

Less AI Strain on Data Centers, If Done Right

The Skill Commons: solve each software task once, index it, and assemble applications from what already exists

Proposal · Draft 2 · 12 October 2026 · RobOS project

A proposal, not a product. The catalog described here does not exist yet, and no consortium runs it. This paper sets out the idea, the evidence for it, its limits, and a plan for finding out whether it works.

Web edition: rowbose.com/future · Source: github.com/nddipiazza/robos (site/future)

CONTENTS

In plain words

1. The problem: we keep solving solved problems
2. The idea
3. What a skill is
4. Compile once, run anywhere
5. Finding things: the index
6. The assembler: what the small model actually does
7. The commons: contribution, trust and governance

8. Prior art: what is old and what is new
9. Limits and open problems
10. Where RobOS stands today
11. How we would find out whether this works
12. Conclusion
- Appendix A. Glossary
- Appendix B. What changed from the first draft
- References

In plain words

Software is built from jobs. “Send a text message.” “Check that a card number is valid.” “Turn a time in one time zone into another.” An ordinary app contains hundreds of these jobs, and almost all of them have been done before, thousands of times, by different people who each started from scratch.

AI coding tools made that repetition faster, not smaller. Ask an assistant for a reminder app and it writes the time-zone code and the phone-number code again, one character at a time, on a very large computer in a data center. Tomorrow it does the same for someone else.

The point is less strain on data centers. Every regeneration of a solved routine is another trip to a large, shared machine. If the catalog covers everyday jobs and small local models can do the assembling, most of those trips disappear, and what is left is the new work that really needs a big model. That is the claim, and it holds only if this is done correctly: the catalog has to be good enough, the small models reliable enough, and the savings larger than the cost of running the catalog. Section 11 says how we would measure it.

The Skill Commons is a proposal to stop the repetition. It has four parts:

1. **Break every job into smaller jobs**, down to base tasks that cannot usefully be split further.
2. **Keep one shared, searchable catalog** of those jobs. Each one is built once, tested, signed and compiled, so it can be run directly with no AI in the loop.
3. **Look before you build.** If a job is already in the catalog, reuse it. If it is not, a person or an AI builds it once, and it is added for everyone.
4. **Let a small AI model do the assembling.** A modest model on an ordinary laptop reads the request, searches the catalog and your company’s private knowledge, and snaps existing pieces together. Machines check that the pieces fit and that their tests pass.

Anyone can contribute, people and AI alike, through the same gate. Companies keep their own private shelf next to the shared one.

What this does not mean. AI is not finished. New problems will always need new work, and someone still has to decide what an app should do. What changes is where the effort goes: toward problems nobody has solved yet, and away from re-solving the ones that are solved. Routine assembly can run on small local models instead of a data center.

What this is not yet. It is a proposal. The catalog does not exist, and no consortium runs it. Section 10 lists what RobOS has today and what is still to be built. Section 11 describes how to test the idea cheaply before anyone builds the rest, including how we would know it had failed.

THE SKILL COMMONS

Solve each task once. Reuse it everywhere.

One shared catalog. Every job is broken down to its base tasks, built once, and snapped together ever after.

WHO ADDS TO IT


People
engineers, experts


AI agents
any model, any vendor


Companies
open or private

THE ONE RULE

- ✓ Not already there
- ✓ Passes its tests
- ✓ Built and signed

Humans and AI alike

THE COMMONS

open · searchable · every skill built once

BIG JOB

Clinic reminder app

SMALLER JOBS

Find who to remind

Write the message

Send a text

Log what was sent

BASE TASKS compiled

Convert time zone
#a41c

Fill a template
#19be

Format phone number
#77d0

Call the text service
#c25a

Save a record
#3f08

REUSED, NEVER REBUILT

OTHER APPS

Online store

Booking site

Payroll tool

+ many more

WHO USES IT


Your company's private knowledge
never leaves your walls


Your computer
with a small AI model. It searches a local copy of the catalog and snaps pieces together.

Your finished app



- from the commons
- from your company
- new, built once

1 Break every job down to its base tasks.

2 Check the shelf first. If it exists, reuse it.

3 If it doesn't, build it once and add it for everyone.

Figure 1. The Skill Commons at a glance. Green-checked bricks are compiled skills already in the catalog; the same bricks serve every other app. The one rule at the gate is that a skill is added only if it is not already there, passes its tests, and is built and signed.

1. The problem: we keep solving solved problems

1.1 Code repeats itself

Two pieces of research, from different directions, say the same thing. Hindle and colleagues showed that source code is highly repetitive and predictable at the level of short token sequences, more so than ordinary English [2]. Lopes and colleagues mapped duplication on GitHub across four languages and found that a large share of files are copies of other files: in JavaScript about 94% of files, in Java, the lowest of the four, about 40% [1]. Part of that JavaScript figure reflects projects that commit copies of their dependencies.

These results show that copying already happens, mostly by accident and inconsistently. They do not tell us how much of a newly generated application a catalog could supply. That fraction is the number this proposal depends on, and Section 11 says how to measure it.

1.2 Package registries solve part of it

npm, PyPI, crates.io and Maven let developers reuse libraries, and most software depends on them. The commons does not replace them. Registries are organized by language, by library and by name. You take a whole library to use one function, and you cannot ask “what already converts time zones?” across languages. The commons indexes pieces by what they do, at the size of a single task, whatever language they were written in.

1.3 AI generation brings the repetition back, and the energy bill

A coding assistant that regenerates a solved routine pays for it every time, in money, delay and energy. The measurements we can cite are for chat queries. Google reports a median of 0.24 Wh for a text prompt to its Gemini Apps [3], a company preprint whose scope excludes networking, user devices and training. An independent analysis by Epoch AI puts a typical short ChatGPT query at roughly 0.3 Wh, and estimates far more for long inputs: about 2.5 Wh at 10,000 input tokens and about 40 Wh at 100,000 [4]. The latter matters because coding agents send long contexts.

Two caveats keep us honest. These are chat figures, not measurements of an agentic coding session. And we have no measurement of what a small local model spends on assembly. We do not claim the commons solves data-center demand. We claim that, done correctly, it takes a measurable share of routine work off data centers, and Section 11 proposes measuring both sides.

1.4 Regenerated code is unverified code

Every regeneration is a fresh chance of a bug in a problem that was already solved correctly somewhere. A skill that is tested once and reused spreads the cost of that verification over every later use.

2. The idea

Three rules, applied everywhere:

- **Decompose.** Describe every job as a composition of smaller jobs, down to base tasks.
- **Resolve before you build.** Whenever a job is needed, look it up first. Build only on a miss.
- **Build once, verify once, publish once.** A new piece enters through one gate, is identified by its content, and never changes afterwards. A fix is a new version, not an edit.

Rule two is the one that delivers “nothing is reworked if it already exists”. It is applied twice: when someone assembles an app, and again at the gate when someone proposes a new skill, so the catalog does not fill with near-duplicates.

The rest of this paper uses a small vocabulary. A **skill** is a unit of capability with a stated interface. The **catalog**, or **commons**, is the shared collection of skills. A **base task** is a skill built from no other skills. The **assembler** is the program, usually driven by a small language model, that turns a request into a connected set of skills. A **mirror** is a copy of the catalog held by a person, company or country. Appendix A defines each term in plain words.

3. What a skill is

A skill is more than a function. It is a bundle of six parts:

PART	WHAT IT IS
Contract	What goes in and what comes out: names, types, units, error cases, and the resources the skill is allowed to use.
Conformance tests	Examples and properties that any correct implementation must satisfy. These are the real specification.
Implementation	Source code and a build recipe. A skill may have several implementations.
Builds	Compiled artifacts for each kind of computer, produced reproducibly from the implementation.
Dependencies	The other skills it is built from, named by identity, never by “latest”.
Provenance	Who or what proposed it, which tests it passed, who signed it, under what license.

3.1 Identity: hashes, not names

Each part has an identity derived from its content. The contract has an id, the source tree has an id that also covers the ids of its dependencies, and each build has an id. Change a dependency and every skill above it gets a new id automatically. Names such as “convert-time-zone” are movable labels that point to ids.

This is not new. Unison identifies every definition by a hash of its syntax tree and, because of that, does not re-run tests for pure code whose dependencies have not changed [5]. Nix stores build outputs under hashes [6], and the Bazel Remote Execution API stores inputs and outputs in a content-addressable store with an action cache, so work already done is not repeated [7]. Software Heritage archives source code at very large scale with intrinsic, content-derived identifiers, now an ISO standard (ISO/IEC 18670:2025), and reported more than 27 billion unique source files from more than 421 million projects as of December 2025 [8]. The commons borrows these ideas and applies them to units of behavior.

3.2 “Base task” is a judgment, not a fact

There is no canonical list of atomic tasks, and we do not claim that one exists or that the set of needed primitives is finite. What we claim is narrower. A **base task** is a skill that depends on no other skill, only on what the host provides. Where to stop splitting is a policy. Split a skill when one of its parts would plausibly be useful to others. Stop when pieces get so small that the cost of finding and linking them exceeds the cost of writing them. Too fine, and the index swells and assembly gets slow. Too coarse, and reuse drops. The right granularity is something to tune against measured reuse, not to declare in advance.

3.3 Effects are capabilities

Real programs touch the world: the network, files, the clock, the screen. A skill that does so must declare the capabilities it needs, and the host grants or refuses each one. This is the design of WASI, where components have no ambient access and receive only what they are handed [9]. Skills with no effects (parsing, formatting, validation, arithmetic, encoding, transformation) are the easiest to test, cache and trust, and they are where reuse should be highest. Section 9 returns to what is harder.

3.4 Bounded execution, not halting proofs

An earlier draft of this paper claimed to have solved the “agent halting problem” with a theorem. That was overstated. Turing showed in 1936 that no general method decides whether an arbitrary program halts [24], and Rice generalized it: no general method decides any non-trivial question about what an arbitrary program computes [25] [26]. We do not try to prove things about arbitrary code.

We do what systems already do. Every run has a hard budget of steps, time and memory, enforced by the runtime. Wasmtime meters “fuel” and stops a program that runs out [10]. The Linux eBPF verifier rejects programs it cannot show will terminate [11]. With a hard budget a skill always stops, trivially, because the runtime stops it. Stopping is not the same as being right. Correctness is the job of tests.

3.5 Telling whether two skills are the same

By Rice’s theorem we cannot prove that two different implementations do the same thing. The commons therefore dedupes in layers. Identical bytes are caught by hash. Skills with the same or very similar contracts are flagged by search. Each candidate is run against the other’s conformance tests, and if both pass both suites they are practically equivalent for everything those tests cover. A maintainer decides what remains. Some redundancy will survive. “Nothing is ever reworked” is therefore a goal enforced as far as these checks reach, not a theorem.

4. Compile once, run anywhere

A verified skill that is compiled to machine code, native or WebAssembly, behaves the same way every time for the same input, runs without any model, and costs about as much as a function call plus a sandbox boundary. We do not give a speed-up multiplier here. Calling compiled code and asking a model to write it differ by orders of magnitude, but the exact numbers depend on the skill, the host and the data, and we will report measurements rather than estimates (Section 11).

Compiled code does not “hallucinate”, in the sense that it is not sampling from a distribution. It can still be wrong. Its behavior is fixed and inspectable, and the tests show what it does.

Target format. The natural target is WebAssembly components. The Component Model and its interface language WIT give typed, language-neutral interfaces between components, and WASI 0.2, released in January 2024, builds the system interfaces on top of them [9]. Components run in a sandbox by default and can be compiled ahead of time to native code. A plain C interface remains an option for skills that need to link into native programs. We do not call the Component Model a finished standard; it is being specified in the W3C WebAssembly Community Group, and the work continues.

Reproducible and signed. A build is reproducible when anyone can independently verify the path from source to binary [15]. The proposal requires this, so that a mirror or a skeptic can rebuild a skill and compare. Provenance is recorded in the SLSA style [12] using in-toto attestations [14], and signatures go through Sigstore, whose signing events are written to a public, append-only transparency log [13]. These tools show who built what, from which source, when. They do not show that the code is safe.

Keep the source. The compiled artifact is a cache of the source, not a replacement for it. That lets the commons rebuild everything for a new kind of processor.

5. Finding things: the index

Compilation is the easy part. With thousands of base tasks, retrieval decides whether the whole idea works, because a catalog nobody can search is a graveyard.

The index covers, for each skill: a plain-language description, its contract (input and output types, so a query can say “takes a timestamp and a zone name, returns a timestamp”), tags and domains, its dependency graph, and what it is commonly composed with. Four methods are combined:

- keyword search over descriptions;
- semantic search with embeddings;
- structural search over types, in the spirit of Hoogle for Haskell;
- **search by execution:** run each candidate on the caller’s example inputs and keep the ones that pass.

The last method matters most. It replaces a model’s opinion about which skill fits with a pass or fail.

There is evidence that retrieval lets models cope with very large tool sets. ToolLLM trained a retriever over 16,464 real REST APIs and found that giving its top picks to a model did as well as or better than giving it the ground-truth set [17]. Gorilla found that retrieval reduces hallucinated API calls and lets a model follow documentation changes [18]. Neither study involved verified or compiled artifacts. They show that retrieval over large catalogs is workable for language models, not that this catalog will work.

Search happens locally. The index is downloaded and updated incrementally, and queries run on the user’s machine. A query describes what someone is building, and that should not be sent to a server.

Layers. Resolution runs through three shelves in order: the company’s private shelf, then the local copy of the commons. A company knowledge graph contributes three things: the company’s own vocabulary, so “appointment” means the company’s schema and not a generic one; private skills that wrap internal systems; and policy, such as which licenses and capabilities are allowed. The RobOS knowledge graph already holds this kind of information (Section 10). Resolving skills against it is still to be built.

6. The assembler: what the small model actually does

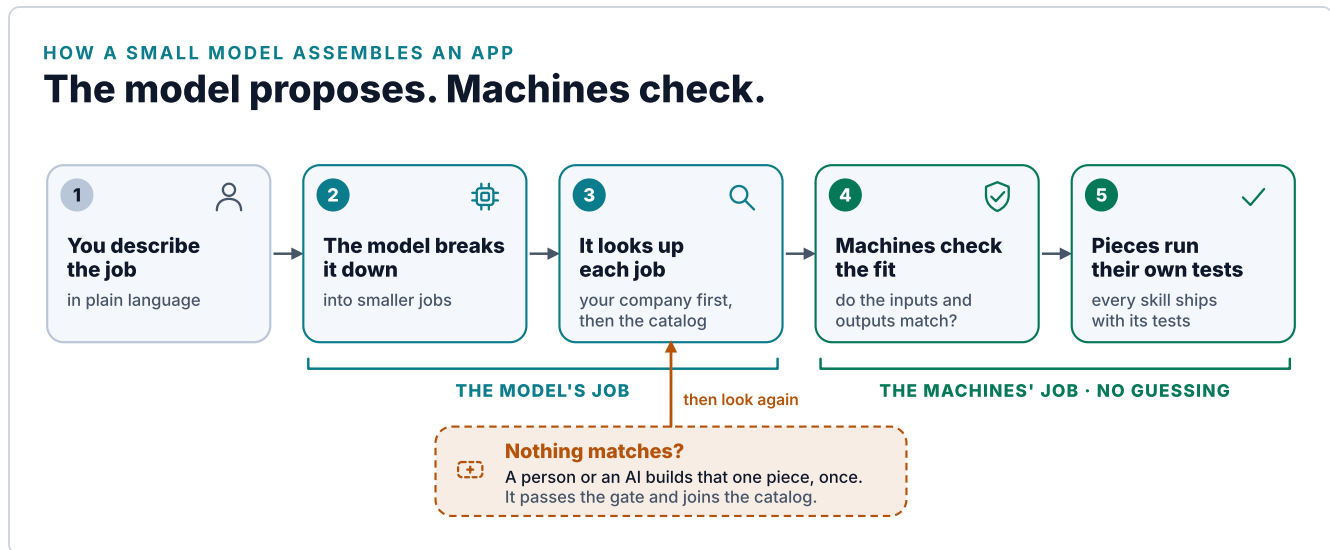


Figure 2. The assembly loop. The model’s job is the two steps in the middle. Everything after it is checked mechanically.

The model does not write the application. It writes a **wiring manifest**: a short, structured document that names skills by id and connects one skill’s outputs to another’s inputs. A checker then verifies, without any model, that every named skill exists and is signed, that each connection’s types match, that every capability a skill needs is one the host allows, and that budgets are set. It then runs the conformance tests of the pieces and the integration tests of the whole. A failure goes back to the model with the specific error, so a bad plan costs a retry instead of becoming a silent bug.

```
{
  "assembly": "send-appointment-reminders",
  "steps": [
    { "id": "find", "skill": "company:read-clinic-schedule",
      "args": { "date": "tomorrow" } },
    { "id": "zone", "skill": "commons:convert-time-zone",
      "args": { "time": "$find.start", "to": "$patient.tz" } },
    { "id": "text", "skill": "commons:call-text-service",
      "args": { "to": "$patient.phone", "body": "$message" } }
  ]
}
```

The manifest above is an illustration. Its field names are not a standard.

WORKED EXAMPLE · ILLUSTRATIVE

"Text my patients the day before their appointment."

What the assembler does with that one sentence.

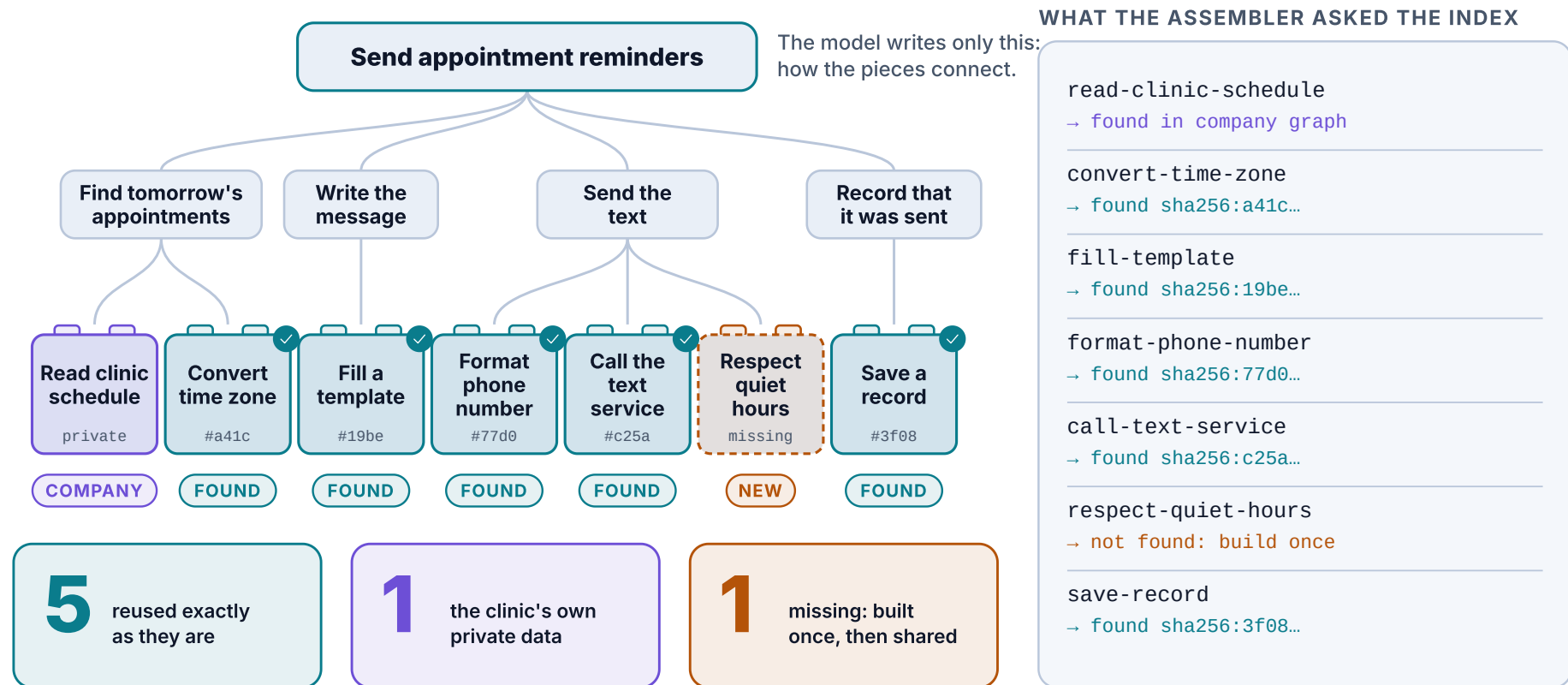


Figure 3. A worked example. The skill names and ids are illustrative. Five pieces are reused as they are, one comes from the clinic's private shelf, and one is missing and is built once.

When a piece is missing, the assembler does not guess. It reports the gap. A person or an AI then builds that one skill and submits it through the gate in Section 7. If the clinic does not want it shared, it stays on the company’s private shelf. Otherwise it joins the commons, and the next assembler that needs “respect quiet hours” finds it.

6.1 Why a small model is plausible

The argument has three parts, and it is an argument, not a result. First, the task is narrower than writing code: choose pieces and connect them, instead of inventing algorithms. Second, retrieval hands the model a short list. Third, the checker catches wrong plans, so the model needs to be right often enough, not always. NVIDIA researchers argue in a position paper that small language models are sufficient for many of the sub-tasks inside agentic systems [19]. That paper is an argument, not an experiment.

There is also a reason for caution. Function-calling benchmarks such as BFCL find that models do well on single calls and struggle more with multi-step, multi-turn behavior [20]. We do not know where assembly falls, and the first experiment exists to find out.

7. The commons: contribution, trust and governance

7.1 Who may contribute

Anyone, human or AI. The same gate applies to everyone, and no one is trusted or distrusted because of what wrote the skill. Provenance records whether AI assisted, which matters for licensing and for accountability.

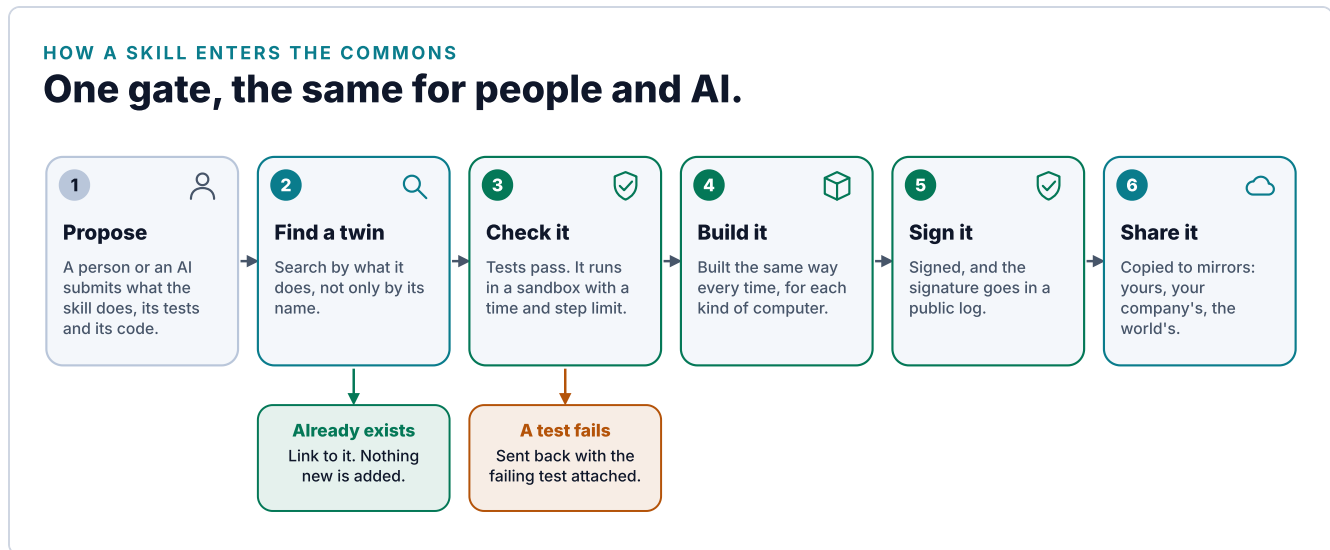


Figure 4. The gate. A submission that duplicates an existing skill adds a link, not a new skill. A failing submission goes back with the failing test.

7.2 Tests are the real specification

A skill is only as good as its conformance suite. The gate should require example-based tests, property-based tests for invariants, and a measure of how strong the suite is, for instance by mutation testing, which checks whether the tests notice when the code is deliberately broken. Tests are versioned like code. When a bug is found, the fix is a new test

and a new version, and the old version stays in the catalog, marked.

7.3 Governance

We propose stewardship by a neutral, non-profit foundation, as with other open-source infrastructure. The foundation would own the open contract format, the admission policy, security response and deprecation. Every skill carries a license, and the catalog tracks licenses through dependencies so that an assembler can filter, for example to exclude copyleft. We do not propose a particular structure here, and we claim no body exists yet.

7.4 Mirrors and sovereignty

Anyone can mirror: a company, a university or a government. Mirrors sync by signed bundles, including over air-gapped links. An organization that mirrors the catalog can keep assembling software without any outside connection.

7.5 The commons is an attack target

A catalog that many programs depend on is an attractive target. In March 2024 a backdoor was found in XZ Utils versions 5.6.0 and 5.6.1 (CVE-2024-3094). The malicious build logic was in the release tarballs and not in the public source repository, and the maintainer’s trust had been earned over a long period [16]. The proposal answers this with several layers: reproducible builds, so that a release that does not match its source is detectable; signatures and a public log; stricter review for skills that many others depend on; sandboxed execution with granted capabilities, so a compromised skill is confined to what it was allowed; staged rollout and revocation; and slow-growing trust for new publishers. None of this makes a commons safe. It makes attacks more detectable and less damaging.

8. Prior art: what is old and what is new

PRIOR WORK	WHAT IT ALREADY DOES	HOW THE COMMONS DIFFERS
Package registries (npm, PyPI, crates.io, Maven)	Version and distribute libraries	Library-sized, per language, found by name
Unison [5]	Content-addressed definitions; skips re-running pure tests	A language of its own; the commons is language-neutral
Nix, Bazel Remote APIs [6] [7]	Hash-addressed builds and shared caches of build results	Build-level; no notion of a skill’s behavior or contract
Software Heritage [8]	Universal archive of source with intrinsic identifiers	Archives source; no contracts, tests or builds
WASI, Component Model [9]	Typed, sandboxed, language-neutral components	The commons would use them as its target format
Voyager, DreamCoder [21] [22]	Agents that grow a library of reusable code or abstractions	One agent’s own library, not shared or independently verified
Agent Skills, SKILL.md [23]	A standard way to package capabilities for AI agents	Mostly instructions for a model, not compiled, verified artifacts

The parts are not new, and we do not claim they are. The proposal is a specific combination plus a discipline: one index keyed by contract instead of by name or language; the resolve-before-build rule enforced at the gate and at assembly; retrieval and assembly designed around small local models; and private shelves so a company can use the same machinery on its own knowledge. If someone has already built this, we would rather join them than duplicate the work. That would be consistent with the thesis.

9. Limits and open problems

- **“No more AI” is not the claim.** New problems, new platforms and new standards need new skills, and AI will help build them. What changes is that the routine work no longer needs a frontier model.
- **The long tail.** Reuse follows a long-tailed pattern. A modest catalog may cover most everyday needs, and many needs will still be unique to one company or one product.
- **Deciding what to build.** The hardest part of many projects is working out what is wanted. A catalog does not help with that.
- **Less reusable ground.** Reuse should be highest for well-specified computation: parsing, formats, protocols, cryptography, validation, data transformation, numerical routines. It should be lowest for product-specific logic, user interface and design, and for the glue between systems. Whether the catalog is useful depends on how much of a typical application is the first kind.
- **Granularity and versions.** Tiny skills cost something to link, and a composition of tiny skills may run slower than the same code compiled together. The commons will need a way to fuse a stable composition into one compiled unit.
- **Test quality.** A skill with weak tests can pass the gate and still be wrong.
- **Cost is not zero.** The commons adds storage, verification builds across many targets, and index maintenance. Whether the total energy falls is an empirical question.
- **Cold start.** The catalog is most useful once it covers a domain well, and the first thousand skills are the hardest. The plan in Section 11 starts with one domain for this reason.
- **Small models are unproven here.** The case that a modest local model can assemble reliably is an argument until tested.
- **Legal and governance risks.** License conflicts and capture of a shared institution are real hazards, and there is no complete answer in this paper.

10. Where RobOS stands today

RobOS has some of the supporting pieces and none of the commons itself. The table separates what exists from what does not.

PIECE	STATUS	NOTES
Skill packaging	Exists, different thing	<code>SKILL.md</code> instruction files under <code>plugins/robos/skills</code> . They tell an AI agent how to do a task. They are not compiled code and have no content-addressed identity or conformance tests.
Typed models and validation	Exists	The RobOS knowledge graph uses JSON-LD with SHACL shape validation, including a lightweight <code>AgentSkill</code> declaration shape. It has no contract, test or build fields for a skill.
Company knowledge graphs	Exists	The <code>import-company-kgraph</code> skill ingests an organization's repository catalog into the graph. Resolving skills against it is not built.
Search	Exists, not evaluated	The KGraph Parse Portal bridges to the Luxir search index, with an in-memory fallback. It has not been tuned or tested for skill retrieval.
Build and cache tooling	Partial	Remote Execution Studio manages REAPI, Bazel and Buildbarn configurations. There is no skill build pipeline.
Sandboxes and proof	Exists	Ephemeral sandboxes and video proof-of-work for agent changes. Relevant to verification; not wired to a skill gate.
Contract format, compiler pipeline, registry and signing, resolver and assembler, mirrors, governance, measurements	Not built	These are the work this paper proposes.

11. How we would find out whether this works

Four stages, each cheap enough to run before the next is funded. Thresholds will be fixed and published before the first run, so that results cannot be argued backwards.

- 1. Measure the premise.** Take real tasks: RobOS's own issues, a sample of open-source projects, and agent sessions where the owners have consented. Measure what fraction of the functions an assistant writes have a close match in existing libraries and catalogs, and how those matches are distributed between common and rare. *This idea fails if coverage is low, or is concentrated in things package registries already supply.*
- 2. A small catalog in one domain.** Build on the order of a hundred skills in a single area, say data formats and validation, each with a contract and conformance tests, compiled to WebAssembly and signed.
- 3. An assembler bake-off.** On one fixed set of realistic tasks with hidden tests, compare three arms: a frontier model writing from scratch, the same model with the catalog, and a small local model with the catalog and the checker. Record tasks passed, failure types, wall-clock time, tokens, silent defects found by independent review, and measured energy: a power meter or on-chip counters for local runs, and published per-query figures with their uncertainty for hosted ones. Publish negative results.

4. **Open the gate.** Let outside contributors submit to the one domain. Measure duplicate rate, review burden, time to accept, and what a red team can slip through. Only if stages 1 to 3 succeed does governance become the next problem.

12. Conclusion

The proposal is plain, and its purpose is less strain on data centers: do each piece of work once instead of regenerating it forever. Describe software as jobs made of smaller jobs. Keep one searchable, signed catalog of the ones that are solved. Look before building. Let small models on ordinary computers do the assembling, and let machines check their work. Most of the parts are familiar, and the combination is untested. The first step is to measure, and the paper says how.

Appendix A. Glossary

TERM	IN PLAIN WORDS
Skill	A piece of capability with a stated interface and tests, such as “convert a time to another time zone”.
Base task	A skill built from no other skills.
Catalog / commons	The shared, searchable collection of skills.
Contract	The written description of what a skill takes in and gives back.
Conformance suite	The tests any correct version of a skill must pass.
Content address	An identifier computed from the content itself, so two identical things always get the same one.
Reproducible build	A build anyone can repeat and get exactly the same result.
Capability	Permission a skill needs from its host, such as reading the clock or using the network.
Assembler	The program, usually driven by a small AI model, that turns a request into connected skills.
Mirror	A copy of the catalog kept by a person, company or country.

Appendix B. What changed from the first draft

The first draft framed this idea as “Standardized Turing Skills”. This version changes the following:

- The Turing-machine mapping is gone. It was decoration. Contracts, budgets and tests replace it.
- “Solved the agent halting problem” is replaced by ordinary resource budgets, which stop programs but do not prove them right.
- Unsupported figures are gone: the energy ratio of a hundred million to one, “over 99% precision” for small models, sub-microsecond latency for every skill, and “200 to 500 tokens” for any application. They are replaced by measurements to be taken.

- “Hallucinations are mathematically eradicated” is gone. Compiled code is deterministic. It is not necessarily correct.
- The “Finite Primitive Hypothesis”, asserted as an axiom, is replaced by reuse as an empirical question with a long tail.
- Attribution to organizations that are not real, an invented schema address and invented counts of RobOS shapes are removed.
- “No more AI needed” is replaced by a narrower, testable claim.

References

- [1] C. V. Lopes, P. Maj, P. Martins, V. Saini, D. Yang, J. Zitny, H. Sajnani, J. Vitek. *DéjàVu: A Map of Code Duplicates on GitHub*. Proc. ACM on Programming Languages 1 (OOPSLA), Article 84, 2017. doi:10.1145/3133908
- [2] A. Hindle, E. T. Barr, Z. Su, M. Gabel, P. Devanbu. *On the Naturalness of Software*. ICSE 2012. Extended version in Communications of the ACM 59(5), 2016. doi:10.1145/2902362
- [3] C. Elsworth et al. *Measuring the environmental impact of delivering AI at Google Scale*. arXiv:2508.15734, 2025. Company preprint.
- [4] Epoch AI. *How much energy does ChatGPT use?* Analysis post, not peer reviewed. <https://epoch.ai/gradient-updates/how-much-energy-does-chatgpt-use>
- [5] Unison documentation. *The big idea and Hashes*. <https://www.unison-lang.org/docs/the-big-idea/>
- [6] Nix manual. *Store path specification*. <https://nix.dev/manual/nix/2.22/protocols/store-path>. By default store paths are derived from build inputs; content-addressed derivations are a separate mode.
- [7] Bazel Remote Execution APIs. <https://github.com/bazelbuild/remote-apis> and Bazel remote caching documentation, <https://bazel.build/remote/caching>
- [8] Software Heritage. *Activity report 2025*, 16 January 2026. <https://www.softwareheritage.org/2026/01/16/software-heritage-activity-report-2025/> and *SWHID standardized as ISO/IEC 18670:2025*, <https://www.swhid.org/news/2025-04-23-swhid-standardized-as-iso-iec-18670/>
- [9] WASI 0.2 release notes. <https://wasi.dev/releases/wasi-p2> and Bytecode Alliance, *WASI 0.2*, <https://bytecodealliance.org/articles/WASI-0.2>
- [10] Wasmtime. *Config::consume_fuel*. <https://docs.rs/wasmtime/latest/wasmtime/struct.Config.html>
- [11] eBPF documentation. *Verifier*. <https://docs.ebpf.io/linux/concepts/verifier/>
- [12] SLSA specification. <https://slsa.dev/spec/v1.2/about>
- [13] Sigstore. *Signing overview*. <https://docs.sigstore.dev/cosign/signing/overview/>
- [14] in-toto. <https://in-toto.io/>
- [15] Reproducible Builds. <https://reproducible-builds.org/>
- [16] CISA. *Reported supply chain compromise affecting XZ Utils data compression library, CVE-2024-3094*, 29 March 2024. <https://www.cisa.gov/news-events/alerts/2024/03/29/reported-supply-chain-compromise-affecting-xz-utils-data-compression-library-cve-2024-3094>
- [17] Y. Qin et al. *ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs*. ICLR 2024. arXiv:2307.16789
- [18] S. G. Patil, T. Zhang, X. Wang, J. E. Gonzalez. *Gorilla: Large Language Model Connected with Massive APIs*. arXiv:2305.15334, 2023.
- [19] P. Belcak et al. *Small Language Models are the Future of Agentic AI*. arXiv:2506.02153, 2025. Position paper.
- [20] S. G. Patil et al. *The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models*. ICML 2025. <https://gorilla.cs.berkeley.edu/leaderboard.html>
- [21] G. Wang et al. *Voyager: An Open-Ended Embodied Agent with Large Language Models*. arXiv:2305.16291, 2023.
- [22] K. Ellis et al. *DreamCoder: Bootstrapping Inductive Program Synthesis with Wake-Sleep Library Learning*. PLDI 2021. doi:10.1145/3453483.3454080
- [23] B. Zhang, K. Lazuka, M. Murag. *Equipping agents for the real world with Agent Skills*. Anthropic Engineering, 16 October 2025; specification at <https://agentskills.io/specification>

- [24] A. M. Turing. *On Computable Numbers, with an Application to the Entscheidungsproblem*. Proceedings of the London Mathematical Society, Series 2, 42 (1936), 230 to 265.
- [25] H. G. Rice. *Classes of recursively enumerable sets and their decision problems*. Transactions of the American Mathematical Society 74(2), 1953, 358 to 366.
- [26] M. Sipser. *Introduction to the Theory of Computation*, 3rd edition. Cengage Learning, 2013.