, Junwu Chen, Yuxing Fei, Zhuohan Li, Philippe Schwaller, and Gerbrand Ceder. CASCADE: Cumulative agentic skill creation through autonomous development and evolution. *arXiv:2512.23880*, 2025.

Carlos Lassance, Hervé Déjean, Thibault Formal, and Stéphane Clinchant. SPLADE-v3: New baselines for SPLADE. *arXiv:2403.06789*, 2024.

Gavin Leech, Juan J. Vazquez, Niclas Kupper, Misha Yagudin, and Laurence Aitchison. Questionable practices in machine learning. *arXiv:2407.12220*, 2024.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Binxu Li, Qunhong Zeng, Di Wang,

Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianyu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han-chung Lee. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv:2602.12670*, 2026a.

Zhuofeng Li, Haoxiang Zhang, Cong Wei, Pan Lu, Ping Nie, Yi Lu, Yuyang Bai, Shangbin Feng, Hangxiao Zhu, Ming Zhong, Yuyu Zhang, Jianwen Xie, Yejin Choi, James Zou, Jiawei Han, Wenhu Chen, Jimmy Lin, Dongfu Jiang, and Yu Zhang. Beyond semantic similarity: Rethinking retrieval for agentic search via direct corpus interaction. *arXiv:2605.05242*, 2026b.

Jimmy Lin. The neural hype and comparisons against weak baselines. *SIGIR Forum*, 52(2):40–51, 2018.

Jimmy Lin. The neural hype, justified! A recantation. *SIGIR Forum*, 53(2):88–93, 2019.

Jimmy Lin. A proposed conceptual framework for a representational approach to information retrieval. *arXiv:2110.01529*, 2021.

Jimmy Lin. Building a culture of reproducibility in academic research. *arXiv:2212.13534*, 2022.

Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pages 2356–2362, 2021.

Jimmy Lin, Arthur Haonan Chen, Carlos Lassance, Xueguang Ma, Ronak Pradeep, Tommaso Teofili, Jasper Xian, Jheng-Hong Yang, Brayden Zhong, and Vincent Zhong. Gosling grows up: Retrieval with learned dense and sparse representations using Anserini. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2025)*, pages 3223–3233, Padua, Italy, 2025.

Jiachen Liu, Jiaxin Pei, Jintao Huang, Chenglei Si, Ao Qu, Xiangru Tang, Runyu Lu, Lichang Chen, Xiaoyan Bai, Haizhong Zheng, Carl Chen, Zhiyang Chen, Haojie Ye, Yujuan Fu, Zexue He, Zijian Jin, Zhenyu Zhang, Shangquan Sun, Maestro Harmon, John Dianzhuo Wang, Jianqiao Zeng, Jiachen Sun, Mingyuan Wu, Baoyu Zhou, Chenyu You, Shijian Lu, Yiming Qiu, Fan Lai, Yuan Yuan, Yao Li, Junyuan Hong, Ruihao Zhu, Beidi Chen, Alex Pentland, Ang Chen, Mosharaf Chowdhury, and Zechen Zhang. The last human-written paper: Agent-native research artifacts. *arXiv:2604.24658*, 2026.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of AI research. *Nature*, 651:914–919, 2026.

Yougang Lyu, Xi Zhang, Xinhao Yi, Yuyue Zhao, Shuyu Guo, Wenxiang Hu, Jan Piotrowski, Jakub Kaliski, Jacopo Urbani, Zaiqiao Meng, Lun Zhou, and Xiaohui Yan. EvoScientist: Towards multi-agent evolving AI scientists for end-to-end scientific discovery. *arXiv:2603.08127*, 2026.

Irina Matveeva, Chris Burges, Timo Burkard, Andy Laucius, and Leon Wong. High accuracy retrieval with multiple nested ranker. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2006)*, pages 437–444, Seattle, Washington, 2006.

Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. Multi-stage document ranking with BERT. *arXiv:1910.14424*, 2019.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, Maohao Shen, Vishy Tirumalashetty, George Lee, Jiawei Han, Tomas Pfister, and Chen-Yu Lee. SkillOS: Learning skill curation for self-evolving agents. *arXiv:2605.06614*, 2026.

Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models. *arXiv:2101.05667*, 2021.

Kirk Roberts, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, Kyle Lo, Ian Soboroff, Ellen Voorhees, Lucy Lu Wang, and William R. Hersh. TREC-COVID: Rationale and structure of an information retrieval shared task for COVID-19. *Journal of the American Medical Informatics Association*, 27(9):1431–1436, 2020.

Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. Can LLMs generate novel research ideas? A large-scale human study with 100+ NLP researchers. *arXiv:2409.04109*, 2024.

Shuzheng Si, Haozhe Zhao, Yu Lei, Qingyi Wang, Dingwei Chen, Zhitong Wang, Zhenhailong Wang, Kangyang Luo, Zheng Wang, Gang Chen, Fanchao Qi, Minjia Zhang, and Maosong Sun. From context to skills: Can language models learn from context skillfully? *arXiv:2604.27660*, 2026.

Rudolf Strijkers, Reginald Cushing, Dmitry Vasyunin, Cees de Laat, Adam S.Z. Belloum, and Robert Meijer. Toward executable scientific publications. *Procedia Computer Science*, 4:707–715, 2011.

Terence Tao. Machine-assisted proof. *Notices of the American Mathematical Society*, 72(1):6–13, 2025.

Ellen M. Voorhees, Shahzad Rajput, and Ian Soboroff. Promoting repeatability through open runs. In *Proceedings of the 7th International Workshop on Evaluating Information Access (EVIA 2016)*, pages 17–20, Tokyo, Japan, 2016.

Lidan Wang, Jimmy Lin, and Donald Metzler. A cascade ranking model for efficient ranked retrieval. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2011)*, pages 105–114, Beijing, China, 2011.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv:2602.08234*, 2026.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-Pack: Packed resources for general Chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2024)*, pages 641–649, Washington, D.C., 2024.

Peilin Yang, Hui Fang, and Jimmy Lin. Anserini: Enabling the use of Lucene for information retrieval research. In *Proceedings of the 40th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2017)*, pages 1253–1256, 2017.

Peilin Yang, Hui Fang, and Jimmy Lin. Anserini: Reproducible ranking baselines using Lucene. *Journal of Data and Information Quality*, 10(4):Article 16, 2018.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web agents can self-improve by discovering and honing skills. *arXiv:2504.07079*, 2025.

Yingli Zhou, Wang Shu, Yaodong Su, Wenchuan Du, Yixiang Fang, and Xuemin Lin. A comprehensive survey on agent skills: Taxonomy, techniques, and applications. *arXiv:2605.07358*, 2026.