



4th Wave Meets 1st Wave: Leveraging Modern AI in ACL2

Alessandro Coglio • Grant Jurgensen
Eric McCarthy • Eric Smith • Stephen Westfold



Kestrel Institute

ACL2 Workshop 2026

Summary

- What can the AI do for me?
- What are the different ways to use the AI?
- How can I use the AI securely?
- Cyber protections
- Plans and usage
- How to start using the AI today!

AI Success: Elliptic Curve Proofs

- Three difficult proofs:
  - Closure of point addition on Montgomery curves
  - Associativity of point addition on Montgomery curves
  - Associativity of point addition on Twisted Edwards curves
- So hard that we skipped them.
  - Had to include them as hyps in theorems – unsatisfying!
  - Hoped maybe an intern would prove them as a summer project
- Claude proved them automatically
  - Over 2000 lines of ACL2 proof!
  - Took a few hours
  - Cost a few dollars
  - Deleted those hyps from the theorems

AI Success: Elliptic Curve Proofs

here is something challenging. there are a few properties about elliptic curves that we delayed proving because they seemed very hard. see twisted-edwards-add-associativity and montgomery-add-closure and montgomery-add-associativity, all in books/kestrel/crypto/ecurve. do you think you can prove any of these? I think maybe none of these is a novel result, so probably there is a proof somewhere you could follow, maybe in a textbook. start with the one that seems easiest. if you want me to try to find a textbook proof for you, let me know, but maybe you can just prove it. the idea is to prove the fact as a theorem so we can remove the corresponding 0-ary functions as assumptions in further proofs.

From Claude:

More from Claude:

How the Montgomery proof goes:

- It uses polynomial certificates checked by Russinoff's SHNF normalizer, covering 7 generic chord/doubling configurations.
- One special configuration involves a point of order 2.
- The other cases either reduce to commutativity or are impossible by a cancellation lemma, X + Y ≠ X for finite X and Y. That lemma relies on the requirement a ≠ ±2.
- I also had to prove "X + ((−X) + Y) = Y" and −(X+Y) = (−X)+(−Y) without assuming associativity.

Proving elliptic curve properties in ACL2.

I'll set up ACL2 in the sandbox, look at the three deferred properties in `books/kestrel/crypto/ecurve`, and start on whichever looks most tractable.

Created a file, read 3 files, and 108 more steps ›

First milestone: **`(montgomery-add-closure)` is now a proved theorem** (draft book certifies in ~2 s).

How: I found `books/projects/shnf` (Russinoff's verified sparse-Horner polynomial normalizer, from his curve25519 group-law proof) and built a small bridge, `ecurve/prime-field-shnf.lisp`, so prime-field identities over `add` / `sub` / `mul` / `neg` can be proved by computation. On top of it, a macro `defthm-pfield-implication` takes hypothesis equations, a conclusion, and *certificate cofactors* (which I compute offline with sympy), and ACL2 checks $m\cdot(L-R) = \Sigma\, c_i(L_i-R_i)$ by normalization. Closure is then two certificates (doubling and chord cases) plus a case split over `montgomery-add`.

Next I'll look at the two associativity properties — the twisted Edwards one (complete curves, one uniform formula; Hales has an elementary proof of exactly this) looks more tractable than Montgomery's case-heavy one, so I'll go there next.

Read 2 files, ran 24 commands ›

Second milestone: **`(twisted-edwards-add-associativity)` is now proved too** (draft book certifies in ~3 s).

The core is two certificates (x and y coordinates) in Hales' style: the intermediate sums are variables tied to their defining equations, and the multiplier is the product of the six addition denominators (nonzero by completeness, using the existing `1±d.x1.x2.y1.y2-not-0` rules plus closure) and powers of $a - d\cdot y_i^2$ (nonzero since $a \neq d$). The cofactors run to thousands of terms, so instead of writing them out I made the macro compute them at certification time by pseudo-division (untrusted program-mode code inside `make-event`); only the SHNF check is trusted, and it takes ~0.3 s per coordinate.

Moving on to Montgomery associativity, the case-heavy one.

Created 3 files, read 4 files, and 134 more steps ›

Progress update: **Montgomery associativity is now proved too.** `(montgomery-add-associativity)` is a theorem (`montgomery-add-associativity-holds`), and the book certifies. That means all three properties you asked about are proved:

- `montgomery-add-closure`
- `twisted-edwards-add-associativity`
- `montgomery-add-associativity`

AI Success: Elliptic Curve Proofs

Created a file, read 3 files, and 108 more steps ›



...

AI Success: Bertrand's Postulate

Formalizing 100 Theorems

There is a "top 100" of mathematical theorems on the web, which is a rather arbitrary list (and most of the theorems seem rather elementary), but still is nice to look at. On the current page I keep track of which theorems from this list have been formalized. Currently the fraction that already has been formalized seems to be

100%

The page does not keep track of *all* formalizations of these theorems. It just shows formalizations in systems that have formalized a significant number of theorems, or that have formalized a theorem that none of the others have done. The systems that this page refers to are (in order of the number of theorems that have been formalized, so the more interesting systems for mathematics are near the top):

Lean	100
HOL Light	99
Isabelle	95
Rocq	80
Metamath	75
Mizar	71
nqthm/ACL2	48
ProofPower	43
PVS	26
Imandra	20
Megalodon	12
Naproche	10
NuPRL/MetaPRL	8

Math

100-theorems

doc/100-theorems.lisp :DIR :SYSTEM

ACL2 versions of (some of) the Top 100 Theorems List

This page collects those theorems from the Formalizing 100 Theorems page that have been proved in ACL2.

Note that ACL2 has largely focused on proving theorems that are rather different from these (e.g., the correctness of large and complex hardware designs).

1. The Irrationality of the Square Root of 2

Theorem: `irrational-sqrt-2`

```
(defthm irrational-sqrt-2
  (implies (equal (* x x) 2)
           (and (real/rationalp x)
                (not (rationalp x)))))
```

By Ruben Gamboa, in books/nonstd/nsa/sqrt-2.lisp.

2. Fundamental Theorem of Algebra

Theorems `fundamental-theorem-of-algebra` and `fundamental-theorem-of-algebra-sk`.

By Ruben Gamboa and John Cowles, in books/workshops/2018/gamboa-cowles/complex-polynomials.lisp.

Note: These theorems require ACL2(r).

3. The Denumerability of the Rational Numbers

AI Success: Bertrand's Postulate

- Question: Can Claude prove some of the 100 theorems?
- Claude and I discussed what to try first

#98 Bertrand's Postulate (recommended). For every $n \geq 1$ there is a prime p with $n < p \leq 2n$. It can use Russinoff's `dm::primep` from `projects/numbers/euclid.lisp`, which the page already relies on for #11, #20 and #80. So a reviewer only has to read this:

lisp

```
(defun-sk prime-in-range (n)
  (exists p (and (dm::primep p) (< n p) (<= p (* 2 n)))))

(defthm bertrand-postulate
  (implies (posp n)
           (prime-in-range n)))
```

AI Success: Bertrand's Postulate

ok, go for Bertrand's Postulate first. can you just work until you get it? can you base the proof on what was done in other provers or in textbooks?

Proving Bertrand's Postulate formally using ACL2.

I'll set up ACL2 in the sandbox, then build an Erdős-style proof (following the structure of the Lean/Isabelle/HOL Light developments and the textbook "Proofs from THE BOOK" argument) until it certifies.

Created 6 files, read 4 files, and 163 more steps ›

Bertrand's Postulate is proved and certifies in about 30 seconds. The statement:

lisp

```
(defun-sk exists-prime-between-n-and-2n (n)
  (exists (p)
    (and (dm::primep p)
         (< n p)
         (<= p (* 2 n)))))

(defthm bertrand-postulate
  (implies (posp n)
           (exists-prime-between-n-and-2n n)))
```

It's Erdős's proof, structured like Mathlib's, but done entirely in integer arithmetic with no reals or logs. Values n < 1024 are checked by evaluation. I'm now tidying up, putting it in the tree, adding the #98 entry to `100-theorems.lisp`, and preparing a patch.

98. Bertrand's Postulate

Theorem: `bertrand-postulate`

```
(defthm bertrand-postulate
  (implies (posp n)
           (exists-prime-between-n-and-2n n)))
```

where:

```
(defun-sk exists-prime-between-n-and-2n (n)
  (exists (p)
    (and (dm::primep p)
         (< n p)
         (<= p (* 2 n)))))
```

By Eric Smith and Claude, in books/kestrel/number-theory/bertrand/bertrand.lisp.

AI Success: BearSSL AES proofs

Let's see if you can work together to verify some crypto code using Axe, like the other proofs I've done. We should pick a target that compiles nicely to x86. Maybe some important crypto library in C or Rust. Can you list some ideas? I'm hoping the code will be simple and well-factored, not mixing core crypto functionality with other stuff like printing or file I/O. Ideally we'd verify multiple algorithms from the same library. It's okay if someone has already verified it. What are good candidates to try to verify? This could lead to Axe improvements and maybe even new crypto specs that I write in the process.

Claude suggested several projects, including Monocypher and BearSSL

very helpful analysis. i am already working on monocypher (great minds think alike). so, why don't you try bearssl, starting with its AES? btw, what do you mean by bitsliced in this context?

AI Success: BearSSL AES proofs

I'll start BearSSL AES now. The plan: get BearSSL 0.6 (its latest release) into the ACL2 Docker sandbox, write a small harness per AES implementation, lift each with `def-unrolled`, and prove it equal to your AES spec. I'll go from easiest to hardest: `aes_small`, then `aes_big`, then `aes_ct`, then `aes_ct64`.

What "bitsliced" means here. Normally AES code works one byte at a time, and the S-box is a table lookup: `S[x]`. A bitsliced version turns the data sideways. Bit k of every byte goes into the same machine word, so a 32-bit word holds one bit position from 32 different bytes.

In BearSSL's `aes_ct`:

- Two 16-byte blocks (32 bytes) are spread over eight 32-bit words `q[0..7]`. Word `q[k]` holds bit k of all 32 bytes. `br_aes_ct_ortho` does this transpose, and running it again undoes it.
- The S-box is never looked up. It's computed as a Boolean circuit: the Boyar–Peralta circuit, about 113 XOR/AND/XNOR gates applied to whole words. Each gate is one machine instruction, so a single run of the circuit computes 32 S-boxes at once.
- ShiftRows and MixColumns become fixed bit-moves within and between the words, done with masks, shifts and rotates.

`aes_ct64` does the same with 64-bit words, which gives four blocks per run.

The reason is constant time. Table lookups indexed by key-dependent bytes leak through the cache. Bitsliced code has no data-dependent memory accesses or branches.

What this means for Axe. In the two table versions, the S-box shows up in the lifted DAG as an 8-bit array lookup, which matches the spec fairly directly. In the bitsliced versions there's no lookup to match. Proving them equal to the spec means showing that a 113-gate circuit computes the S-box on every bit of the state, and the state is transposed throughout the computation. That's a genuinely different equivalence problem, and it's where I expect Axe to need help.

AI Success: BearSSL AES proofs

First result: **BearSSL's `aes_big` (T-table) AES-128 is verified.** That covers the key schedule plus one-block encryption, built with gcc -Os (BearSSL's default flags) for x86-64. `def-unrolled` lifts it in about 20 s to a 1,885-node DAG with no memory reads left in it. `prove-equal-with-axe` then shows it equal to your `aes-128-encrypt`, with the miter discharged in sweep 0, in about 21 minutes.

...

I asked Claude for info on some issues it found in the tools (e.g., missing rewrite rules), but I didn't help it with the proofs

...

I verified all four BearSSL AES implementations (aes_big, aes_small, aes_ct and aes_ct64) with Axe: the AES-128 key schedule plus encryption of one block. Along the way I found a bug in the x86isa model itself, plus a set of Axe issues. Nothing is pushed.

Code	Lift	Proof	How
aes_big	18 s	1,251 s	key schedule plus the internal block function
aes_small	67 s	9 s	through BearSSL's CBC API
aes_ct	586 s	106 s	CBC API; Axe rewrote the whole proof goal to true
aes_ct64	924 s	220 s	CBC API; rewritten to true; needs the MOVD fix

Let the AI plan how to use the AI

- Tell it your goal — let it suggest ways to accomplish it.

> I have a largish interesting project that I would like you to manage: implementing operational semantics specifications in ACL2 for NIST-approved post-quantum cryptographic algorithms. Please make an inventory of such algorithms, and after I review it, we can go over of the requirements and resources. I expect that we will want to recruit team member agents to work on the algorithms. Feel free to suggest an organizational structure that will result in reusable components and general additions to the ACL2 cryptographic libraries in addition to the main algorithms.

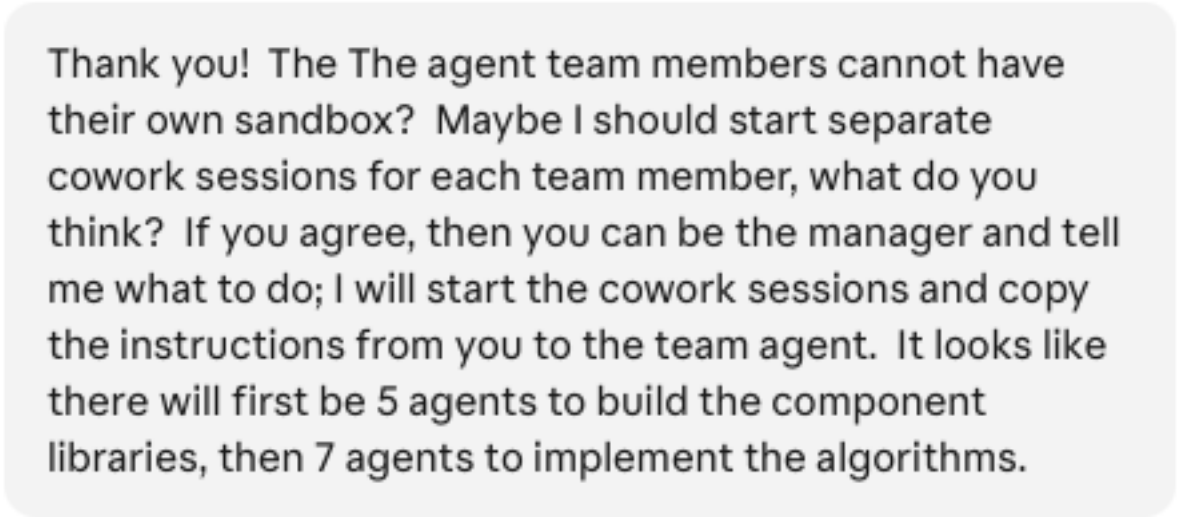
Don’t let the AI build ACL2 in its sandbox

- You can use images from github.com/KestrelInstitute/acl2-docker
- We use the nightly images on the testing-kestrel branch which include all the community books, certified, and Z3 and STP and an old Java.
- There is also acl2-allcerts on the master branch, updated less often.

> Please look at github.com/KestrelInstitute/acl2-docker/INSTALL.md and get `ghcr.io/kestrelinstitute/kestrel-allcerts-java:dffe46c-amd64` instead of acl2-allcerts. Please read the agent instructions and acl2-mcp instructions. It might be that you can use the image for fast lookup and fast xdoc lookup and you will have the option of trying things out. The agent team members who implement the post-quantum algorithms will definitely need to use the skills and mcp server so it is good if you try them out so your instructions to them can be correct.

If you need a team, build one.

- Designate one Claude Cowork session as the manager and a few as workers. Each Cowork session has its own 2 cores and 8 GB.



Both you and the AI have good ideas

- Claude came up with a messaging system for the agents to communicate:

Sessions can't see each other's files, so everything goes through one GitHub repository. Each member works on its own branch and opens pull requests, which I review and merge. The plan and decisions live in the repository too, so a fresh manager session can take over if this one runs long.

What I need from you, in order:

1. Answers to the questions at the end of the doc. I need at least the operational-semantics one (in my comment), scope, and how much proof you want; I'll propose defaults for the rest.
2. A private GitHub repository for the project, with push access for this session. You'll see the same kind of approval prompt as earlier.
3. I then build the skeleton here: conventions, interface signatures for each component, the test-vector harness and setup scripts. Then I write the three briefs, and you start a session for each and paste it in.
4. While they work, tell me when one reports done or blocked, and paste my reply back to it.

I got tired of asking the agents to read messages

Decision for you: even “check messages” could go away, but only if sessions wake themselves on a schedule. A session only acts when someone writes to it. I’d check every 30 minutes, and each team would check every 30 minutes while it’s waiting on me. Work would then keep moving while you’re away, but every check uses some of your usage even when nothing is new. Should I turn scheduled check-ins on?

Yes, please turn scheduled check-ins on for the next 6 hours. Then let the teams rest and I will look at the results tomorrow.

Since my subscription weekly reset was in 6 hours, this got a lot of work done overnight.

PQC Algorithms in ACL2 — Inventory & Organization

Nov 12, 2026 · @Eric McCarthy

Project status

As of Nov 15, all three final FIPS standards are specified in ACL2, pass every NIST ACVP test vector, and have their main correctness theorems proved. PQXDH is done, with only the KEM's decryption assumed; HQC-KEM is under way; LMS, XMSS and FN-DSA have not started. TFTP (RFC 1350), added Nov 13, is specified with its options, proved safe, and proved to terminate.

Algorithm	Status	Test vectors	Proved
ML-KEM (FIPS 203)	Done	All 240 ACVP cases pass	K-PKE decrypts within the no Decaps recovers Encaps's ke pass the key checks
ML-DSA (FIPS 204)	Done	All 1,335 ACVP cases pass	Every signature verifies, at ev
SLH-DSA (FIPS 205, with SP 800-230's sets)	Done	All 1,248 ACVP cases pass, at all 12 parameter sets	Every signature verifies, at ev
PQXDH (Signal, revision 3)	Done	RFC 7748, RFC 8032 and RFC 5869 vectors; Wycheproof's X25519, X448 and HKDF cases; Kyber-1024's 100 known-answer tests	Both parties derive the same that K-PKE decrypts; X25519 X25519 public key is its XEdD coordinate; every XEdDSA sig no assumption
HQC-KEM (expected FIPS 207; HQC version 5.1)	Under way	The codes, parameters, hashes, vectors, samplers and HQC-PKE match the reference implementation's values at every level; the KEM's 300 known-answer tests come in milestone 3	Decoding corrects every erro Reed–Muller radius
LMS and HSS (SP 800-208, RFC 8554)	Not started	None yet	Worked examples in `hbs/`: ev verifies, and a signer that follo never reuses an index
XMSS and XMSS^MT (SP 800-208, RFC 8391)	Not started	None yet	Worked example of XMSS in `h` signature verifies; the same s
TFTP (RFC 1350, with RFC 1123 §4.2; not cryptography)	Done	58 transfers recorded between five real implementations, 18 of them with options (checked by the manager); the spec matches them except where an implementation departs from the RFCs	Packets: decoding and encod inverses; netascii's conversio transfer is safe under any loss reordering; over a channel tha delivers exactly the file, one D block; duplicates can't multip

AI Success: Finding Bugs

- AI review is finding many bugs in ACL2 and the community books
  - We are fixing them as fast as we can
  - One-time effort to find decades of bugs
- All kinds of bugs/issues:
  - Stale comment
  - Theorem has the wrong name
  - Ill-formed xdoc
  - Extra free variable in rule
  - Duplicate theorem
  - Rule could be improved
  - Theory invariant names non-existent rule
  - …
  - Incorrect model of CPU instruction
  - Soundness bug in ACL2, including very tricky ones

AI Success: Finding Bugs

- Example:



`logeqv-negative` has a dangling free variable

BV-001 | CONFIRMED FIXED | Type: Defect | Severity: Medium

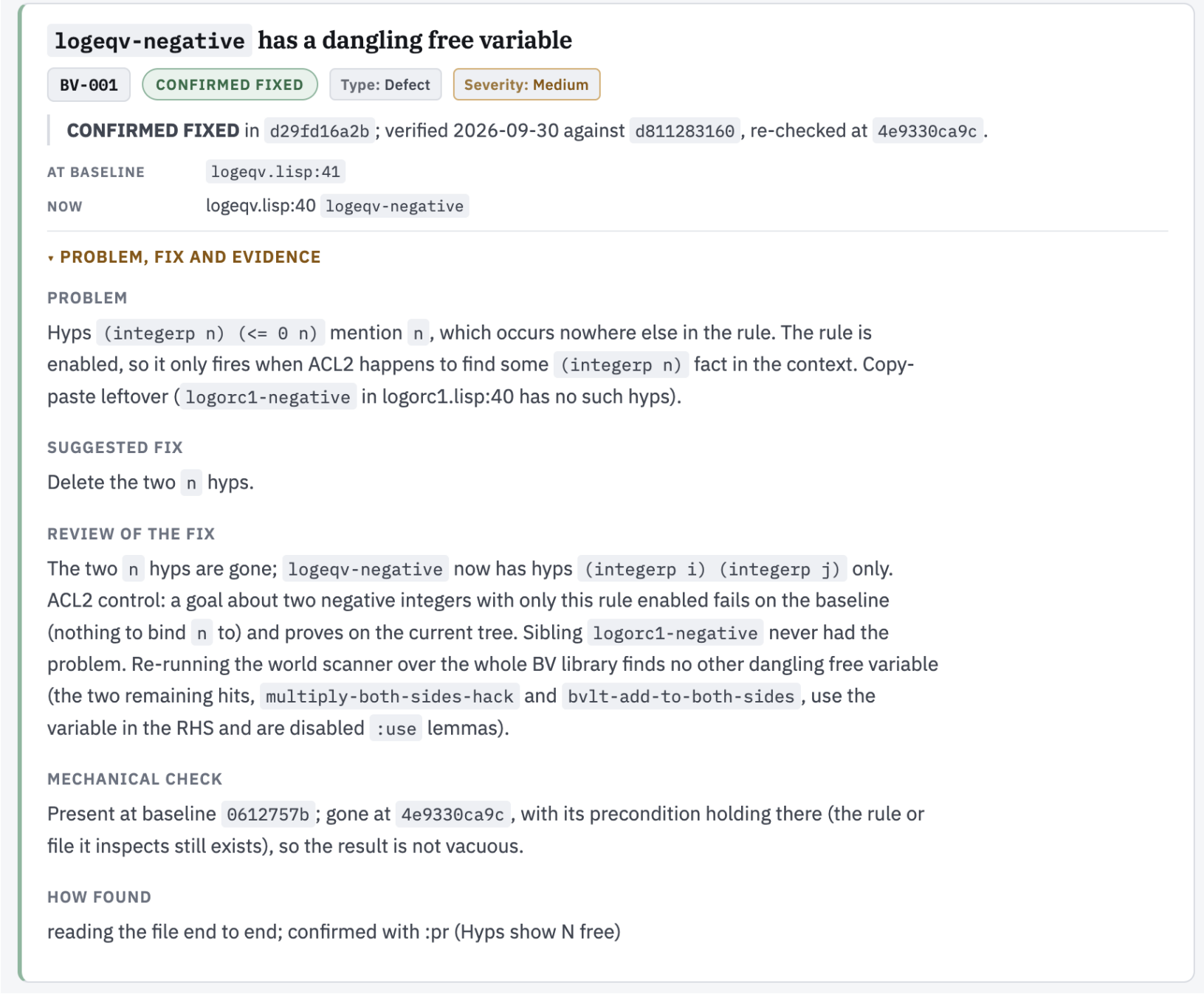
> **CONFIRMED FIXED** in `d29fd16a2b`; verified 2026-09-30 against `d811283160`, re-checked at `4e9330ca9c`.

AT BASELINE	`logeqv.lisp:41`
NOW	logeqv.lisp:40 `logeqv-negative`

▸ PROBLEM, FIX AND EVIDENCE

AI Success: Finding Bugs

- In full detail:
  - I usually don't read all this!



`logeqv-negative` has a dangling free variable

BV-001 | CONFIRMED FIXED | Type: Defect | Severity: Medium

CONFIRMED FIXED in `d29fd16a2b`; verified 2026-09-30 against `d811283160`, re-checked at `4e9330ca9c`.

AT BASELINE `logeqv.lisp:41`

NOW logeqv.lisp:40 `logeqv-negative`

▾ PROBLEM, FIX AND EVIDENCE

PROBLEM

Hyps `(integerp n) (<= 0 n)` mention `n`, which occurs nowhere else in the rule. The rule is enabled, so it only fires when ACL2 happens to find some `(integerp n)` fact in the context. Copy-paste leftover (`logorc1-negative` in logorc1.lisp:40 has no such hyps).

SUGGESTED FIX

Delete the two `n` hyps.

REVIEW OF THE FIX

The two `n` hyps are gone; `logeqv-negative` now has hyps `(integerp i) (integerp j)` only. ACL2 control: a goal about two negative integers with only this rule enabled fails on the baseline (nothing to bind `n` to) and proves on the current tree. Sibling `logorc1-negative` never had the problem. Re-running the world scanner over the whole BV library finds no other dangling free variable (the two remaining hits, `multiply-both-sides-hack` and `bvlt-add-to-both-sides`, use the variable in the RHS and are disabled `:use` lemmas).

MECHANICAL CHECK

Present at baseline `0612757b`; gone at `4e9330ca9c`, with its precondition holding there (the rule or file it inspects still exists), so the result is not vacuous.

HOW FOUND

reading the file end to end; confirmed with :pr (Hyps show N free)

Soundness bugs

- We have a long list of soundness bugs to fix.
  - Including some bugs in CCL and SBCL.
- Matt and J are working hard on the list (see recent commits).
- Once we run out of bugs, we might ask the community to find more – but not yet!
- Most bugs people sent to us have been already known.
- Maybe best not to publish lots of “ACL2 Zero-Days”

AI Workflow: Finding Bugs

- Give the AI access to your code and tools (ACL2)
  - Details later in this tutorial…
- Have it review any code you care about
  - Brace yourself for a long list of bugs!
- Have it create reports that track bugs and fixes
  - It manages the report.
  - All important stuff goes in the report, not just the chat.
  - You periodically say “Grab the latest commits and update the report.”
  - It vets all fixes and only marks an item done when it is satisfied.
  - Fixes can lead to additional bugs, which it tracks in the report.
  - You can mark an item “Won’t fix”.
- From now on, have the AI review all of your commits
  - Issues go into the report.
- Optional: Have the AI create patch files (apply them with `git apply`)
  - and tests that now pass
  - and release notes
  - and commit messages (apply them with `git am`)

Making tau cheap: three optimizations to tau.lisp

Profiled under SBCL; prover output bit-identical in every A/B, mini-proveall passes

1

fixnum

Fixnum declarations and = in the tau-pair list loops (member, subset, near-subset, insert). Plain ACL2, no raw Lisp.

10%

tau overhead on the Kestrel suite: 477 s → 51 s (897 books)

2

memoing

Raw-Lisp caches: recognizer evaluation on constants, and an EQUAL hash shadowing the completion alist (calist).

−368 s

4,492 books with tau turned off: database upkeep still runs

3

merge

Batch merge of implicant pairs in add-recogs-to-tau1: one O(k+m) pass instead of k inserts, with exact fallbacks.

15.3 → 2.1 s

tau time in (verify-guards valid-expr), C validator

Six installable versions (v0-stock … v6-merge-only) in tau-versions/, via install-version.sh

Proving tau correct in ACL2's own logic

Package TAU: 60 books, ~45,000 lines, generated from tau.lisp by gen.py; all certify

1 Extract

Logic-mode copies of all of tau.lisp except 30 non-terminating definitions; fuelled models where needed

2 Taus

add-to-tau1 sound, exact, complete; tau-union; combine-tau; batch merge; CNF/DNF

3 Queries

tau-term core, tau-assume, tau-rewrite (big switches), conjunctive rules, tau-clausep

4 Database

tau-put* and every rule adder keep the world invariants; the implicant database stays closed

Capstone: after any sequence of defuns and stored theorems, a clause that tau-clausep proves is true. Non-vacuous: the invariants have a witness world, the semantic hypotheses have a concrete model, and the real ACL2 world passes the decidable invariants.

What the proofs found: nine bugs in tau.lisp

Each has a reproduction and a fix in tau-versions/-bug-report.md; none reported upstream yet*

Unsound

tau-implies, pos-evg branch

A signature rule fires without checking an unevaluable recognizer: ACL2 proves NIL

Unsound

tighten-bound collapse

Interval collapsed to a point ignores neg-evgs and an existing pos-evg

Unsound

collect-<?-k-x

Drops every neg-evg after a leading (NIL), losing exclusions

Incomplete

tau-put* on contradiction

Stops propagating, so tau fails goals it proved before the rule

Incomplete

bounds skip predecessors

q → p, then p → x < 3: tau cannot prove q → x < 3; the other order works

Incomplete

big-switch on BITP

Opened BITP test stored with a non-variable switch; never fires

Closure

add-to-tau1 order

(not posp) then integerp loses x ≤ 0; the same order hides two contradictions

Closure

collapse to a constant

An implicant tau that collapses to 0 drops tau-put*'s predecessor index

Error

natp / posp disabled

Raw-Lisp error in ev-fncall-w-tau-recog

Fixes: all installed in the logic books, so the theorems are about fixed code. The four closure-related fixes are also in ACL2's tau.lisp, with a clean full regression.

The proof books

60 books in tau-versions/top.lisp, in dependency order; cert.pl top rebuilds them all. Bold: main results

Extracted code

tau-logic-common
tau-logic
tau-logic-v6

Taus

tau-batch-equiv
cnf-dnf-correct
tau-semantics
tau-union-sound
add-to-tau-termination
add-to-tau-sound
add-to-tau-complete
combine-tau-sound

Queries

tau-queries-sound
tau-term-core
conjunctive-rules-sound
tau-clausep-core
tau-clausep-lst-core
tau-rewrite-sound
tau-assume-basic-sound
tau-assume-sound

Store invariants

reachable-tau
tau-put-invariants
tau-pred-invariants
add-to-tau1-exact
add-tau-signature-rule-invariants
add-tau-signature-rule-world

World invariants and rule adders

tau-world-frame
tau-world-invariants
tau-put-star
tau-put-star-fixed
tau-put-star-fixed2
tau-adders-small
tau-adder-big-switch
tau-adder-boolean
tau-adder-conjunctive
add-tau-rule-dispatch
tau-normalize-sound
add-tau-rule-of-theorem
tau-termp-normalize
tau-remove-guard-holders
tau-rule-sequence
tau-visit-defuns
tau-invariants-witness
tau-semantics-consistent

Closure

tau-closure-abstract
tau-closure-nodeg
tau-closure-seq
tau-closure-refine
tau-closure-concrete
tau-closure-original
tau-closure-concrete-seq
tau-closure-concrete-witness
tau-closure-fuel
tau-closure-deg
tau-closure-concrete-deg
tau-closure-algebra
tau-add-order-fixed
tau-closure-algebra-fx
tau-put-collapse-fixed
tau-closure-algebra-fx2
tau-closure-algebra-fx3

Possible generalizations of tau

Ideas, not work done; the closure axioms say what any new domain must satisfy

1 Pluggable abstract domains

Intervals are one domain. Congruences (evenp), bit widths or finite constant sets could plug in; closure carries over for any domain meeting the ten axioms.

2 Relations between variables

Tau is monadic: facts about one term. Literals such as x < y or (len x) = (len y) would need difference bounds or equality classes.

3 Parameterized recognizers

One (list-of p) instead of a recognizer per element type, with signature rules that carry the parameter through car, nth and append.

4 Limited disjunction

A tau is a conjunction. Small unions, such as natp-or-nil, would avoid case splits in signature rules over maybe types.

5 Other equivalences

Tau reasons about EQUAL. Recognizers respecting perm, set-equiv or a fixing function would let it work under congruences.

6 Verified clause processor

The capstone is the statement a clause processor needs: run the proved tau, or a tau over user domains, via :clause-processor.

AI Success: ACL2 Live mode for Emacs

- Claude created a new Emacs mode for ACL2.
- Live, automatic processing of files.
  - Start checking events.
  - Successful events become green.
  - Try to continue after failures.
    - Red = failure
    - Purple = success (after a failure)
- Claude wrote all the code.
- Many rounds of me trying it and requesting changes.

- Reach out if you want to try it.

```
(defthm integerp-of-mod
  (implies (integerp y)
           (equal (integerp (mod x y))
                  (integerp (fix x))))
  :hints (("Goal" ;:cases (integerp x)
           :in-theory (enable mod))))

(defthm integerp-of-mod-type
  (implies (and (integerp x)
                (integerp y))
           (integerp (mod x y)))
  :rule-classes :type-prescription
  :hints (("Goal" :in-theory (enable mod))))

(defthm nonneg-of-mod-type
  (implies (and (<= 0 x)
                ;; (rationalp x)
                (<= 0 y)
                (rationalp y))
           (<= 0 (mod x y)))
  :rule-classes :type-prescription
  :hints (("Goal" :cases ((equal 0 y))
           :in-theory (enable mod *-of-floor-upper-bound))))

(defthm nonneg-of-mod-type-2
  (implies (and ;(<= 0 x)
                (rationalp x)
                (< 0 y)
                (rationalp y))
           (<= 0 (mod x y)))
  :rule-classes :type-prescription
  :hints (("Goal" :cases ((equal 0 y))
                  :in-theory (enable mod *-of-floor-upper-bound))))
```

Many ways to use agents

GPT:

- Codex CLI
- Codex desktop app
- Codex IDE extensions
- chatgpt.com Chat
- chatgpt.com Work
- Codex Cloud
- Various API key offerings
- Others: Slack; Teams; etc.

Claude:

- Claude Code CLI
- Claude desktop app
- Claude IDE extensions
- claude.ai Chat
- claude.ai Cowork
- claude.ai/code
- Various API key offerings
- Others: Slack; MS365; etc.

These give you cheap cloud machines

GPT:

- Codex CLI
- Codex desktop app
- Codex IDE extensions
- chatgpt.com Chat
- chatgpt.com Work
- Codex Cloud
- Various API key offerings
- Others: Slack; Teams; etc.

Claude:

- Claude Code CLI
- Claude desktop app
- Claude IDE extensions
- claude.ai Chat
- claude.ai Cowork
- claude.ai/code
- Various API key offerings
- Others: Slack; MS365; etc.

Advantages: security; simplicity; compute

These are local — be careful with security!

GPT:

- Codex CLI
- Codex desktop app
- Codex IDE extensions
- chatgpt.com Chat
- chatgpt.com Work
- Codex Cloud
- Various API key offerings
- Others: Slack; Teams; etc.

Claude:

- Claude Code CLI
- Claude desktop app
- Claude IDE extensions
- claude.ai Chat
- claude.ai Cowork
- claude.ai/code
- Various API key offerings
- Others: Slack; MS365; etc.

some advantages on next slide

Why bother with local apps for agents?

- Build up shared memory files, what the agent learns from you about how to do ACL2 proofs and other things.
- Using MCP lets you get ACL2 session log
- Using MCP lets you control multiple ACL2 sessions on a big local machine (cloud machines usually have only 2, 4, or 9 CPUs)
- `/export`

Using Docker

- ACL2 Docker images of various sorts available. Quick start:

```
docker pull ghcr.io/kestrelinstitute/kestrel-allcerts-java:latest
```

- This particular image is built nightly from the testing-kestrel branch, and it has **all community books certified** in the image. Also has Z3, STP, and an old Java.
- This image also has **fast, agent-friendly xdoc**.
- For details see `INSTALL.md` under https://github.com/KestrelInstitute/acl2-docker/
- You can use the Docker image in place of an ACL2 executable, locally with or without AI, or in a cloud AI session.

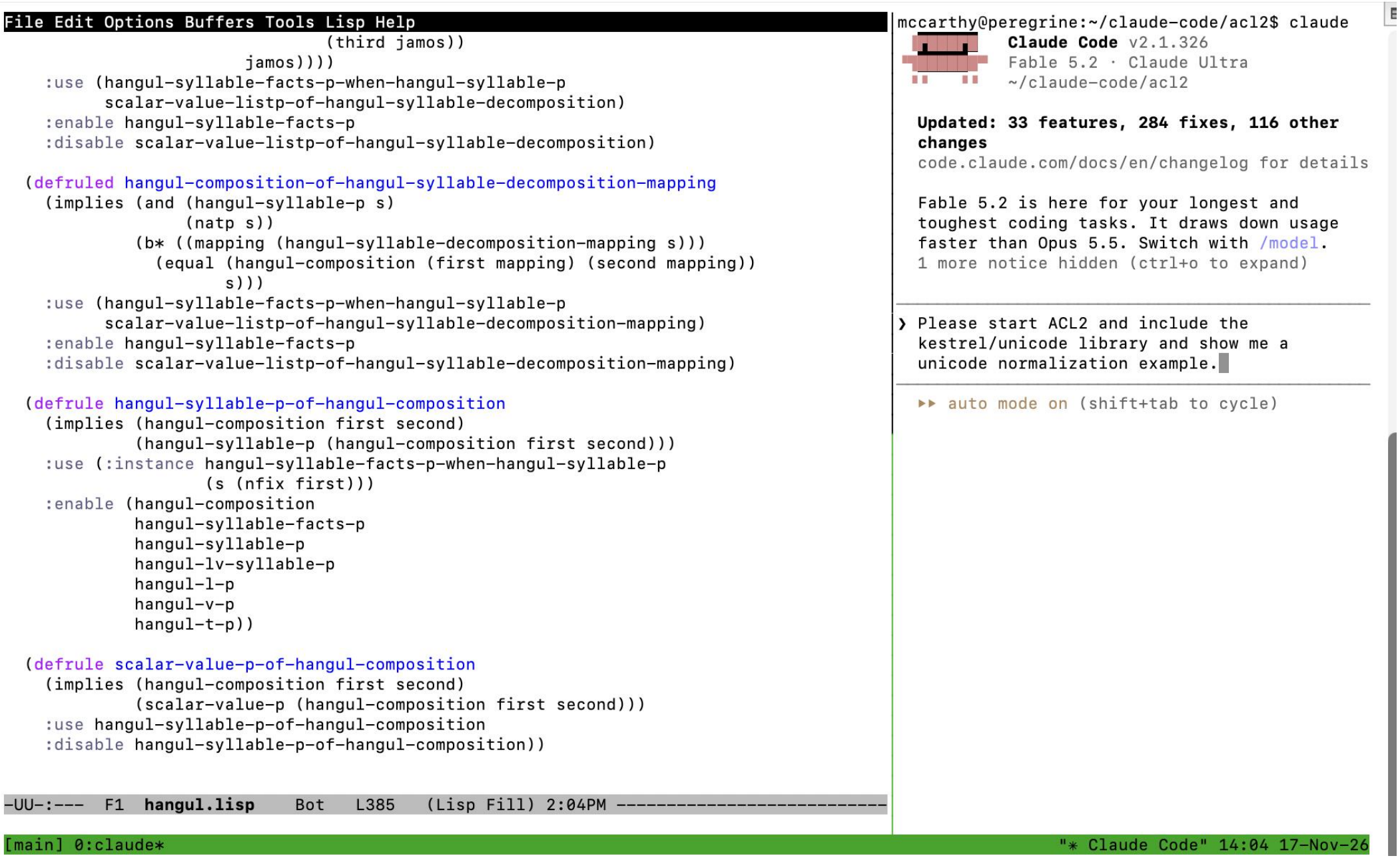
Using MCP

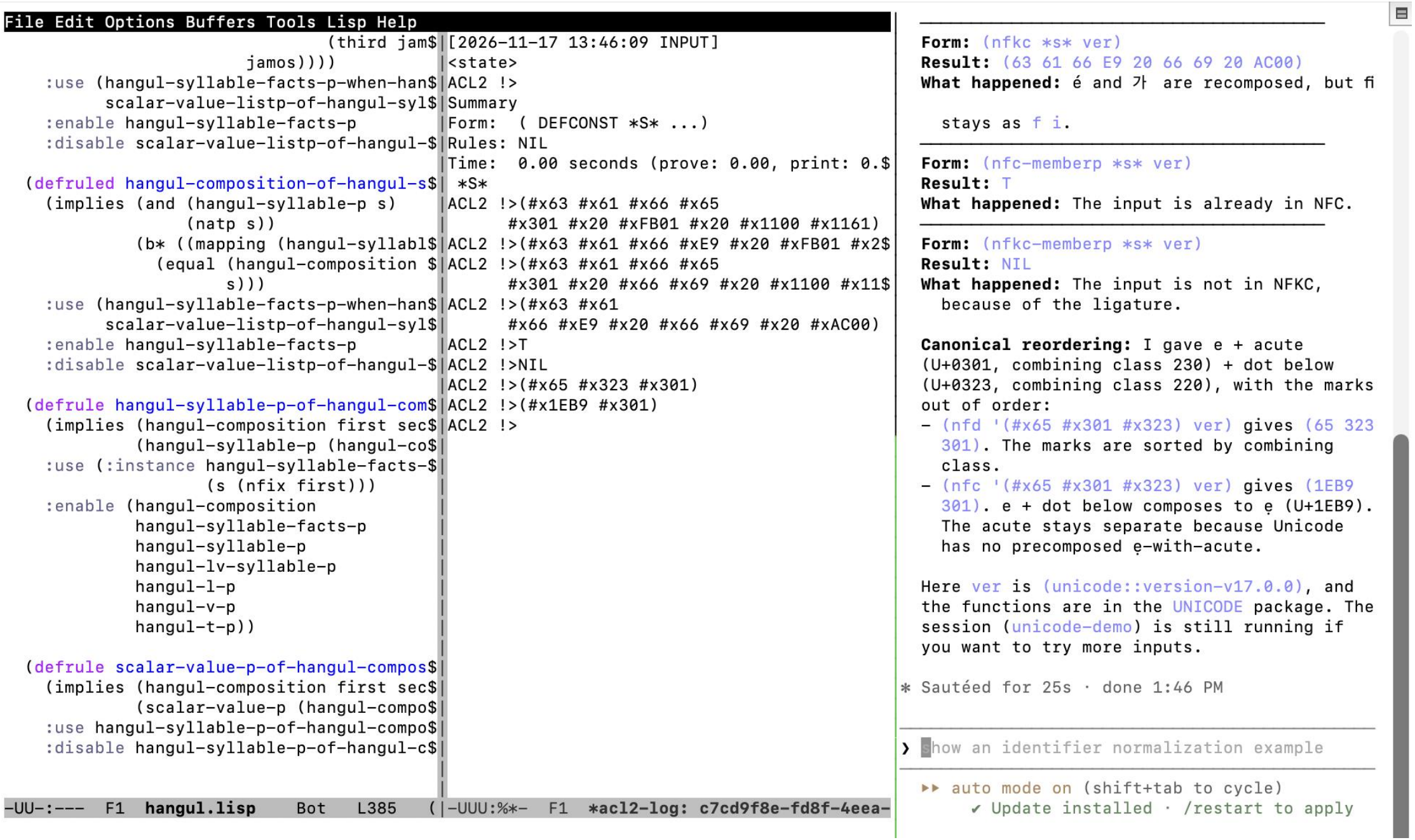
- MCP = Model Context Protocol
- acl2-mcp is at https://github.com/bendyarm/acl2-mcp

  There is documentation there.
- It enables an AI agent to control ACL2 via a persistent python program that manages the I/O. Prevents the agent's context from being overloaded with proof output, by eliding responses.
- It saves a session log that the agent can grep for debugging details. If running locally, it can display the ACL2 session to a window so you can see what it is doing.

Using acl2-mcp + emacs + tmux + ssh

- See `INSTALL.remote-tmux-emacs.md` in https://github.com/bendyarm/acl2-mcp
- Suggested layout: tmux window split into two panes: the left is 2/3 of the total and used for Emacs; the right is 1/3 and used for a shell with Claude Code CLI.
- When acl2-mcp starts, the Emacs window is split into two side-by-side panes, the right of which displays the ACL2 session log.
- The next slides show an example.
- This can be useful if you have a powerful server without a desktop and you don’t mind forgoing use of a mouse.


File Edit Options Buffers Tools Lisp Help
(third jamos))
jamos))))
:use (hangul-syllable-facts-p-when-hangul-syllable-p
scalar-value-listp-of-hangul-syllable-decomposition)
:enable hangul-syllable-facts-p
:disable scalar-value-listp-of-hangul-syllable-decomposition)
(defruled hangul-composition-of-hangul-syllable-decomposition-mapping
(implies (and (hangul-syllable-p s)
(natp s))
(b* ((mapping (hangul-syllable-decomposition-mapping s)))
(equal (hangul-composition (first mapping) (second mapping))
s)))
:use (hangul-syllable-facts-p-when-hangul-syllable-p
scalar-value-listp-of-hangul-syllable-decomposition-mapping)
:enable hangul-syllable-facts-p
:disable scalar-value-listp-of-hangul-syllable-decomposition-mapping)
(defrule hangul-syllable-p-of-hangul-composition
(implies (hangul-composition first second)
(hangul-syllable-p (hangul-composition first second)))
:use (:instance hangul-syllable-facts-p-when-hangul-syllable-p
(s (nfix first)))
:enable (hangul-composition
hangul-syllable-facts-p
hangul-syllable-p
hangul-lv-syllable-p
hangul-l-p
hangul-v-p
hangul-t-p))
(defrule scalar-value-p-of-hangul-composition
(implies (hangul-composition first second)
(scalar-value-p (hangul-composition first second)))
:use hangul-syllable-p-of-hangul-composition
:disable hangul-syllable-p-of-hangul-composition))
-UU-:--- F1 hangul.lisp Bot L385 (Lisp Fill) 2:04PM
mccarthy@peregrine:~/claude-code/acl2$ claude
Claude Code v2.1.326
Fable 5.2 · Claude Ultra
~/claude-code/acl2
Updated: 33 features, 284 fixes, 116 other changes
code.claude.com/docs/en/changelog for details
Fable 5.2 is here for your longest and toughest coding tasks. It draws down usage faster than Opus 5.5. Switch with /model.
1 more notice hidden (ctrl+o to expand)
❯ Please start ACL2 and include the kestrel/unicode library and show me a unicode normalization example.
▸▸ auto mode on (shift+tab to cycle)
[main] 0:claude*
"* Claude Code" 14:04 17-Nov-26


File Edit Options Buffers Tools Lisp Help
(third jam$
jamos))))
:use (hangul-syllable-facts-p-when-han$
scalar-value-listp-of-hangul-syl$
:enable hangul-syllable-facts-p
:disable scalar-value-listp-of-hangul-$
(defruled hangul-composition-of-hangul-s$
(implies (and (hangul-syllable-p s)
(natp s))
(b* ((mapping (hangul-syllabl$
(equal (hangul-composition $
s)))
:use (hangul-syllable-facts-p-when-han$
scalar-value-listp-of-hangul-syl$
:enable hangul-syllable-facts-p
:disable scalar-value-listp-of-hangul-$
(defrule hangul-syllable-p-of-hangul-com$
(implies (hangul-composition first sec$
(hangul-syllable-p (hangul-co$
:use (:instance hangul-syllable-facts-$
(s (nfix first)))
:enable (hangul-composition
hangul-syllable-facts-p
hangul-syllable-p
hangul-lv-syllable-p
hangul-l-p
hangul-v-p
hangul-t-p))
(defrule scalar-value-p-of-hangul-compos$
(implies (hangul-composition first sec$
(scalar-value-p (hangul-compo$
:use hangul-syllable-p-of-hangul-compo$
:disable hangul-syllable-p-of-hangul-c$
-UU-:--- F1 hangul.lisp Bot L385 (
[2026-11-17 13:46:09 INPUT]
<state>
ACL2 !>
Summary
Form: (DEFCONST *S* ...)
Rules: NIL
Time: 0.00 seconds (prove: 0.00, print: 0.$
S
ACL2 !>(#x63 #x61 #x66 #x65
#x301 #x20 #xFB01 #x20 #x1100 #x1161)
ACL2 !>(#x63 #x61 #x66 #xE9 #x20 #xFB01 #x2$
ACL2 !>(#x63 #x61 #x66 #x65
#x301 #x20 #x66 #x69 #x20 #x1100 #x11$
ACL2 !>(#x63 #x61
#x66 #xE9 #x20 #x66 #x69 #x20 #xAC00)
ACL2 !>T
ACL2 !>NIL
ACL2 !>(#x65 #x323 #x301)
ACL2 !>(#x1EB9 #x301)
ACL2 !>
-UUU:%*- F1 *acl2-log: c7cd9f8e-fd8f-4eea-
Form: (nfkc *s* ver)
Result: (63 61 66 E9 20 66 69 20 AC00)
What happened: é and 가 are recomposed, but fi
stays as f i.
Form: (nfc-memberp *s* ver)
Result: T
What happened: The input is already in NFC.
Form: (nfkc-memberp *s* ver)
Result: NIL
What happened: The input is not in NFKC,
because of the ligature.
Canonical reordering: I gave e + acute
(U+0301, combining class 230) + dot below
(U+0323, combining class 220), with the marks
out of order:
- (nfd '(#x65 #x301 #x323) ver) gives (65 323
301). The marks are sorted by combining
class.
- (nfc '(#x65 #x301 #x323) ver) gives (1EB9
301). e + dot below composes to ẹ (U+1EB9).
The acute stays separate because Unicode
has no precomposed ẹ-with-acute.
Here ver is (unicode::version-v17.0.0), and
the functions are in the UNICODE package. The
session (unicode-demo) is still running if
you want to try more inputs.
* Sautéed for 25s · done 1:46 PM
› show an identifier normalization example
▸▸ auto mode on (shift+tab to cycle)
✔ Update installed · /restart to apply

Prompts; skills; memories

- For each user prompt, the AI starts with a trained base model but no context.
- Then it does “prompt assembly to create a prompt payload” or “context construction to create a context window” (same thing) with this information:

The same in all conversations

- system prompt
- project instructions (AGENTS.md or CLAUDE.md)
- tool declarations and skills (AGENTS.md or skills/<skill-name>/SKILL.md)
- memories

Varies between conversations

- retrieved context (RAG)
- conversation history
- current user prompt

Prompts; skills; memories

- When the context gets too large, the harness does “compaction” — it tries to keep the earlier context in a compacted form. After a few compactions, the AI can forget that it ever did certain things.
- Because of these limitations, it is hard to do continual learning. Learning (improved performance on the tasks you are interested in) is best done by compacting information into project instructions and skill files and memories.

Project instructions and skills

Claude Code CLI by default uses

- `~/.claude/CLAUDE.md` for project instructions and
- `~/.claude/skills/<skill-name>/SKILL.md` for skills

There are other places as well.

Other models/harnesses have generally standardized on `AGENTS.md`, in the project root directory, for both project instructions and skills.

There are some simple example skills in the `for-agents/` directory in https://github.com/bendyarm/acl2-mcp

See also (at the top of that repo) `AGENTS.md` and `CLAUDE.md`

Memories

- Both Claude Code CLI and Codex CLI have ways to extract and save things that look useful for later, which are written to memory files in a compact format.
- For example, on Claude Code CLI, it writes files in `~/.claude/projects/<name-mangled-start-dir>/memory/`
- When moving to another platform, you can go through them, do some spring cleaning, and copy them to wherever the other platform stores memories.

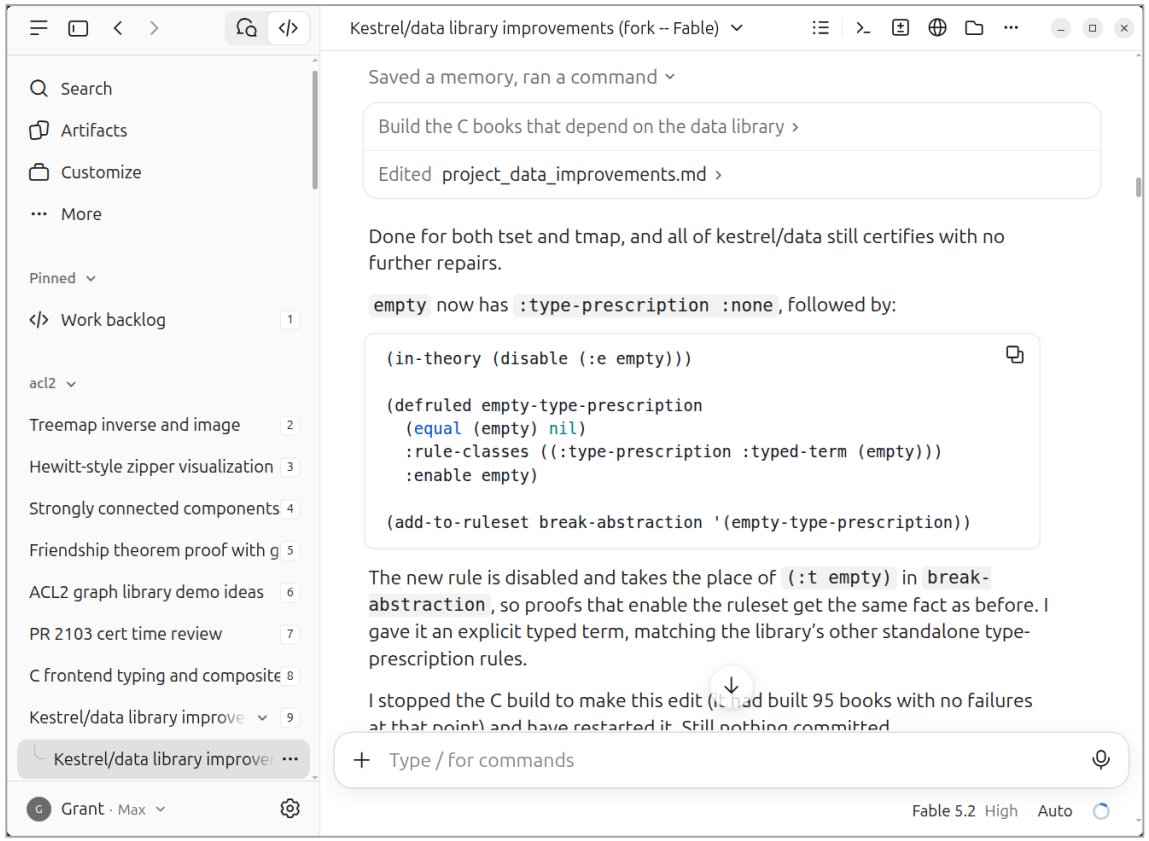
Claude “Cowork” (now integrated w/ chat)

- Runs on an Anthropic VM in the cloud.
  - No need to install anything on your machine.
  - No need to approve code it wants to run on your machine.
- Can install and run ACL2 and other tools
  - Can’t do a full regression (few cores, not much memory)
- Can run sub-agents
- Creates reports for you to view
- Creates artifacts for you to download (patch files, tarballs)
  - or connect your repos
- Tracks many (named) chats
  - pinned chats, recent chats



The Desktop App: Same Claude Code, Different Front End

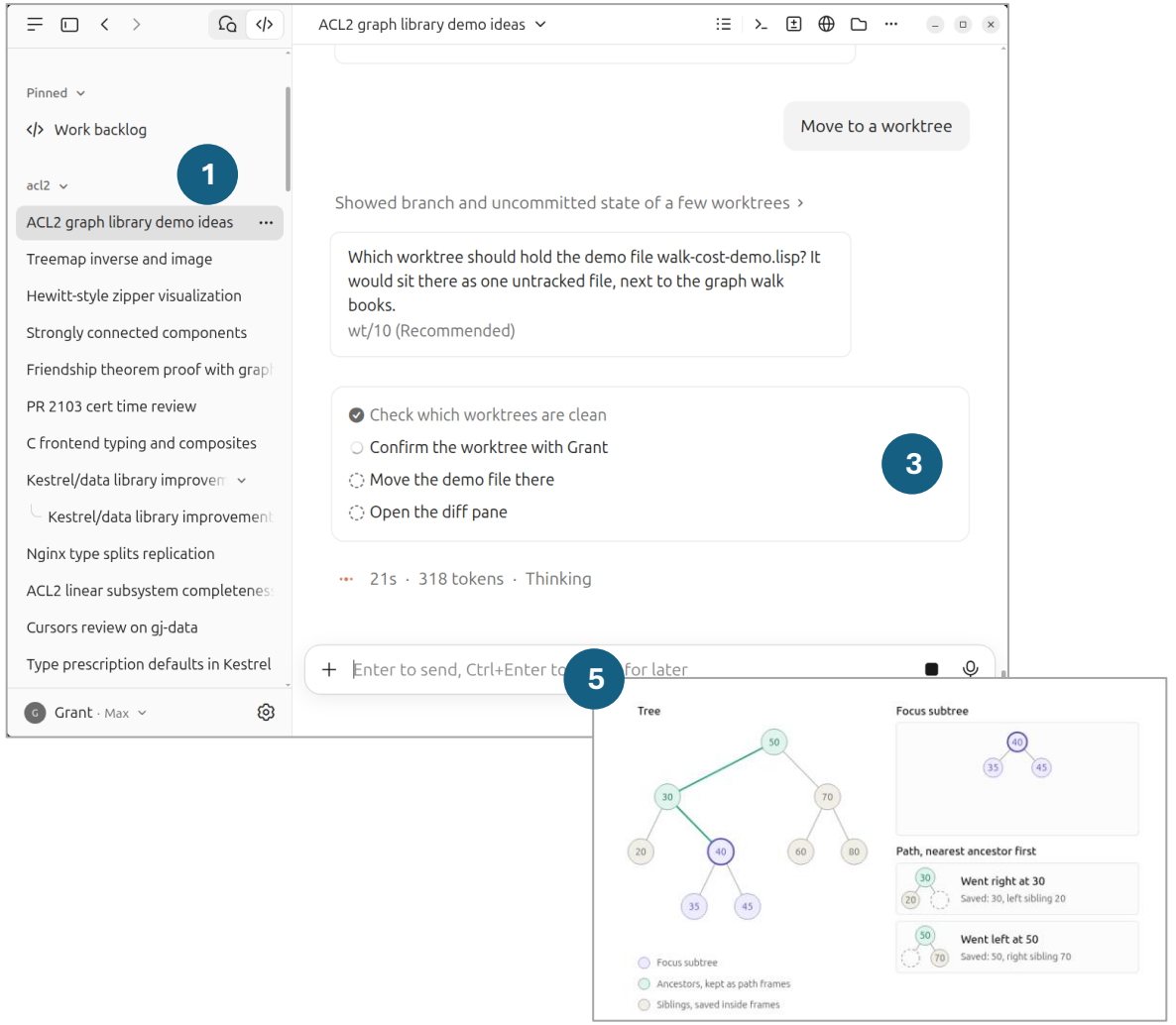
- The Claude desktop app has a tab for ordinary chat and one for Code.
- The Code tab is Claude Code: the same engine as in the terminal, reading, editing and running things in your repository in the same way.
- It uses the same configuration: instruction files, skills, MCP servers.
  - The same permission rules apply.
- What changes is the interface around the conversation.



What the GUI Adds

1. **Sessions in a sidebar:** several at once, each with a title and status; two can sit side by side.
2. **Reviewing changes:** a diff of everything the session changed; file references open in a pane.
3. **Following progress:** a task list of what it plans to do and where it is.
4. **Attachments:** paste a screenshot, a PDF or any other file.
5. **Inline visuals:** it can draw diagrams and charts in its reply.

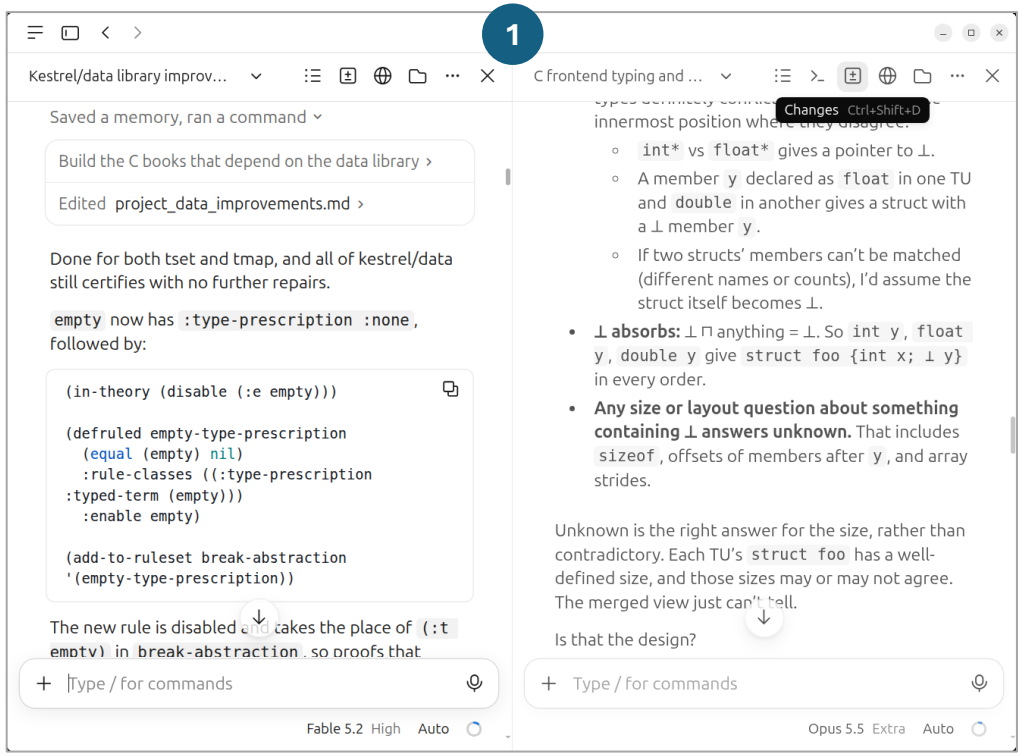
Still CLI only: scripting and other non-interactive use.



What the GUI Adds

1. **Sessions in a sidebar:** several at once, each with a title and status; two can sit side by side.
2. **Reviewing changes:** a diff of everything the session changed; file references open in a pane.
3. **Following progress:** a task list of what it plans to do and where it is.
4. **Attachments:** paste a screenshot, a PDF or any other file.
5. **Inline visuals:** it can draw diagrams and charts in its reply.

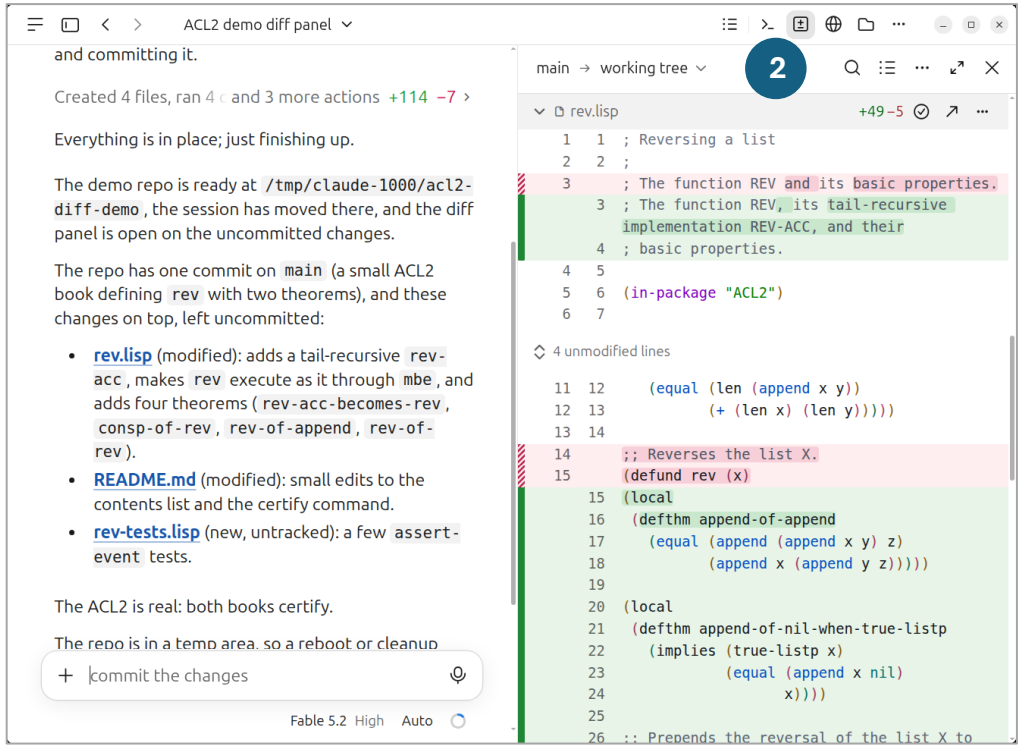
Still CLI only: scripting and other non-interactive use.



What the GUI Adds

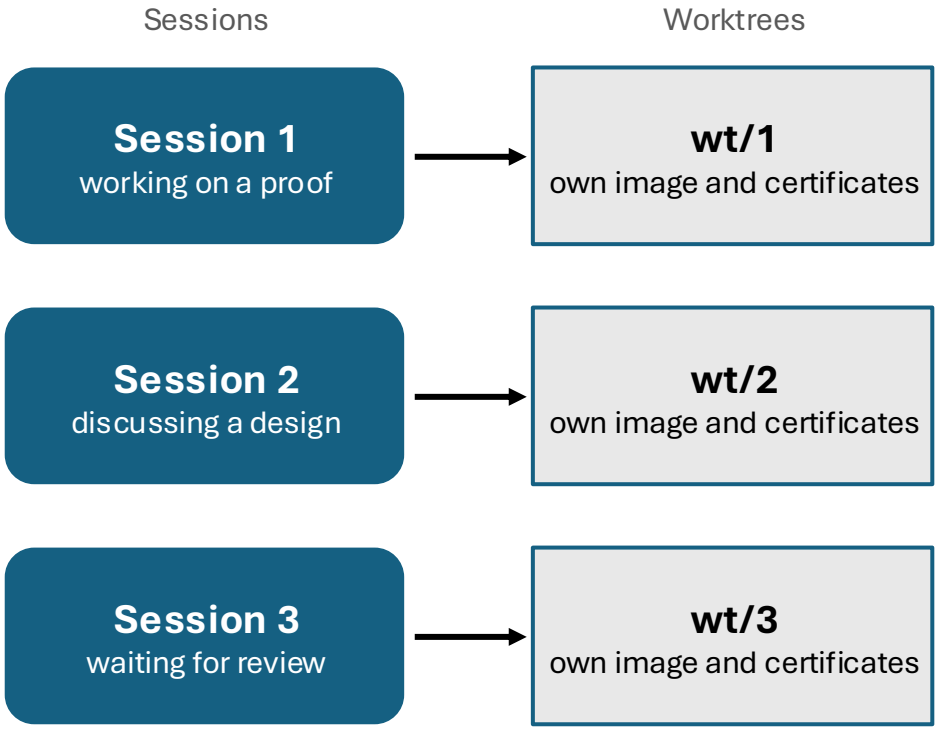
1. **Sessions in a sidebar:** several at once, each with a title and status; two can sit side by side.
2. **Reviewing changes:** a diff of everything the session changed; file references open in a pane.
3. **Following progress:** a task list of what it plans to do and where it is.
4. **Attachments:** paste a screenshot, a PDF or any other file.
5. **Inline visuals:** it can draw diagrams and charts in its reply.

Still CLI only: scripting and other non-interactive use.



Using It for ACL2 Work

- **One session per worktree:** each worktree has its own ACL2 image and certificates, so parallel sessions don't interfere.
  - A fixed pool of worktrees, reused: a new one costs an image build and a recertification.
- **Waiting doesn't block you:** while one session is thinking or working on proofs, you discuss a design in another.
- **You review before it counts:** read the diff before anything is committed.



Instruction Files: What They Are

- Plain Markdown files of standing instructions, written by you in ordinary English.
- Loaded automatically at the start of every session, before your first message.
- Two levels:
  - Global, for how you like to work in any project.
  - Project, for the facts and rules of one codebase; it can be checked in and shared.
- Claude Code reads `CLAUDE.md` and Codex reads `AGENTS.md`. Claude Code can also read `AGENTS.md`, so one project file can serve both.
- They are guidance, not enforcement: hard limits belong in permissions.

```
~/.claude/CLAUDE.md     global
<repo>/AGENTS.md        project
```

In the Project File: AGENTS.md

Facts it can’t see from the code

```
Each worktree has its own image (`saved_acl2`) and its own certificates. Never
certify books or open a session with another worktree's image …
```

```
Branch new work off `testing-kestrel`, not `master` or `origin/master`: it is
the integration branch that Kestrel work is tested against.
```

Conventions

```
Commit subjects are `[area] Subject`: the ACL2 library or package name in
brackets (`[C$]`, `[C2C]`, `[data]`, `[graph]`) …
```

In Your Global File: CLAUDE.md

How you want to work

```
A question, or a design point we haven't settled, is not a go-ahead. Answer or
discuss it, and don't edit files or start implementing until I say to.
```

```
When you have several things for me to respond to …, don't ask for all the
decisions at once; I end up skimming or answering only some.
```

Correcting its habits

```
Write comments and documentation for a later reader of the code, not for this
conversation: say what the code does or means, briefly.
```

```
Before committing, or when I ask you to finalize changes, look for existing code
related to what you added:
- Code that already does the job: use it and drop the new version.
```

Skills: What They Are

- A skill is a folder with a `SKILL.md` file: instructions for one kind of task.
  - It can also carry scripts and reference material.
- At the start of a session, the AI sees only each skill's name and description.
- When a task matches the description, it loads the full text. You can also invoke a skill by name.
- So a skill can be long and detailed, at no cost to sessions that don't need it.

```
~/.claude/skills/acl2/SKILL.md
name: acl2
description: "Conventions for working on ACL2 code: … Load it before reading, writing,
reviewing or running ACL2 code … even when the user has not mentioned ACL2. …"
```

In the acl2 Skill (1 of 2)

Rules and theory

```
Keep functions disabled. Introduce a new function with `define` or `defund`, not
`defun`, unless it is a simple abbreviation. …
```

```
State each rule as generally as it holds. Drop any hypothesis the rule does not
need …
```

Rules and theory: a personal preference

```
Before proving a lemma about a library's functions, look for it in the
libraries. Prefer the `kestrel/*-light` books to their `std` equivalents when
they have the lemma.
```

In the acl2 Skill (2 of 2)

Hints

```
Never key a hint to a subgoal name such as `"Subgoal *1/3"`: the names shift
whenever the proof changes shape, and the hint silently stops applying. Factor
out, or generalize, a separate lemma instead.
```

```
Treat a recurring hint as a sign of a missing rule: a repeated `:use` may be a
rewrite rule worth proving, and a repeated `:expand` may call for an opener
lemma.
```

Book organization and style

```
When editing an existing book, match its conventions: `define` or `defun`,
`defrule` or `defthm`, `b*` or `let` and `mv-let`, line width and formatting.
```

Memory: What the AI Writes Down

- Notes the AI writes for itself as you work: what you told it, corrections, the state of ongoing work.
- Scope: one project. Every session in the project shares it, and it stays on your machine.
- An index is loaded at the start of each session, and individual notes are read when relevant.
- It is plain Markdown, so you can read it, edit it and delete from it.

```
name: treeset-treemap-mirror
type: feedback

Grant (2026-11-08): treeset and treemap are closely related libraries. For each rule change or addition in one, briefly consider whether the same change makes sense in the other, and say so explicitly.

**Why:** keeps the two libraries' rule sets parallel; gaps found in one … usually exist in the other.
```

What Goes Where

	Instruction file	Skill	Memory
Written by	you	you	the AI
Scope	you, or one project	you, or one project	one project
Loaded	every session	when its description matches the task	index every session, notes when relevant
Holds	facts and rules that always apply	detailed guidance for one kind of work	specifics of the project and its ongoing work

Instruction files and skills grow over time: go back to them periodically to address persistent issues.

Security Configurations and Practices

- To work and interact closely with the AI, the AI needs to access (part of) the user's machine (e.g. edit the files in the git repo).
- Both Claude Code and Codex provide configurable mechanisms to limit the AI's access to the user's machine.
  - These mechanisms are (presumably) outside the AI proper, i.e. they are traditional access control algorithms that gate the AI's actions.
  - The defaults might be okay, but it is better not to trust that.
  - These mechanisms are well documented, but somewhat complex.
- In addition, one should follow certain practices.
- One can use the AI to help with this too, but should carefully check everything against the official documentation.

Claude Code’s Working Directory

- It’s the directory where Claude Code is started.
- The AI has read access to it.
- Two main kinds of write access:
  - Manual: ask the user for every edit — too tedious.
  - Auto-accept edits: edit without asking — use with git.
- Suggestions:
  - Use git, so edits are independently tracked.
  - Although the AI shows the diffs of the edits, run ‘git diff’ in the OS shell (or Emacs) to double-check.
  - Prohibit the AI from doing ‘git commit’ (discussed later).

Claude Code’s Permissions

- Permissions control what the AI itself can do directly (vs. indirectly by running separate processes, which is covered later).
- Three kinds: deny > ask > allow — in that order of evaluation.
- Permissions apply to file actions, shell command invocations, web fetching, etc., and can use wildcards.
- Permissions are configured in ~/.claude/settings.json.
- Suggestions:
  - Deny access to files that don’t need access.
  - Deny certain shell command invocations.

Claude Code's Permissions — Examples

```
"permissions": {
    "deny": [
        "Edit(//usr/**)",
        "Read(~/.ssh/**)",
        "Read(~/Pictures/**)",
        "Bash(git commit *)",
        "Bash(git push *)",
        "Bash(npm *)",
        ...
    ],
    "allow": [
        "Bash(git log *)",
        "Bash(git diff *)",
        "Bash(git show *)",
        ...
    ],
    ...
}
```

Here 'Bash' means any OS shell, not just the actual `bash` shell.

Claude Code’s Bash Sandbox

- (Here ‘Bash’ means any OS shell, not just the actual `bash` shell.)
- This mechanism covers what the AI may do indirectly, by running separate processes.
- It uses underlying OS sandboxing mechanisms, such as Seatbelt on macOS.
- Allow or deny access to files and network hosts.
- Default read access to most files.
- The bash sandbox is configured in ~/.claude/settings.json.
- Suggestions:
  - Deny access to home and similar directories.
  - Allow access to working directories.
  - (Unlike permissions, ‘deny’ does not have priority over ‘allow’.)

Codex’s Permissions and Sandboxing

- Despite the word similarity, these differ from Claude Code’s:
  - Codex’s sandboxing is an older mechanism, which may go away.
  - Codex’s permissions is the newer mechanism to use, which combines features like Claude Code’s permissions and bash sandboxing.
  - Codex’s premissions use OS sandboxing, e.g. Seatbelt in macOS.
- Codex’s permissions are configured in ~/.codex/config.toml.
- One can configure multiple permission profiles, and set a default.
- A permission profile is typically defined by extending one of the predefined profiles.

Codex’s Permissions — Structure & Examples

```
default_permissions = "<name>"

[permissions.<name>]
extends = "<predefined-profile>"

[permissions.<name>.filesystem]
":root" = "deny" # deny entire filesystem by default
":minimal" = "read" # minimal runtime paths for common tools
".../lib/sbcl/" = "read” # needed to run SBCL
... # enable more as needed

[permissions.<name>.filesystem.":workspace_roots"]
"." = "write" # working directory
... # additional workspaces, if any

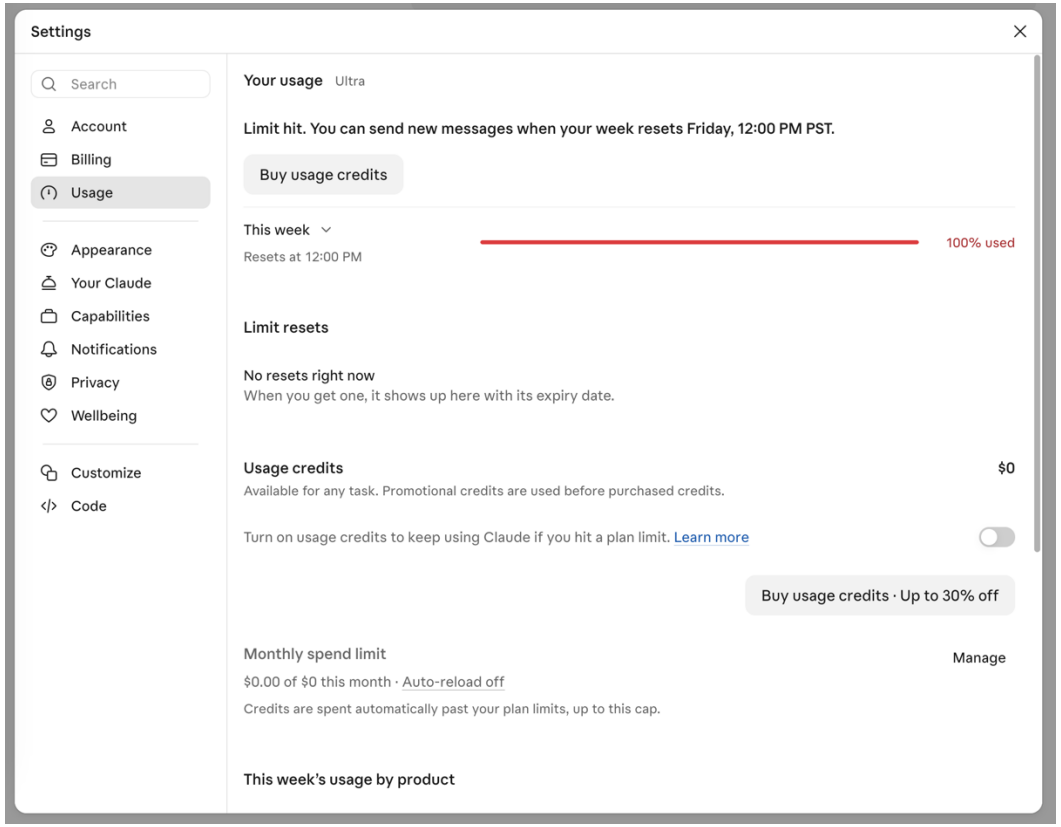
[permissions.<name>.network]
enabled = false
```

Privacy Considerations

- The AI sees what it has access to: interactions, files, etc.
- This should not be an issue for open-source code, publishable research, etc. (except perhaps for timing of disclosure).
- If this is a concern, use a local model, but it will not be as powerful as a cloud model.
- With a cloud model, there is a setting to use your data to train the AI, which you can turn off.
- When the AI asks “How am I doing?”, if you respond it may also share some data; but you do not have to respond.

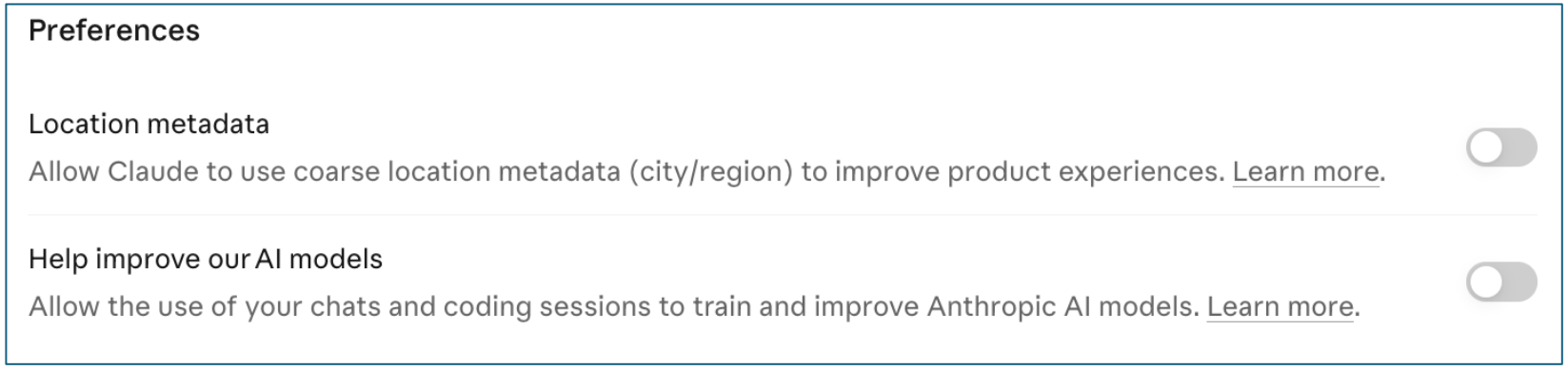
Plans and Usage

- Anthropic: $20, $100, or $200 per month
- OpenAI: $8?, $20, $100, $200, $500 per month
- Plans seem to be heavily subsidized
  - compared to API rates
- Bigger models burn through it faster
  - Fable costs more than Opus
  - and may have a separate limit
- Plans can include API credits
- Plans can include resets!
  - These expire
- Claude plans have a weekly limit
  - e.g., reset on Friday at noon
  - Thursday night it's time to get creative!
- Some plans have 5-hour usage windows
  - Annoying but can save you from burning the whole week



Cyber Protections

- Anthropic:
  - Get at least Defense access, for Mythos
    - Easy: Show ID, show face on video
    - Still blocks “exploits”
  - Try: Emphasize that it’s **your** system (when it is)
  - Try: Have it try to prove correctness – failures reveal bugs
- OpenAI:
  - Daybreak Blue?
- **Also remember to turn off model training on your data.**



Suggested Tasks for the AI

- Code review
- Any proofs you always wanted, including ambitious ones
  - search for “TODO: Prove …”
  - verify parts of the ACL2 system code
  - verify parts of Axe
  - apply our tools to verify critical software
- Cross-checks models/specs
  - Same prover, different provers (via untrusted translations or embeddings)
- Write tests
- Port proofs from other provers?
- Suggest RHS for a rule
- Improve proofs (remove assumptions, generalize rules)
- Speed up code, proofs, books, regression suites
- Restructure books (“prove this without that big library”)

- Danger:
  - Vibe coding models or specs (“spec slop”)
  - Letting the same agent that is proving the theorem also state the theorem
  - Tests that just check for current (maybe wrong) behavior
  - **Not checking AI-generated material for bad stuff (skip-proofs, defaxiom, raw mode, sys-calls !)**