

HOW TO USE AI IN MATHEMATICAL RESEARCH

TAYLOR DUPUY

ABSTRACT. This guide is my talk *How to use AI in Mathematical Research Effectively* (University of Rochester, 23 October 2026), written out with details, sources, and corrections. It covers what a large language model is; the harnesses that turn models into agents; the standards (skills, `AGENTS.md`, and MCP); models, prices, and plans; autoformalization in Lean; AI usage statements; and orchestration. How the guide itself was made is in Section D.

CONTENTS

1. Introduction	1
2. LLMs: next token predictors	3
3. Harnesses: conventional software on top of LLMs	15
4. Standards: skills, <code>AGENTS.md</code> , and MCP	20
5. Models, prices, and plans	28
6. Autoformalization	34
7. AI usage statements	42
8. Orchestration	45
9. Costs, quality, and credit	46
Appendix A. Exercises	47
Appendix B. A catalogue of skills	50
Appendix C. Other reporting styles	51
Appendix D. How this document was made	54
References	55

1. INTRODUCTION

On 23 October 2026 I gave a talk at the University of Rochester called *How to use AI in Mathematical Research Effectively*. This guide is that talk written out: the same order and the same examples, with what I said aloud typed up, plus the details, sources, and corrections that did not fit in an hour. It starts with what a large language model is, then the harness software that turns a model into an agent, the standards (skills, `AGENTS.md`, and MCP) that make agents configurable, the market of models and plans, autoformalization in Lean, AI usage statements, and orchestration. Exercises, a catalogue of skills, another style of reporting, and an account of how this document was made are in the appendices.

1.1. Where are you? I opened the talk with Steve Yegge’s chart of eight stages of working with AI (Figure 1), from the post that introduced his orchestrator Gas Town on 1 January 2026 [Yeg26] steve-yegge.medium.com/welcome-to-gas-town. In short (the labels are paraphrased): (1) near-zero AI, meaning completions and the occasional chat question; (2) an agent in the editor that asks

Date: October 25, 2026.

This work was supported by the National Science Foundation under award DMS-2401570 [U.S24].

permission; (3) an agent in the editor running unsupervised (“YOLO”); (4) the agent fills the editor; (5) a single command-line agent; (6) three to five command-line agents in parallel; (7) ten or more agents, managed by hand; and (8) your own orchestrator, a program that coordinates the agents for you. Yegge writes that unless you are at stage 7 you are not ready for Gas Town.

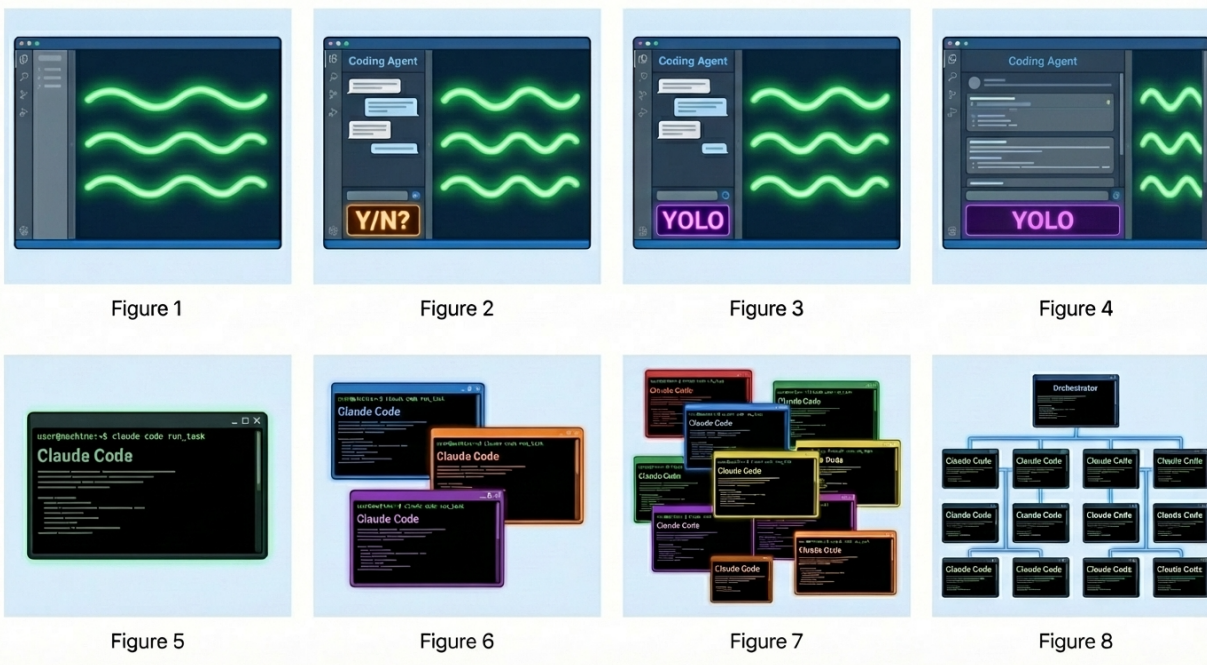


FIGURE 1. Yegge’s eight stages of working with AI [Yeg26].

Most mathematicians I talk to are at stage 1 or 2. Stage 5, a single agent in the terminal working in a well-organized folder with written rules, is the big jump, and most of this guide is about getting there. Stages 6 to 8 are the subject of the section on orchestration.

1.2. A morning report. Here is what stage 8 looks like for me. On the evening of 10 September 2026 my coordinating agent (called the Mayor; it runs on my orchestration pack *mathcity*, described with orchestration in Section 8) was working through a queue of *briefs*: short documents, each asking me to decide one thing. At about 21:40 Eastern time I typed this to it:

```
> I'm going to bed for the night. You are on your own. You have the power to
  adjudicate. Choose the easiest things to do and adjudicate based on policy.
```

Before doing anything, the agent recorded the delegation as a decision in the project’s issue tracker, together with the limits it set itself. This is that record as it is stored (abridged; the identifiers are internal):

```
$ bd show gt-y650td
Taylor, verbatim, on leaving for the night: "I'm going to bed for the night.
You are on your own. You have the power to adjudicate. Choose the easiest
things to do and adjudicate based on policy."
```

```
Scope as QUIMBY 69 construes it, conservatively:
```

```
- Verdicts DERIVED from Adopted policy only [...], each recorded with the
```

```
rule IDs it derives from and adjudicated_by naming this delegation + this
bead. A verdict that requires judgment rather than derivation is NOT
covered and waits for Taylor.
```

- Easiest-first, per instruction.
- LIMITS that survive the delegation: (1) the he-2265wo Dolt conflict -- Taylor explicitly chose 'inspection first, resolution plan back to me before any write' minutes before delegating; the more specific ruling wins, so NO conflict writes tonight. (2) Nothing irreversible without either prior explicit authorization or clean policy derivation. (3) No mass/bulk actions [...]. (4) Every verdict individually recorded -- no batch verdicts.

```
All overnight verdicts cite this bead as their authority anchor.
```

By morning it had decided nine “briefs” by derivation from adopted policy and skipped three that needed judgment. It merged a pull request under the delegation, with the derivation recorded. Earlier that evening I had also asked for a campaign on differential Gröbner bases; overnight it turned this into three public issues with required tests (Ollivier’s standard bases, a new type for partial differential algebra, and Mansfield’s differential Gröbner bases) [Dup26d], and handed them to the orchestrator, where one reached “in progress”. What was waiting for me were the three judgment calls and the repair plan I had asked to see first.

1.3. Software and AI assistance. This guide was drafted by AI agents (Claude Opus 5.5 in Claude Code) from my talk, my notes, and the sources cited, and its factual claims were checked against those sources by further agents. I chose the content and the structure and revised the text, and I am responsible for it. The full account, with the tokens used, is in Section D.

2. LLMs: NEXT TOKEN PREDICTORS

This section says what a large language model is, in the order of the talk: next-word prediction, tokens, embeddings, sampling, neural nets, transformers, and training. It then says what the labs disclose about their models, what “open-weight” means, and what it takes to run and improve an open model. It ends with a real next-token distribution, which is the reason every output has to be checked. References for depth are collected at the end.

2.1. Shannon’s next-word predictor. Take the text of *Moby Dick*, which Project Gutenberg distributes as a plain-text file [Mel]. Run over the book and, for every four consecutive words, record the word that follows them. This gives, for each sequence of four words, an empirical probability distribution on the fifth. Now start from four words of the book, draw a fifth word from its distribution, drop the first word, and repeat. The result is a random text in which every five consecutive words occur somewhere in Melville. The following program does this; its output is not reproduced here.

ngram.py

```
import collections, random, re

text = open("moby-dick.txt", encoding="utf-8").read()
words = re.findall(r"[\w']+|[,;:!?]", text)

table = collections.defaultdict(list)          # 4 words -> every 5th word seen
for i in range(len(words) - 4):
    table[tuple(words[i:i + 4])].append(words[i + 4])
```

```

state = ("Call", "me", "Ishmael", ".")
output = list(state)
for _ in range(60):
    successors = table.get(state)
    if not successors:
        break
    word = random.choice(successors)      # sample the 5th word
    output.append(word)
    state = state[1:] + (word,)
print(" ".join(output))

```

This is Shannon’s construction. In the paper that founded information theory he generated “approximations to English” in which each letter, and then each word, is drawn from its distribution conditioned on the preceding ones, by opening a book at random and reading until the conditioning text appeared [Sha48, §3]. He stopped at two letters, and one word, of context, noting that “the labor involved becomes enormous at the next stage” [Sha48, §3], and later measured the entropy of English by having people guess the next letter [Sha51]. A table of counts does not generalize: in a single book almost every sequence of four words occurs once, so the program above mostly copies Melville.

A large language model is a sophisticated version of the same machine, with three changes. It predicts the next *token*, a word or a piece of a word (Section 2.2). It represents tokens as vectors, so that tokens with similar meanings are close (Section 2.3). And it replaces the table of counts by a function of the preceding tokens built from matrices and vectors, the *parameters*, which are fitted to a very large body of text by gradient descent (Sections 2.5 and 2.7): a classical parameter-estimation problem. The function is built from MLPs and one further kind of layer, attention, which turns a tuple of token vectors into another tuple (Section 2.6).

2.2. Tokens. A model never sees letters. A fixed *tokenizer* cuts text into pieces from a finite set V , the *vocabulary*, whose elements are *tokens*. Figure 2 shows OpenAI’s tokenizer `tiktoken` with the encoding `o200k_base` (used by the GPT-4o and GPT-5 families) [Ope26k] `openai/tiktoken` applied to one sentence in plain English and in $\text{\LaTeX}$; each chip is one token, with its integer identifier underneath. The word “elliptic” is two tokens, “`\ell`ellipt” and “`ic`”, and the space before “2” is a token of its own. In the $\text{\LaTeX}$ version `\mathbb` becomes three tokens (`/\`, `math`, and `bb`) and the dollar signs fuse with their neighbours (`\$` and `$.`), so the 36 characters of English are 11 tokens and the 52 characters of $\text{\LaTeX}$ are 19. This encoding has $|V| = 200,019$ tokens, special tokens included; other tokenizers split differently, and Anthropic’s and Google’s are not published in this form.

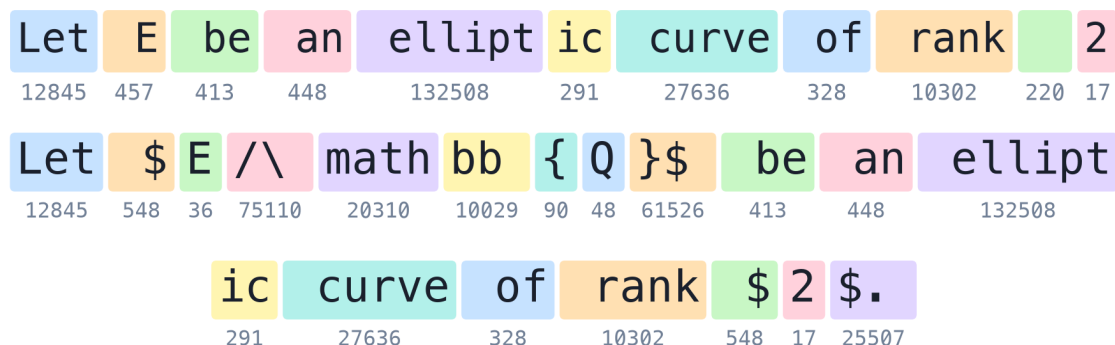


FIGURE 2. One sentence in English and in $\text{\LaTeX}$, tokenized [Ope26k].

Among the special tokens is an *end-of-sequence* token $\langle \text{eos} \rangle \in V$: training text is followed by it, so a model learns to emit it where a text ends, and generation stops when it is sampled [PH22, §4, Alg. 15]. Tokens are also the unit of account: input and output are both priced per million tokens (Section 5).

2.3. Embeddings. Identify each token $v \in V$ with the basis vector e_v of $\mathbb{R}^V$. The first step of a model is a linear map

$$E: \mathbb{R}^V \longrightarrow \mathbb{R}^d,$$

called the *embedding*; $\mathbb{R}^d$ is the *embedding space*.

Remark 2.1. Since d is much smaller than $|V|$ (Table 1), E has a kernel of dimension at least $|V| - d$; it is a projection, and the word “embedding” is accurate only for its restriction to the finite set $\{e_v\}$.

The geometry of the image carries meaning. Mikolov, Yih, and Zweig found that differences of word vectors encode relations: the vector $x_{\text{king}} - x_{\text{man}} + x_{\text{woman}}$ is closer, in cosine similarity, to x_{queen} than to any other word vector [MYZ13, §4.1, Fig. 2] (Figure 3).

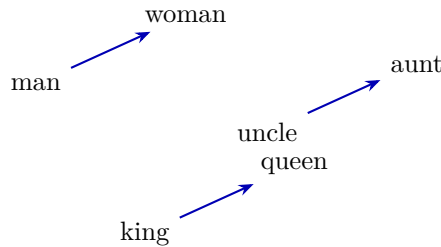


FIGURE 3. Relations as parallel offsets, after [MYZ13, Fig. 2].

Table 1 gives $|V|$, d , the number of layers, and the number of parameters of some open models. For the models marked MoE (*mixture of experts* [SMM⁺17]) each token is routed through only some of the parameters, so the table gives the total and the number active per token.

Model	$ V $	d	Layers	Parameters
GPT-2 (small), 2019	50,257	768	12	124M
GPT-2 XL, 2019	50,257	1,600	48	1.5B
GPT-3, 2020	50,257	12,288	96	175B
Llama 3 8B, 2024	128,256	4,096	32	8B
Llama 3 70B, 2024	128,256	8,192	80	70B
Llama 3.1 405B, 2024	128,256	16,384	126	405B
DeepSeek-V3 (MoE), 2024	129,280	7,168	61	671B / 37B active
gpt-oss-120b (MoE), 2025	201,088	2,880	36	117B / 5.1B active
Qwen3.5-397B-A17B (MoE), 2026	248,320	4,096	60	397B / 17B active
Table 5.1, GPT-6.1 Sol, Gemini 4 Argon			not disclosed	

TABLE 1. Vocabulary, width, depth, and size of open models [RWC⁺19, BMR⁺20, G⁺24, Dee24, Ope25a, Qwe26a].

2.4. Language models and sampling.

Definition 2.1. Let V be a vocabulary.

- (1) $V^* = \bigsqcup_{n \geq 0} V^n$ is the set of finite sequences of tokens, and $\text{Prob}(V)$ is the simplex of probability distributions on V .
- (2) A *language model* with parameters θ is a map $M_\theta: V^* \rightarrow \text{Prob}(V)$; the number $M_\theta(x)(y)$ is the probability that the token after x is y .

Text is generated by sampling. Starting from a prompt $x \in V^*$, draw $y \sim M_\theta(x)$, replace x by the concatenation xy , and repeat until $y = \langle \text{eos} \rangle$ or a length limit is reached [PH22, Algs. 14–15]. This is a Markov chain on V^* whose state is the whole sequence so far; the Moby Dick program of Section 2.1 is the case in which M depends only on the last four words and is given by a table.

Remark 2.2. In practice the draw is made at a *temperature* $T > 0$, from the distribution proportional to $M_\theta(x)(y)^{1/T}$: $T \rightarrow 0$ always picks the most likely token, $T = 1$ is the model itself, and $T \rightarrow \infty$ is uniform [PH22, §7, Alg. 14].¹

A conversation is the same chain run on a longer and longer sequence; the parameters θ do not change while you talk to the model.

Remark 2.3. In Shannon’s terms a language model is a *channel*: a conditional distribution of outputs given inputs [Sha48, §11]. This is the precise sense in which the output of a model is random. Everything below is about how M_θ is built and how θ is chosen; whatever the construction, generation is sampling from M_θ .

2.5. MLPs. Let $x(1), \dots, x(n) \in \mathbb{R}^d$ be the vectors from one stage of a network, and put $x = (x(1), \dots, x(n)) \in \mathcal{X}_n = (\mathbb{R}^d)^n$. In the notation used here, a *head* is a map $H: \mathcal{X}_n \rightarrow \mathcal{X}_n$: it takes the preceding stage’s tuple of vectors and produces a new tuple. An individual output need not use every vector in the input.

Write $\max^+(t) = \max(t, 0)$, applied to vectors coordinatewise (Figure 4).

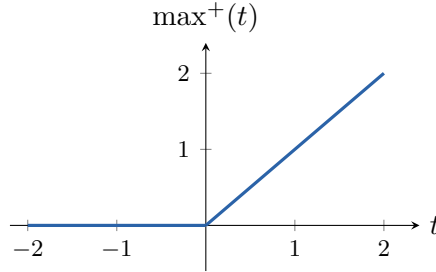


FIGURE 4. The rectified linear unit (ReLU), $\max^+(t) = \max(t, 0)$.

Definition 2.2. For a matrix $A: \mathbb{R}^d \rightarrow \mathbb{R}^d$ and $b \in \mathbb{R}^d$, the single-vector rule $F_{A,b}: \mathbb{R}^d \rightarrow \mathbb{R}^d$ is $F_{A,b}(u) = \max^+(Au + b)$. Its *MLP head* acts on the whole tuple by

$$M_{A,b}(x)(i) = F_{A,b}(x(i)) = \max^+(Ax(i) + b).$$

Choose r MLP heads $M_1, \dots, M_r$, a learned linear map $P: (\mathbb{R}^d)^r \rightarrow \mathbb{R}^d$, and $c \in \mathbb{R}^d$. Their *MLP layer* is

$$\mathcal{L}(x)(i) = P(M_1(x)(i), \dots, M_r(x)(i)) + c.$$

A finite composition of MLP layers is an *MLP neural net* on $\mathcal{X}_n$.

¹The *effort* setting of Claude and the *reasoning effort* of OpenAI’s models are not temperatures. They control how many tokens the model spends, hidden “thinking” tokens included [Ant26h, Ope26j].

When $n = 1$, one MLP layer is an affine combination of r ReLU heads, the familiar one-hidden-stage MLP. Composing layers, allowing the vector width to vary between stages, and ending with an affine readout gives the usual definition of an MLP (multilayer perceptron): for compatible matrices and biases,

$$z_0 = u, \quad z_\ell = \max^+(A_\ell z_{\ell-1} + b_\ell) \quad (1 \leq \ell < L), \quad f(u) = A_L z_{L-1} + b_L.$$

Several MLP heads in one layer can be folded into a wider hidden vector; the head notation makes the parallel branches explicit. For $n > 1$, the same MLP acts separately at every position. We use *MLP head* as terminology for this exposition; standard transformer descriptions call the position-wise network an MLP or feed-forward sublayer [VSP⁺17, §3.1]. Other activation functions can replace ReLU.²

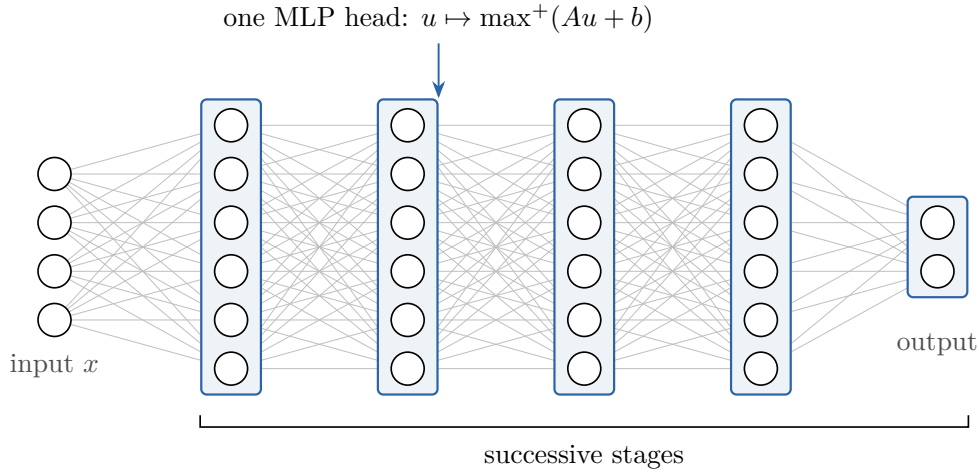


FIGURE 5. An MLP composed of one-head ReLU stages and an affine output. An MLP layer may use several heads.

Fitting a net to data is parameter estimation: choose the parameters to minimize a loss over examples, by gradient descent, with the gradient computed by the chain rule (*backpropagation*) [GBC16, Ch. 6].

2.6. Transformers. For $m \geq 1$, write $[m] = \{1, \dots, m\}$. The *softmax* map

$$\text{softmax}: \mathbb{R}^m \longrightarrow \text{Prob}([m]), \quad \text{softmax}(s)(i) = \frac{e^{s_i}}{\sum_{r=1}^m e^{s_r}}$$

turns a vector $s = (s_1, \dots, s_m)$ into a probability distribution on m elements: every coordinate is positive and their sum is one (Figure 6).

²Convolutional neural networks (CNNs) have a similar layered form, but their matrices are constrained to be convolutions, which makes the layers commute with translations [BBCV21, §5.1].

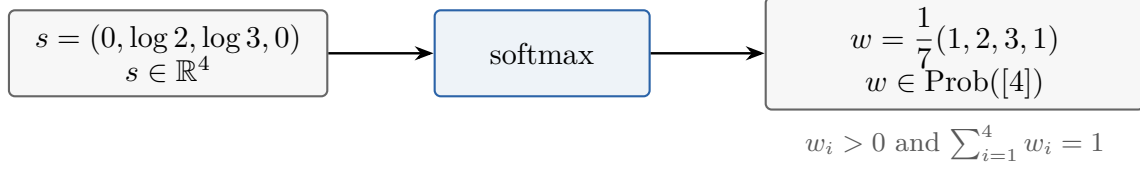


FIGURE 6. An illustrative four-coordinate score vector becomes a probability distribution on four positions.

Definition 2.3. For the token vectors $x(1), \dots, x(n)$, an *attention head* is a map $A_{A,B,C}: \mathcal{X}_n \rightarrow \mathcal{X}_n$ specified by three learned matrices $A, B, C: \mathbb{R}^d \rightarrow \mathbb{R}^d$. For each i , define

$$v(i) = (\langle Ax(i), Bx(j) \rangle : j = 1, \dots, n), \quad w(i) = \text{softmax}(v(i)),$$

and write $w(i, j)$ for the j th coordinate of $w(i)$. The output $A_{A,B,C}(x) = (y(1), \dots, y(n))$ has

$$y(i) = \sum_{j=1}^n w(i, j) Cx(j).$$

For example, if $n = 4$ and $v(4)$ is the score vector in Figure 6, then

$$y(4) = \frac{1}{7}Cx(1) + \frac{2}{7}Cx(2) + \frac{3}{7}Cx(3) + \frac{1}{7}Cx(4).$$

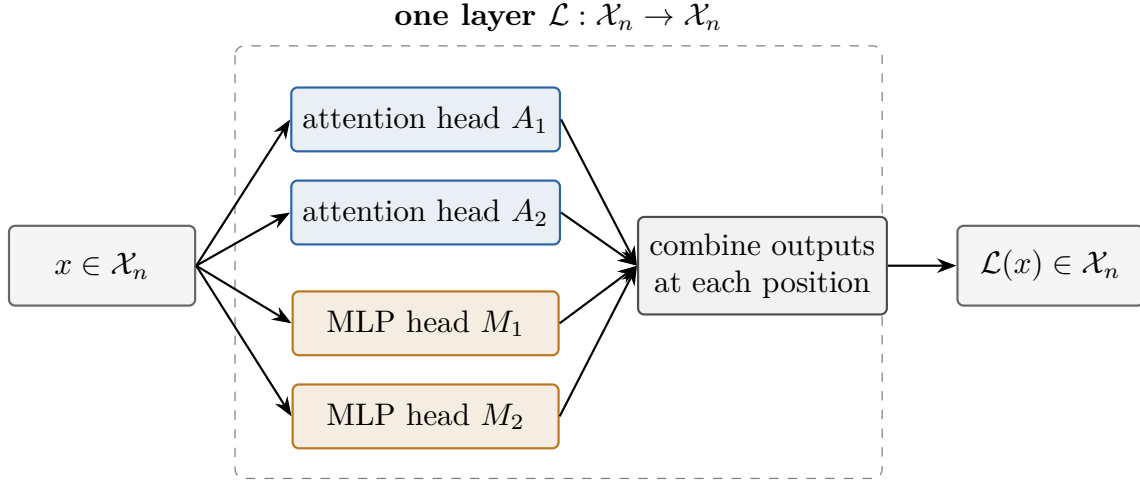
Each $y(i)$ is a weighted average of transformed token vectors. The matrices A, B, C are fitted parameters; the weights $w(i, j)$ are calculated afresh from each input. The conventional factor $1/\sqrt{d}$ in the scores can be absorbed into A [VSP⁺17, §3.2].

In this form, position i can use every position j . For next-token prediction we use *causal* attention: in $v(i)$ keep only the scores with $j \leq i$, apply softmax to that shorter vector, and sum only over $j \leq i$ in $y(i)$. Thus i sees itself and earlier positions. For example, at “it” in “The ring R is commutative, so it ...”, one head might put substantial weight on the earlier R . The word *self* says that both positions come from the same x .

The MLP heads of Definition 2.2 and attention heads above are all maps $\mathcal{X}_n \rightarrow \mathcal{X}_n$. A layer can combine either kind: choose heads $H_1, \dots, H_r$, a learned linear map $P: (\mathbb{R}^d)^r \rightarrow \mathbb{R}^d$, and $c \in \mathbb{R}^d$, then set

$$\mathcal{L}(x)(i) = P(H_1(x)(i), \dots, H_r(x)(i)) + c.$$

All the heads read the same x . A layer of attention heads is an *attention layer*; a layer of MLP heads is an *MLP layer*; a layer may also contain both kinds. An MLP head’s output at position i uses only $x(i)$; an attention head’s output can use the other positions too (Figure 7).



All heads read the same x ; the combination returns one vector per position.

$$\mathcal{X}_n \xrightarrow{\mathcal{L}_1} \mathcal{X}_n \xrightarrow{\mathcal{L}_2} \dots \xrightarrow{\mathcal{L}_L} \mathcal{X}_n$$

FIGURE 7. Heads of either kind take the same collection of token vectors $x \in \mathcal{X}_n$. Combining their outputs gives a layer $\mathcal{L} : \mathcal{X}_n \rightarrow \mathcal{X}_n$; all-attention, all-MLP, and mixed layers use the same interface.

Because every layer maps $\mathcal{X}_n$ to itself, layers compose: $\mathcal{L}_L \circ \dots \circ \mathcal{L}_1$. The Rochester talk's schematic in Figure 8 shows one familiar transformer arrangement: an attention layer followed by an MLP layer, with skip connections and normalization. The definition of a layer also permits other combinations.

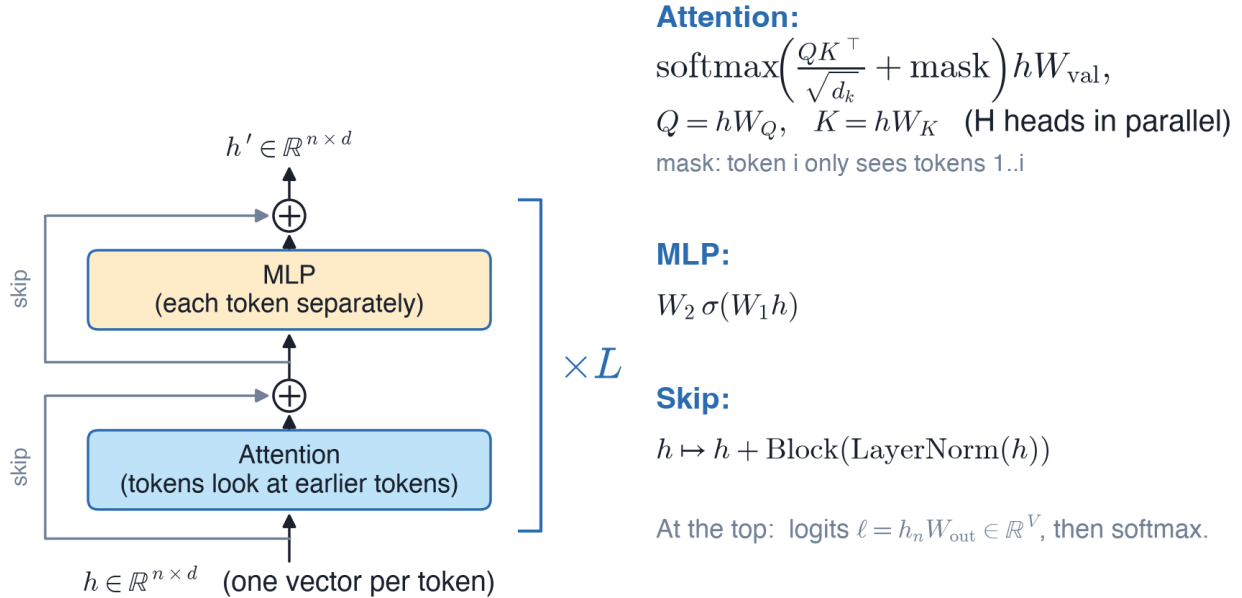


FIGURE 8. One transformer arrangement from the Rochester talk. The attention step mixes token positions; the MLP step acts at each position. Each step adds its input back.

The embedded token sequence lies in $\mathcal{X}_n$. After the layers, a final linear map and softmax send its last vector to a probability distribution on the vocabulary:

$$V^n \xrightarrow{E} \mathcal{X}_n \xrightarrow{\mathcal{L}_L \circ \dots \circ \mathcal{L}_1} \mathcal{X}_n \longrightarrow \text{Prob}(V).$$

In training the output map is applied at every position, so position i predicts token $i + 1$; in generation only the last position is used. That fitting long compositions of these maps to next-token data works so well is an empirical fact, not something the formulas alone explain.

2.7. Training. Training chooses θ , in two stages. *Pre-training* minimizes a certain function ³ over a very large corpus of text by gradient descent. ⁴ This is unsupervised learning. One can predict text and then use a sample of text to see how the model is performing. The output of this process is called the model.

Post-training adapts the model. Supervised fine-tuning on demonstrations (a prompt and a good answer) makes it behave like an assistant. Reinforcement learning then samples outputs $y \sim M_\theta(x)$, scores each with a *reward function* $r(x, y)$, and adjusts θ to increase $\mathbb{E}_{y \sim M_\theta(x)}[r(x, y)]$, with a penalty for moving far from a reference model. The reward comes either from a model trained on human preference comparisons (*RLHF*, reinforcement learning from human feedback [OWJ⁺22]) or from an automatic check of the answer, such as a known final answer or a test suite (*RLVR*, reinforcement learning with verifiable rewards [L⁺24]). Training changes which distribution M_θ outputs, not the fact that generation samples from it.

Remark 2.4. The effect of the second kind can be large. DeepSeek-R1-Zero was trained by reinforcement learning with rule-based accuracy rewards directly from a base model; its average pass@1 on the 2024 AIME rose from 15.6% to 77.9% over about 10,000 steps, and 86.7% with the majority answer of 16 samples (Figure 9) [DG⁺25]. The dashed line is the average score of human AIME participants.

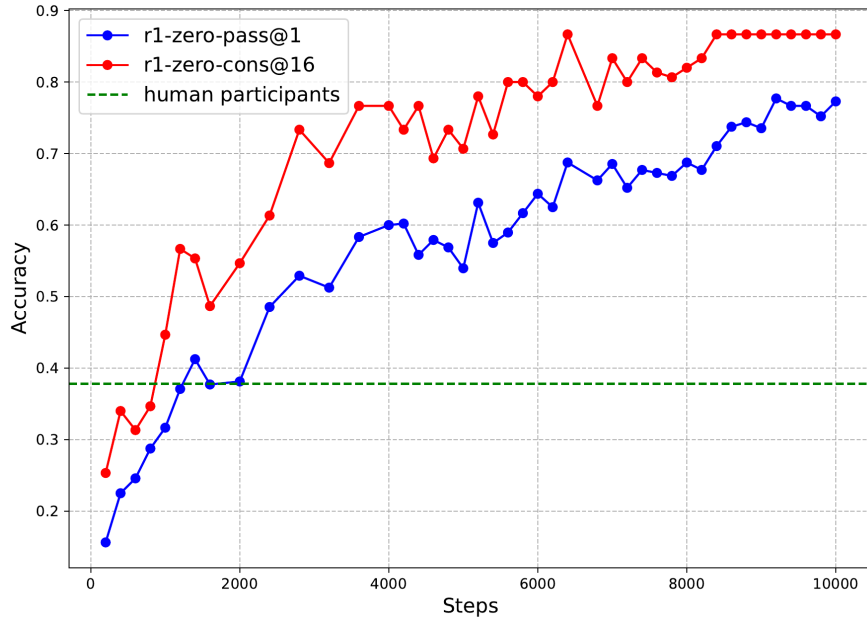


FIGURE 9. DeepSeek-R1-Zero’s AIME 2024 accuracy during training [DG⁺25, Fig. 1(a)].

³the cross-entropy $-\sum_i \log M_\theta(t_1 \dots t_i)(t_{i+1})$

⁴Llama 3 was pre-trained on 15.6 trillion tokens [G⁺24].

Remark 2.5. A proof checker is a verifiable reward: Lean accepts a proof or does not. Published systems use it this way. AlphaProof learns to find Lean proofs by reinforcement learning on millions of autoformalized problems [HMS⁺26], and DeepSeek-Prover-V2 is trained with a reward of 1 for each generated Lean proof that the checker verifies [RSS⁺25, §2.2].

Remark 2.6. Mathematics also serves as a benchmark. FrontierMath consists of unpublished problems with automatically checkable answers, of which the best models solved under 2% in November 2024 [GEB⁺24]; in First Proof, expert reviewers grade the solutions of AI systems to ten research problems with known solutions of at most twelve pages; the third batch is in Figure 10 [Fir26b]. Current standings on general benchmarks are discussed with prices (Section 5), and First Proof with autoformalization (Section 6).

2.8. What is disclosed, and open-weight models. Some developers describe their models in detail. Google’s Gemini 2.5 technical report says that the models are sparse mixture-of-experts transformers, describes the pre-training data by kind (web documents, code, images, audio, and video, with a cutoff of January 2025), and names the training hardware, but it gives no parameter counts [Gem25, §2]. DeepSeek’s reports give the architecture and the parameter counts [Dee24]. For the newest frontier models of Anthropic, OpenAI, and Google, the architecture, the number of parameters, the training data, and the compute are not disclosed (Table 1).

A model is *open-weight* when the developer publishes θ , the *weights*, as files anyone can download, usually from Hugging Face. That is all it means. *Open source*, in the Open Source Initiative’s definition, also requires the training code and enough information about the training data to rebuild the system [Ope24], and almost no strong model ships those. The licenses vary: Qwen, Gemma, and gpt-oss use Apache 2.0, DeepSeek and Xiaomi use MIT, and Kimi K3 and GLM-5.3 use MIT-style licenses whose extra conditions bind companies above a revenue threshold (\$20 million a year for Kimi, \$10 billion for GLM) that sell the model as a service [Moo26b, Z.a26c]. An open model run on your own machine keeps unpublished work on that machine, has no rate limits, and does not change: a weights file gives the same distribution in five years, while hosted models are retired on the provider’s schedule [Oll26b]. Hosted open models are also cheap: DeepSeek sells V4.1 Flash at \$0.15 per million input tokens and \$0.60 per million output tokens [Ope26q].

On research mathematics the frontier is far ahead. In the third First Proof batch the open-weight entrant, built on Kimi K3, had major errors in every submission; the organizers found it “generally less capable than the leading frontier model of December 2025” [Fir26b] (Figure 10). On the Artificial Analysis Intelligence Index the best open-weight model scores 46 against 58 for the best closed one [Art26c, Art26b].



FIGURE 10. First Proof, third batch, 14 October 2026 [Fir26b].

The best open-weight models. Table 2 lists strong open-weight models (Artificial Analysis also lists StepFun’s Step 5 Preview, at 44) and the best ones that fit on ordinary hardware. The total number of parameters decides how much memory a model needs; the active number decides how fast it runs.

Model	Developer	Total / active	License	AA	Ollama tag (4-bit size)
MiMo-V2.6-Pro	Xiaomi	1.02T / 42B	MIT	46	none; Hugging Face only
DeepSeek V4.1	DeepSeek	552B / 16B	MIT	45	deepseek-v4.1-flash:cloud
Flash					
GLM-5.3	Z.ai	753B / 40B	own	45	glm-5.3:cloud
Kimi K3	Moonshot AI	2.8T / 104B	own	44	kimi-k3:cloud
GLM-5.3-Flash	Z.ai	320B / 18B	MIT	42	glm-5.3-flash:cloud
Qwen3.8-27B	Alibaba	27.8B dense	Apache	34	qwen3.8:27b (18 GB)
Qwen3.6-35B-A3B	Alibaba	35B / 3B	Apache	18	qwen3.6:35b (23–24 GB)
Gemma 4 31B	Google	30.7B dense	Apache	15	gemma4:31b (19–20 GB)
Gemma 4 26B A4B	Google	25.2B / 3.8B	Apache	17	gemma4:26b (16–19 GB)
gpt-oss-120b	OpenAI	117B / 5.1B	Apache	12	gpt-oss:120b (65 GB)
gpt-oss-20b	OpenAI	21B / 3.6B	Apache	9	gpt-oss:20b (14 GB)
<i>For comparison:</i> Claude Opus 5.5 (closed)				58	—

TABLE 2. Open-weight models, 25 October 2026; AA is the Artificial Analysis Intelligence Index v4.3.2.

Sources for Table 2: parameters and license (“own” means the developer’s own license) from the Hugging Face model cards [Xia26, Dee26, Z.a26a, Moo26a, Z.a26b, Qwe26c, Qwe26b, Goo26d, Ope25a, Ope25b], except the GLM-5.3 counts, which are from Artificial Analysis [Art26c]; scores from Artificial Analysis at the highest reasoning setting listed [Art26c, Art26b, Art26a]; Ollama tags and download sizes from the Ollama library [Oll26k, Oll26j, Oll26d, Oll26g, Oll26e, Oll26f, Oll26h, Oll26c]. The top five do not run on a laptop; a tag ending in `:cloud` means the model runs

on Ollama’s servers, not yours [Oll26b]. Between them and the models that fit in 32 GB (scores 34 and below) there is a cliff, and Qwen3.8-27B is the strongest model on the laptop side. The skill `get-best-models` redoes this search for a given amount of memory.

What it takes to run them. The binding constraint is memory. A model with P parameters stored in b bytes each needs about $P \cdot b$ bytes, plus a cache that grows with the length of the text it reads, plus overhead; $b = 2$ at 16 bits (how most models are released), 1 at 8 bits, and about 0.5 at 4 bits. So Qwen3.8-27B needs about 56 GB at 16 bits and 14 GB at 4 bits; Ollama’s 4-bit file is 18 GB, because practical 4-bit formats spend nearer 5 bits per weight and the file includes an image encoder [Oll26k]. Its cache is 64 KiB per token, about 2 GB for 32,000 tokens of text [Qwe26c]. On a Mac the graphics processor shares the machine’s unified memory; on a PC what counts is the graphics card’s memory. Storing weights in 8 or 4 bits (*quantization*) is lossy, but a large study on Llama 3.1 found 8-bit weights “effectively lossless” and 4-bit weights “rivaling 8-bit” [KMP⁺24]; those are standard benchmarks, not long chains of reasoning. The cheapest hosted providers often serve 4-bit versions, and OpenRouter lists the precision of each [Ope26q].

Speed is bounded by memory bandwidth: to write each token the machine reads every active weight once, so

$$\text{tokens per second} \lesssim \frac{\text{memory bandwidth}}{\text{active parameters} \times b}.$$

On an M3 Max with 400 GB/s, a 7B model at 4 bits writes 66 tokens per second against a bound of about 105 [lla23]. This is why mixtures of experts matter on a laptop. Table 3 puts this together.

Tier	What fits (4-bit unless noted)	Rough speed	Price
Laptop, 16–32 GB (MacBook Pro M5: 153 GB/s)	gpt-oss-20b, small Gemma 4 at 16 GB; Qwen3.8-27B, Gemma 4 26B or 31B at 32 GB	MoE (3–4B active): tens of tokens/s; 27B dense: under 10	the laptop you have
Workstation, 64–128 GB (MacBook Pro or Mac Studio, M5 Max: 460–614 GB/s)	gpt-oss-120b (65 GB); Qwen3.8-27B at 8 or 16 bits; Qwen3.6-35B-A3B	27B dense at 4 bits: about 20–30 tokens/s	Mac Studio M5 Max from \$2,499 with 36 GB; more memory costs more
Big unified memory, 256–512 GB (M5 Ultra: 1.2 TB/s)	GLM-5.3-Flash ($\approx$ 160 GB); DeepSeek V4.1 Flash ($\approx$ 280 GB, 512 GB only)	16–18B active: tens of tokens/s	Mac Studio M5 Ultra from \$5,499; no published price for 512 GB
Data-center GPUs (H100 80 GB, B200 180 GB)	gpt-oss-120b on one H100; DeepSeek V4.1 Flash or GLM-5.3 ($\approx$ 380 GB) on eight H100s; Kimi K3 ($\approx$ 1.4 TB) needs more than one 8-GPU machine	fast, and serves many users at once	rent H100 \$3.29–\$4.29, B200 \$6.69–\$6.99 per GPU-hour
Somebody else’s GPUs (API)	every model in Table 2	depends on the provider	DeepSeek V4.1 Flash \$0.15/\$0.60; GLM-5.3 \$1.40/\$4.40; Kimi K3 \$3/\$15 per million in/out

TABLE 3. Hardware tiers for open-weight models, October 2026; speeds are estimates from the bandwidth bound.

Sources for Table 3: memory sizes and bandwidths from Apple’s specifications [App26b, App26a]; Mac Studio prices from Apple’s announcement as reported by MacTech [Mac26]; “fits on a single 80 GB GPU” from the gpt-oss card [Ope25a]; GPU rental from Lambda’s on-demand price list [Lam26]; API prices are each developer’s own, as listed on OpenRouter [Ope26q]. The big-model

memory figures are $P \times 0.5$ from Table 2, before the cache. Eight rented H100s cost about \$32 an hour, so renting by the hour pays only for large batch jobs or for data that must stay on hardware you control; for the big models, “open” mostly means that you choose who runs them.

Using and improving open models. An open model needs a harness, the program that sends it text and runs the tools it asks for (Section 3). With Ollama [Oll26i] installed, `ollama launch claude` runs Claude Code on a local model [Oll26a], and `codex -oss` does the same for Codex [Ope26a]. Those harnesses are tuned for their own labs’ models; harnesses not tied to a vendor, and how to point any harness at any model, are described with harnesses (Sections 3.2 and 3.5). In practice I use a frontier model for proofs and literature and an open model for work that should not leave my machine or that would exhaust a plan’s limits.

Open weights can also be changed. *Fine-tuning* continues training on your own data. LoRA freezes the weights and trains a low-rank correction $A_i \mapsto A_i + B_i C_i$ to each matrix, which for GPT-3 reduces the number of trainable parameters by a factor of 10,000 and the memory needed by a factor of three [HSW⁺21]. QLoRA does the same on a 4-bit model and fine-tunes a 65B model on a single 48 GB GPU; its best model needed 24 hours on one GPU [DPHZ23], which at the H100 rental prices of Table 3 is about \$80–\$100. The open Lean provers are fine-tuned open models of this kind. DeepSeek-Prover-V2 proves 88.9% (at pass@8192) of the MiniF2F test set and 47 of the 658 PutnamBench problems [RSS⁺25], and Goedel-Prover-V2-32B, which uses feedback from the Lean compiler to revise its proofs, proves 88.1% (and 90.4% with self-correction) of MiniF2F at pass@32 [LTL⁺25]; both release their weights.

2.9. A real next-token distribution. For the prompt “Let E be an elliptic curve over $\mathbb{Q}$. By the Mordell-Weil theorem, the group $E(\mathbb{Q})$ is”, the small open model Llama 3.2 3B [Met24], run on a laptop with Ollama [Oll26i] on 22 October 2026, gives the probabilities in Figure 11 to the eight most likely of its 128,256 tokens (`␣` marks a leading space): 34.3% to “`␣fin`”, 32.3% to “`␣finite`”, and 24.4% to “`␣a`”. After “`␣fin`” the next token is “`it`” (as in “finitely generated”) with probability 99.7%. After “`␣finite`” the sentence ends at once with probability about 67%, so roughly one time in five this model says that $E(\mathbb{Q})$ is finite. Lowering the temperature to 0.5 gives “`␣fin`” 42% and “`␣finite`” 37%: sharper, but the error is still there.

Prompt:

“Let E be an elliptic curve over $\mathbb{Q}$. By the Mordell-Weil theorem,
the group $E(\mathbb{Q})$ is `␣` ”

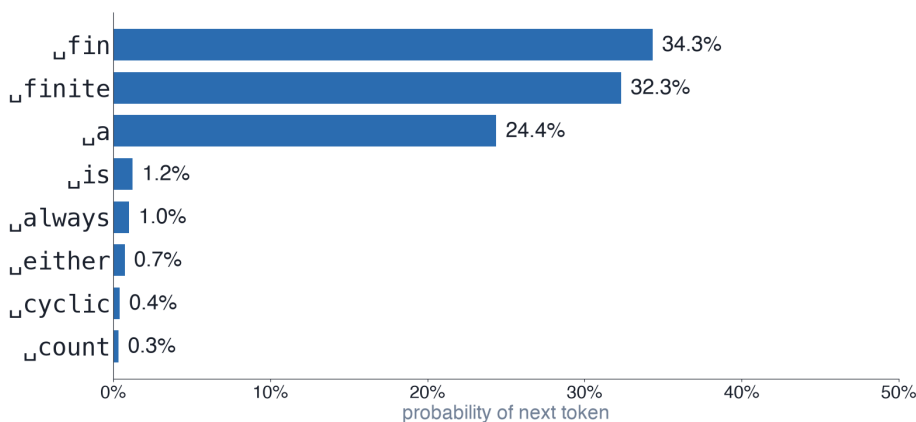


FIGURE 11. Next-token probabilities from Llama 3.2 3B [Met24], 22 October 2026.

Nothing in Definition 2.1 distinguishes true from false. Frontier models are much sharper on this prompt, but generation is still sampling from M_θ , which is why asking twice can give two answers, and why every output has to be checked.

Further reading. Turner’s short introduction [Tur23] describes the transformer precisely for mathematically minded readers, and Phuong and Hutter [PH22] give pseudocode for every step above. Lecture 8, “Transformers”, of MIT’s 6.7960 Deep Learning (Fall 2024, Phillip Isola) treats attention with examples from vision [Iso24] ocw.mit.edu. The standard textbook is [GBC16], and the lectures of the seminar *Machine Learning for the Working Mathematician* [GGW22] explain how the problems are formulated.

3. HARNESSES: CONVENTIONAL SOFTWARE ON TOP OF LLMs

A language model, as described in Section 2, maps a string of tokens to a probability distribution on the next token. Everything else that people call “AI” in research (reading your files, running Sage or Lean, editing a $\text{\LaTeX}$ source, querying a database) is done by ordinary software wrapped around the model. This section describes that software, how to install it, and how it manages what the model sees.

3.1. Model, harness, agent. A *model* (or LLM, large language model) is the trained network itself: it reads text and produces text. Claude Haiku, Sonnet, Opus, and Fable (Anthropic) and Luna, Terra, Sol, and Astra (OpenAI) are models. A *harness* is the conventional software that lets a model act on a computer: it runs the model in a loop, shows it your files, runs the commands it asks for, and feeds the results back. Claude Code and Codex are harnesses. People often say “Claude” for Claude Code and “GPT” for Codex, but the two layers are separate products, and either harness can run the other lab’s models (Section 3.5).

Definition 3.1. An *agent* is a model together with a harness: the model produces text, and the harness, which is conventional software, manages the context and manages the tools. The two talk to each other in JSON.

The *context* is everything the model sees on one call: the harness’s own instructions, your instructions, the files it has read, the output of commands, and its own earlier replies. *JSON* is a plain-text format for structured data, lists of labelled fields in braces; it appears in logs, and you never need to write it yourself. A model on its own can only produce text. When it wants to use a tool, the text it produces is a structured request; the harness carries the request out and appends the answer to the context, and the model reads the answer on its next call. Figure 12 shows the shape of one such exchange schematically; the listings below are the real one.

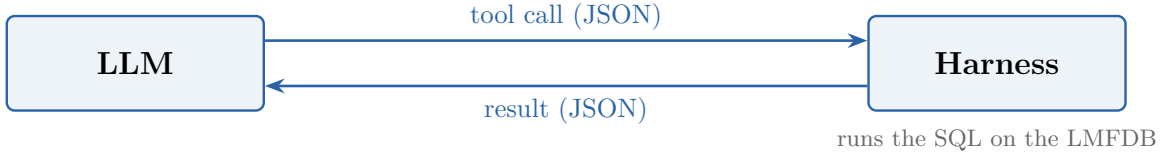
Here is one round of that loop, from the example of the rank-2 elliptic curve of smallest conductor (the full exercise is in the exercises appendix, Section A). Asked for that curve, with the LMFDB’s MCP server connected (Section 4.3), the model emits a tool call. The field names below are simplified from Claude’s `tool_use` block [Ant26q]; each harness wraps the call in its own JSON, but the content is the same.

```
{
  "type": "tool_use",
  "name": "run_sql",
  "input": {
    "sql": "SELECT lmfdb_label, ainvs, conductor, rank
           FROM ec_curvedata WHERE rank = 2
           ORDER BY conductor LIMIT 1"
  }
}
```

The harness sends the query to the LMFDB server and appends the server’s reply, verbatim (run on 25 October 2026), to the context:

```
{
  "columns": ["lmfdb_label", "ainvs", "conductor", "rank"],
  "row_count": 1,
  "rows": [{"lmfdb_label": "389.a1", "ainvs": ["0", "1", "1", "-2", "0"],
    "conductor": 389, "rank": 2}]}
}
```

On its next call the model reads this and answers that the curve is 389.a1, $y^2 + y = x^3 + x^2 - 2x$, of conductor 389. A chat window in a browser runs the same models inside a much weaker harness (Section 3.2.1).



the model emits:

```
{
  "tool": "lmfdb.run_sql",
  "input": {
    "sql": "SELECT lmfdb_label FROM ec_curvedata
    WHERE rank=2 ORDER BY conductor LIMIT 1"
  }
}
```

the harness replies; the reply is appended to the context:

```
{"result": "389.a1"}      389.a1:  $y^2 + y = x^3 + x^2 - 2x$ , conductor 389
```

FIGURE 12. One tool call: the model asks, the harness runs it and replies.

3.2. Which harness. Each lab ships a terminal agent, a desktop app, and a web app (Table 4). The terminal agents are the research tools, because they read and write the files in a folder and run Sage, Magma, Lean, and $\text{\LaTeX}$ on your machine; the desktop apps run the same agents in a window. There are also open-source harnesses not tied to any lab: OpenCode [Ope26m] [anomalyco/opencode](#) and Oh My Pi [Böl26] [can1357/oh-my-pi](#), a fork of Mario Zechner’s minimal agent Pi [Zec26]; they matter most for open-weight models (Section 2.8). I work in Claude Code and use Codex as a second opinion. The documentation is at [Ant26a] [code.claude.com/docs](#) for Claude Code and at [Ope26c] [learn.chatgpt.com/docs/codex/cli](#) for Codex. Instruction files and skills are open standards read by all of these harnesses, so moving between them costs little (Section 4).

	OpenAI	Anthropic	Google	Open source
Terminal (CLI)	Codex CLI	Claude Code	Antigravity CLI agy	OpenCode; Oh My Pi omp
Desktop app	Codex app	Claude app (Claude Code and Cowork)	Antigravity IDE	—
Web	ChatGPT	claude.ai	Gemini app	—

TABLE 4. Harnesses by vendor and form, October 2026.

3.2.1. What the web app cannot do. The chat windows at [claude.ai](#) and [chatgpt.com](#) run the same models as the terminal agents, but they see only what is pasted or uploaded into one conversation. A chat window cannot read a folder of notes, a $\text{\LaTeX}$ source, and its bibliography together and

edit them in place. It cannot run Sage, Magma, Lean, or L^AT_EX on your machine and look at the output. It does not read a standing file of instructions for the project (AGENTS.md, Section 4.2) at the start of every session, and its changes are not recorded in version control, where each one can be inspected and undone.

3.3. Installing a harness, and the terminal. A *terminal* is a window in which you type commands instead of clicking; on a Mac, press Cmd+Space, type **Terminal**, and press Enter [Ant26p]. At any moment the terminal is “in” one folder, and commands act there: **pwd** prints that folder, **ls** lists what is in it, and **cd papers/pcf** moves into a subfolder (**cd ..** moves back up). Once an agent is running, you mostly type English, and the agent types the commands.

Claude Code needs a Pro, Max, Team, or Enterprise plan or an API account [Ant26a]; Codex works with a ChatGPT plan [Ope26d]. On a Mac or Linux machine each is one line, and the version check confirms it worked (the output is from my machine on 25 October 2026):

```
$ curl -fsSL https://claude.ai/install.sh | bash
$ claude --version
2.1.310 (Claude Code)
$ curl -fsSL https://chatgpt.com/codex/install.sh | sh
$ codex --version
codex-cli 0.149.0
```

With Homebrew, **brew install -cask claude-code** and **brew install -cask codex** also work. To start, move to the folder you want to work in and type **claude** (or **codex**). The first run opens a browser to log in; for Codex, *Sign in with ChatGPT* uses the plan rather than per-token billing. Inside Claude Code, Esc interrupts the agent, **/help** lists commands, and **exit** quits. On Windows the install pages give the equivalent commands.

3.4. Turning off training. On Anthropic’s consumer plans (Free, Pro, and Max, including Claude Code used from those accounts), chats and coding sessions are used to train models whenever the model-training setting is on [Ant25c, Anta]. The setting is in the Claude app or at claude.ai, under Settings, then Privacy: *Help Improve our AI models*. Turning it off stops training on new sessions; it does not remove anything already used. Commercial plans (Team, Enterprise, and the API) are not used for training by default [Antb]. ChatGPT and Codex have an equivalent setting in ChatGPT’s settings. This is the first thing to change after installing, before anything unpublished goes into a session; what the AMS and journals require beyond it is in Section 7.

3.5. Any model in any harness. A harness sends its requests to an *endpoint* (a web address) with a *key* (a password for that service), both set in its configuration. Changing them runs OpenAI’s models inside Claude Code, Anthropic’s models inside Codex, or an open model on your own machine inside either. The endpoint can be a lab, a router such as OpenRouter (one key, every model), or an Ollama server on your laptop. The settings below are adapted from the documentation of OpenRouter [Ope26p], Ollama [Oll26a], and Codex [Ope26a]; the model names come from OpenRouter’s model list on 22 October 2026, and the snippets were not run. For Claude Code, environment variables:

```
$ export ANTHROPIC_BASE_URL="https://openrouter.ai/api"
$ export ANTHROPIC_AUTH_TOKEN="$OPENROUTER_API_KEY"
$ export ANTHROPIC_API_KEY=""
$ export ANTHROPIC_DEFAULT_SONNET_MODEL="openai/gpt-6.1-sol"
$ claude
$ ollama launch claude          # or a local model through Ollama
```

For Codex, a provider entry in its configuration file, or `codex -oss` for a local model served by Ollama or LM Studio:

`~/.codex/config.toml`

```
model = "anthropic/claude-sonnet-5.5"
model_provider = "openrouter"
[model_providers.openrouter]
name = "OpenRouter"
base_url = "https://openrouter.ai/api/v1"
env_key = "OPENROUTER_API_KEY"
```

The empty `ANTHROPIC_API_KEY` matters for billing, as explained with the plans (Section 5.5). Each harness is tuned for its own lab’s models; OpenRouter’s documentation warns that “Claude Code is optimized for Anthropic models and may not work correctly with other providers” [Ope26p].

3.6. Context windows. A model keeps no memory between calls. On every call the harness assembles one long input, the context: its own system prompt, `AGENTS.md`, the descriptions of the installed skills, a description of every tool, and the whole conversation so far, including every file the agent has read and the output of every command it has run. The *context window* is the largest number of tokens one call can contain, the model’s output included. Current Claude models allow one million tokens [Ant26f], roughly 750,000 words. A few papers, a long log file, and an afternoon of conversation fill a large part of it, and every token in the window is billed on every call.

A larger window does not mean better answers. Anthropic’s documentation states that “as token count grows, accuracy and recall degrade, a phenomenon known as *context rot*” [Ant26f]; Chroma measured the effect on 18 models, including on tasks as simple as repeating a list of words [HTH25]. Wrong, missing, and excess information in the context are, in Dietert’s account, the main causes of bad output [Die26].

Models also have a limit on how many instructions they follow at once. Figure 13 shows the IFScale benchmark, which asks for a business report that uses up to 500 given keywords: accuracy is near perfect up to a threshold and then falls quickly. In 2025 “even the best frontier models only achieve 68% accuracy at the max density of 500 instructions” [JWSM25], and models favoured the earlier instructions. The thresholds rose by an order of magnitude between 2025 and 2026, but the shape of the curve did not change [Vos26]. In Voss’s figure, reproduced with attribution, the dashed curves are 2025 models, the solid ones May 2026 frontier models, and the horizontal axis is logarithmic. An instruction file with many rules is subject to the same curve.

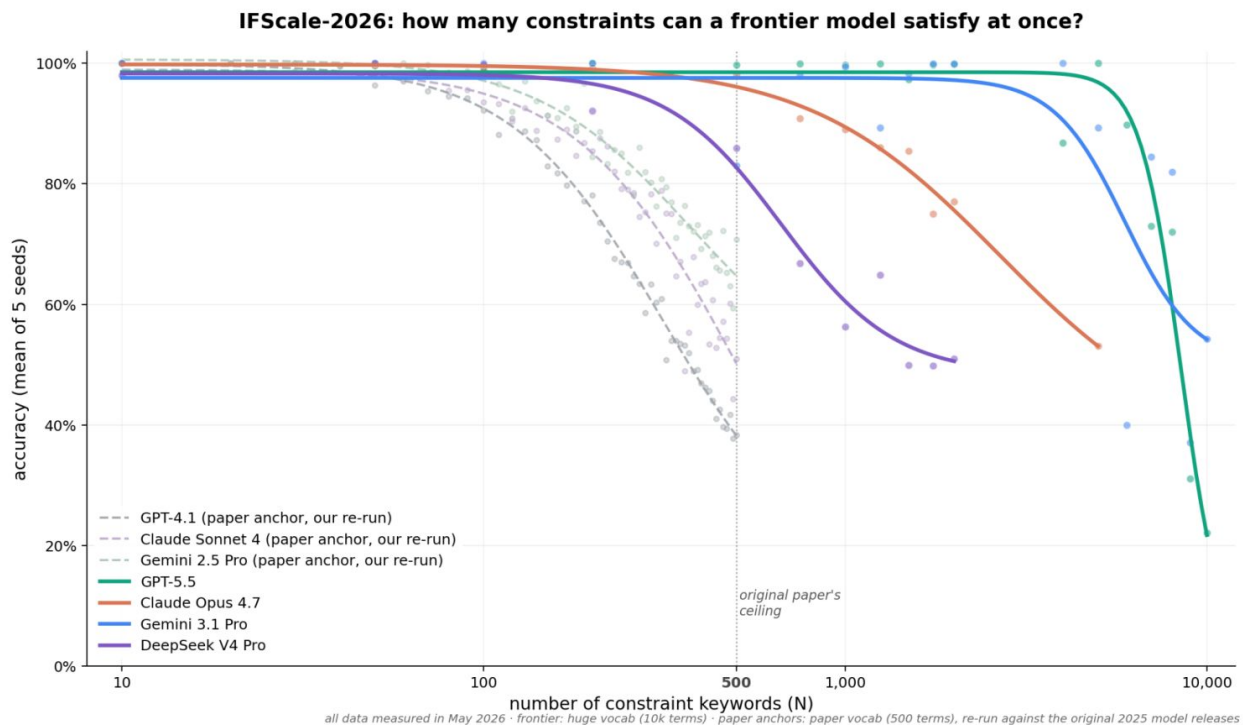


FIGURE 13. Instructions followed against instructions given, by Voss [Vos26].

People who run agents every day report that quality drops well before the window is full. Dex Horthy calls the degraded region “the dumb zone”: in his talk, “around the 40% line is where you’re going to start to see some diminishing returns depending on your task” [Hor25b]. His written guide recommends keeping context use to 40–60% of the window [Hor25a], and Dietert suggests 40–50% [Die26]. These are rules of thumb, not measured constants. Horthy also points out that with many MCP tools loaded a session starts in the dumb zone, since every tool’s description sits in the context [Hor25b]. Figure 14 draws the window as a bar, with segment sizes for illustration only.

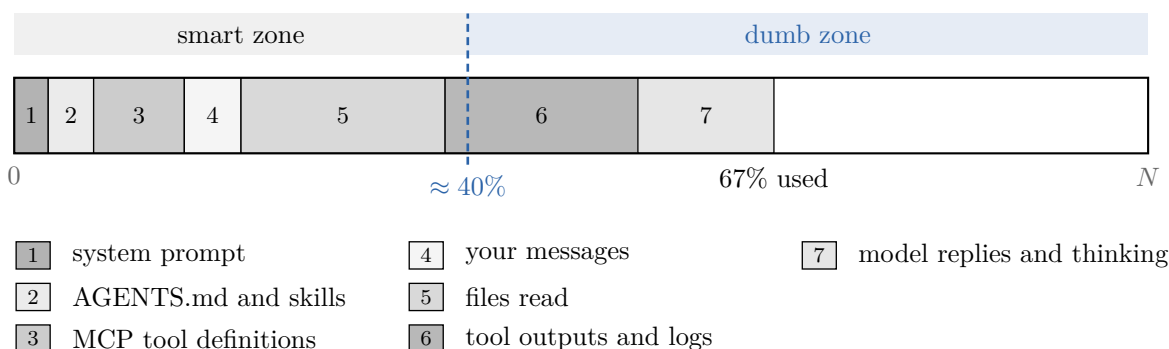


FIGURE 14. The context window and the dumb zone, after Horthy [Hor25b].

3.7. Managing the context. Harnesses give four ways to keep a session out of the dumb zone. The first is short standing instructions: `AGENTS.md` is in the context on every call of every session, so it holds only what the agent cannot work out for itself (Section 4.2), and MCP servers not needed for the project are switched off.

The second is subagents. A *subagent* is a second agent that the first one starts for a subtask; it “runs in its own context window”, does the reading there, and “returns only the summary” [Ant26g]. Forty pages read by a subagent to find where a paper defines X come back as one line, and never enter the main context. Asking “use a subagent to find where the paper defines X ” is enough.

The third is a fresh session. When a session gets long, the agent writes what matters (the goal, what is done, and what is next) to a file; a new session starts empty and reads the file. Claude Code’s `/clear` empties the context, and its documentation recommends it “when switching to unrelated work” [Ant26i].

The fourth is *compaction*: the harness replaces the conversation by a summary of it. Claude Code compacts automatically as the window fills, and `/compact` does it on request, optionally with instructions for what the summary should keep (`/compact focus on the proof of Lemma 3`). After compaction the project’s instruction files and skill descriptions are reloaded from disk, up to five recently read or edited files are re-read, and invoked skills are re-injected; everything else survives only as the summary [Ant26i]. A handoff file written by the agent, by contrast, says what you asked it to say. The command `/context` shows how full the window is and what is in it [Ant26i].

4. STANDARDS: SKILLS, AGENTS.MD, AND MCP

Three things you write or install yourself sit between you and the model: an instruction file that the agent reads at the start of every session, skills that tell it how to do particular tasks, and servers that give it new tools. All three are open standards, so they are not tied to one vendor or one harness. Most of them are kept by the *Agentic AI Foundation* (AAIF), a part of the Linux Foundation founded on 9 December 2025 with three projects: the Model Context Protocol (MCP) from Anthropic, goose from Block, and `AGENTS.md` from OpenAI [The25, Ant25a]. On 22 October 2026 the foundation added four more, among them Agent Skills, contributed by Anthropic, and it now hosts ten projects [The26a, Age26c]. The three a mathematician needs are the subject of this section: skills (Section 4.1), `AGENTS.md` (Section 4.2), and MCP (Section 4.3).

4.1. Skills and prompts. A *skill* is a folder containing a file `SKILL.md`: a name, a one-line description of when to use it, and instructions in English, sometimes with scripts and reference files beside them. The harness keeps only the names and descriptions in view and reads a skill’s full instructions when a task matches; Anthropic calls this *progressive disclosure* [Ant25b, Ant26j], and it is why you can install hundreds of skills without filling the context window. Anthropic introduced skills in October 2025 and published the format as an open standard at [Age26a] `agentskills.io` in December 2025 [Ant25b, Age26b]. Codex, Gemini CLI, Cursor, VS Code, GitHub, OpenCode, and others read the same folders [Wil25, Ope26b], so a skill written once works in each of them. You run a skill by name (`/name` in Claude Code, `$name` in Codex), or the agent picks it up when its description matches what you asked. To install a skill someone has published, point your agent at it and ask it to install it.

A skill is a prompt, but it is a prompt that follows the rules of ordinary programming. It has inputs and a contract (what it promises to produce and what it refuses to do). It calls other skills the way a function calls functions. It can be tested, by running it on fixed cases and checking the output, and it is kept under version control, so a change that makes it worse can be found and undone. The smallest example of composition is Matt Pocock’s `grill-with-docs`, whose whole body is one sentence: “Call the Skill tool twice, for ‘grilling’ and ‘domain-modeling’” [Poc26]. Larger skills route work: a *router* skill reads the request and hands it to one of a family of smaller skills, so you only need to remember one name per area. Every skill mentioned in this guide, with a public link, is in the catalogue of Section B.

Claude Code looks for skills in `.claude/skills` and Codex in `.agents/skills`, in your home folder for every project and in the repository for one project (Table 5). Keep one folder and make the other a *symbolic link*, a file that points to a folder, so that both harnesses see the same skills:

```
$ mkdir -p ~/.agents/skills ~/.claude
$ ln -s ~/.agents/skills ~/.claude/skills
```

On my machine both global folders point to one git repository of skills, so my skills are versioned like code.

4.1.1. *Superpowers*. The set of skills to start with is Jesse Vincent’s *superpowers* [Vin25] `obra/superpowers`. It makes the agent brainstorm with you before building, write a plan, work test-first, debug systematically before fixing, and verify before claiming success. Its README calls it “a complete software development methodology for your coding agents, built on top of a set of composable skills and some initial instructions that make sure your agent uses them.” My own skill packages are built on it. In Claude Code you install it with

```
> /plugin install superpowers@claude-plugins-official
```

and in Codex you type `/plugins`, search for `superpowers`, and install it; install it separately in each harness you use. Its skills trigger on their own, and you can also begin a planning or debugging task with `/using-superpowers`, which makes the agent look for the relevant skill and follow it. Superpowers was written for programmers; the skills for mathematics that grew out of it are in Section B.

4.1.2. *Planning: grill-with-docs*. `grill-with-docs` [Poc26] `mattpocock/skills/.../grill-with-docs` reverses the usual roles: you describe a plan and the agent interviews you, one question at a time, each with a recommended answer, until every branch is settled. As it goes it writes the decisions and a glossary of terms to files, so the interview becomes the project’s documentation. I use it before any project with more than one moving part: a paper, a lecture, a computation, and this guide. In the session in Figure 15, one question has just been settled, glossary terms have been written to a file, and the next question arrives with a recommended answer and alternatives. The usual pattern is that the agent spots a problem you had not thought about, proposes a default, and you make the call.

```

... cleaner — python · claude
... 2026 — rochester — python · claude
~/git — QUMBY 79 — python3 · claude
... endation letter LaTeX — python · claude
... cf — all-char-worker — python · claude
... cf — tauceti-worker — python · claude
+

● Agent "Orchestration/repo slides" is waiting on its own background work · 14m 30s
● I'm still looking through your files for the CSAI talk and AI notes to grab the dumb-zone chart and its citation—the skills section just came back. Next, I have a question about the ChatGPT "Turing machine" slides (1.6-1.7).

● User answered Claude's questions:
  - For the math slide, how precisely do you want to say "it's a Markov process"? The state is the entire token string, truncated to the context window, so the chain lives on  $V^{\leq N}$  and not on  $V$ . Do you want that subtlety on the slide?
  - State = whole context (Recommended)
  - Message queued for delivery to a680f9a1daed972ab at its next tool round. If it finishes first, it is resumed with the message and notifies again, so wait for that notification before resending.
  - Allowed by auto mode classifier

● Write(CONTEXT.md)
  - Wrote 13 lines to CONTEXT.md
    1 # Glossary — "How to use AI in Mathematical Research Effectively"
    2
    3 ==Token== — an element of the model's vocabulary  $V$ ; text is cut into tokens before the model sees it.
    4
    5 ==Context== — the string of tokens  $x \in V^{\leq N}$  the model conditions on;  $N$  is the context window.
    6
    7 ==Language model== — a map  $M: V^{\leq N} \rightarrow \Delta(V)$  assigning a probability  $M(x)(y) = P(xy | x)$  to each next token  $y$ .
    8
    9 ==Generation== — the Markov chain on  $V^{\leq N}$  whose state is the whole context and whose transition is  $x = xy$  with probability  $M(x)(y)$ .
    ... +3 lines (ctrl+o to expand)

● I've started CONTEXT.md with the terms settled so far. Next question, from what the skills agent found:

> Use the link that I gave you and /read-chatgpt
> Message from @general-purpose (ctrl+o to expand)
> Message from @general-purpose (ctrl+o to expand)

[ pdf paper ]

| For the AI-statement slide (3.9), the pdf paper is an unpublished draft in a private repo, and the rendered pages show {taylor(...)} TODO notes. Your coauthors include Rochester faculty who may be in the room. What should the slide show?

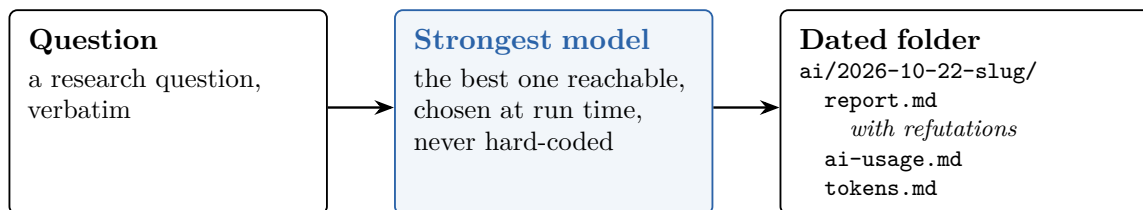
1. Clean summary graphic (Recommended)
  A rendered summary of what the statement reports (models, harness, what each was used for, verification, cost). No draft pages on screen.
2. Actual paper pages
  Show the rendered pages 3-4 of the AI section; I'll recompile with TODO notes hidden first.
3. Both
  Summary graphic, then a click to the real (cleaned) pages.
4. We have permission for this. I want to talk about what I have been working on with Tom and Zhuoran. (By the way, I took a screenshot of this slide — use it when we talk about "grill-with-docs". One slide, just this screenshot.)
5. Chat about this

Enter to select · ?/4 to navigate · ctrl+g to edit in Vim · Esc to cancel
rochester

```

FIGURE 15. A grill-with-docs session in Claude Code.

4.1.3. *Asking the strongest model: frontier-dump.* The simplest thing you can do is hand the strongest model you can reach a question, let it think, and record what it says. That is all **frontier-dump** does, and it is the skill I use most. It sends the question, verbatim, to a fresh agent running the strongest model the harness can reach on the day it runs, and saves a dated folder with a report, an AI-usage record, and a token record. The skill never names a model; its own text says that “frontier” is “decided at run time, not fixed at write time.” What it does fix is the bookkeeping. The report separates proved, conditional, computational, and conjectural claims, and it ends with a section of “refutations and corrected expectations”: every claim in the prompt or the input material that turned out to be false. It also records which model answered and what it cost, which you need for an AI statement (Section 7).



I use it in two ways. I ask both ways: `/frontier-dump "prove X"`, and then, in a separate run, `/frontier-dump "disprove X"`, because a model asked to prove something leans toward proving it and the second run checks the first. And I hand it the question and the sources, not a script. In my experience a detailed prompt written by a cheaper model does worse: it over-specifies, adds framing that is not true (“another agent has already solved this”), and keeps the stronger model from choosing its own approach. An exercise that runs the skill both ways on a lemma you know is in Section A.

The reason the skill names no model is a conjecture from Rich Sutton’s essay “The Bitter Lesson”: general methods that use more computation eventually overtake specific, hand-engineered ones, because computation keeps getting cheaper [Sut19]. Sutton’s examples are chess, Go, speech recognition, and vision. The International Mathematical Olympiad is consistent with it. In 2024

Google DeepMind’s specialized systems, AlphaProof and AlphaGeometry 2, needed the problems translated into a formal language by hand, ran for two to three days, and scored 28 of 42, a silver medal; in 2025 a general model, an advanced version of Gemini Deep Think, read the problems in natural language within the contest time and scored 35 of 42, a gold medal [Goo25]. (Deep Think had extra training on mathematics and some IMO-specific hints in its instructions, and no compute figures were published.) AlphaProof is the clearest case of a specialized tool: an agent modelled on AlphaZero that learns to find proofs in Lean by reinforcement learning [HMS⁺26]. Google invested in tools of this kind, and in 2026 its general models have lagged. On the Artificial Analysis Intelligence Index in late October 2026, Claude Opus 5.5 scores 58 and GPT-6 Astra and Gemini 4 Argon both score 53 [Art26b]; on Epoch AI’s Capabilities Index Gemini 4 Argon ties Claude Fable 5.1 for fourth, behind Claude Opus 5.5, GPT-6 Astra, and GPT-6.1 Sol [Epo26a]. This is consistent with the conjecture; it does not prove it.

First Proof (Section 6.8) gives a current test. In its third batch, in October 2026, five systems had 24 hours each on ten unpublished research problems [Fir26d]. System A was GPT-6 Astra with a short, plain prompt asking for “a complete and rigorous proof” [Fir26c]; it solved 7 of 10 for \$36. The other four were harnesses built around one or more models: ProofCouncil, from the IMProofBench team, which runs several models including GPT-6 Astra and solved 9 of 10 for \$1,888; Siege, from Anthropic; MerLean; and Humanize, built on the open-weight Kimi K3, whose every submission had major errors [Fir26d, Fir26b]. No Google DeepMind system was entered. The harness won this round by two problems at about fifty times the cost; the conjecture predicts that the gap closes as models improve. Everything is public: the organization github.com/1stproof has one repository per batch, with the problems, the human solutions, each system’s submitted code and prompts, the raw outputs, the referee reports, and the costs [Fir26a], and DeepMind’s outputs (solutions to six problems) for Aletheia on the first batch are in [Goo26e] [google-deepmind/superhuman](https://github.com/google-deepmind/superhuman). Reading the prompts next to the scores is the quickest way to see what a harness adds.

The practical test I take from this, for any skill or habit: does it spend little context, and would it matter more, not less, if the model were ten times better? **frontier-dump** passes it.

4.1.4. *Fixed-point review.* Review works best when it is done by a separate agent, with a fresh context and an explicitly hostile brief. My **critical-review** skill tells the agent that its role is “hostile, rigorous critic” and “you are not here to encourage or balance.” Before judging the logic it checks that every file and tool the artifact mentions exists, and its most useful check is *claim provenance*: for each claim the conclusion rests on, what produced it, and does that output actually support it? The verdict is APPROVING or NEEDS-REVISION, with a ranked list of fixes.

Review once and you get a list. Review repeatedly and you get a *fixed-point loop* (Figure 16): one agent reviews the artifact x_n , a second revises it into x_{n+1} to answer the review, and you repeat until the reviewer comes back clean. My skills for this are **fp-finder-skill** for skill files and **fp-finder-latex** for papers, where each round is an independent simulated referee report followed by a revision.

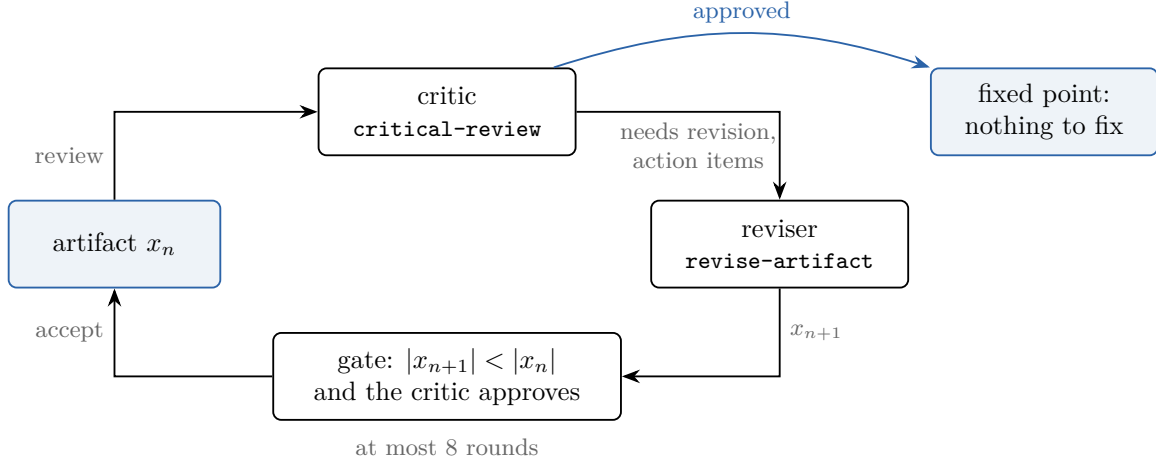


FIGURE 16. The fixed-point loop.

Left alone, the loop does not converge: every round’s easiest fix is to add a sentence, so the artifact grows. The skill’s own description says it was designed “after generic FP loops on SKILL.md files grew unboundedly without converging.” The repair is to make the loop a contraction. A revision x_{n+1} is accepted only if it is strictly shorter than x_n and an independent critic still approves it. Length is a non-negative integer that strictly decreases along accepted revisions, so the loop terminates, with a hard cap of eight rounds as a backstop; the approval condition blocks the obvious cheat, since deleting everything is shorter but the critic rejects it. The loop stops when the critic approves and the reviser, asked once more, cannot make a material improvement. The manuscript version labels its result a simulated review, not a guarantee of acceptance.

4.1.5. *Skills that maintain themselves.* Skills decay the way code does. Every patch adds a sentence, rules start to contradict each other, and the instructions drift away from what the agent does. Long rule lists also cost accuracy, since models follow fewer instructions correctly as the list grows (Figure 13). So the maintenance is built into the skills: the fixed-point loop above, applied to skill files, only accepts revisions that shrink them, and **adjust-behavior** repairs misbehavior without adding rules.

adjust-behavior treats a behavior you don’t like as a bug in the instructions the agent read, and debugs it in four steps (Figure 17). First it reproduces the behavior on demand (“no repro, no repair”). Then it finds the cause: which skill, **AGENTS.md**, or prompt produced it. Then it repairs that text, asking “can this be fixed by giving the agent better context rather than by adding a rule? Default yes”; the file must not grow without a stated reason, and an adversarial review checks the edit against everyone else who reads that file. Finally it verifies, by rerunning the reproduction on the text that will actually ship (“only repaired ships”). When two causes both explain the behavior, it escalates to a stronger model instead of patching both.

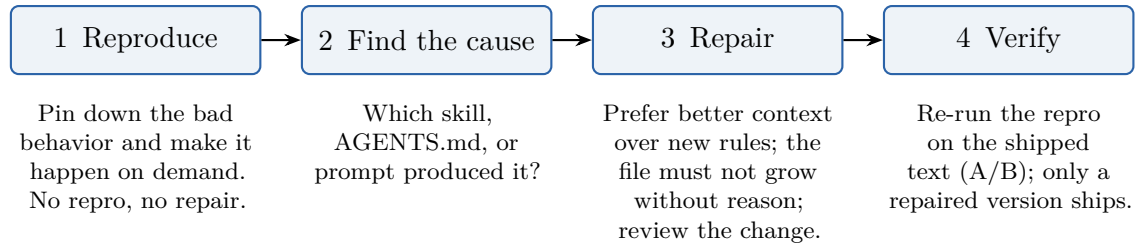


FIGURE 17. The four steps of `adjust-behavior`.

Two smaller skills keep work from multiplying. `check-wheel` makes the agent state a proposal in one sentence and search the places it might already exist (your repositories, existing skills, Magma, the LMFDB, the Stacks Project, and the Python and Sage ecosystems) before writing anything. `xkcd-927`, named after Randall Munroe’s comic about competing standards [Mun11], stops an agent from resolving two disagreeing documents by writing a third: it consolidates into one existing document and marks the others superseded. Finally, whenever an agent must do something exactly the same way twice, a skill should have it write a program and run it, rather than follow a paragraph of prose; programs fix the format, and prose is followed a little differently each run.

4.2. AGENTS.md and spec-driven development.

4.2.1. *Git*. *Git* records the history of a folder. Each snapshot, called a *commit*, saves every file with a short message saying what changed, and a folder whose history *git* keeps is a *repository*. You start one with `git init` and save a snapshot with `git add` and `git commit`; `git log` shows the history and `git diff` shows what changed since the last commit. With an agent, *git* is the undo button and the record of who did what, which you need for an AI statement (Section 7). GitHub is a website that hosts copies of repositories, privately or publicly, and is the easiest way to share a paper with coauthors and their agents.

Learn it. The agent can drive *git* for you while you learn: ask it to commit with a sensible message, make a branch for an experiment, explain the last three commits, or undo what it just did, and to show you each command it runs and why. Within a few weeks you will be reading `git log` and `git diff` yourself. When several agents work in one repository at once, each gets its own working copy through `git worktree` (Section 8). An exercise on letting the agent drive *git* is in Section A.

4.2.2. *A repository for each project*. I keep one repository per project (a paper, a set of lecture notes, a computation), and they all have the same layout. Here is the shape of the repository for the pcf paper [DLTW26], with the names made generic:

my-paper/

```

my-paper/
|-- AGENTS.md          entry point every agent reads first; points to the files below
|-- CLAUDE.md          one line, "See AGENTS.md", so Claude Code finds the same file
|-- LAYOUT.md          what goes where, and what is never tracked
|-- LATEX.md           which .tex and .bib files are the real ones
|-- STYLE.md           mathematical writing conventions
|-- ADR.md             decisions already made: dated, append-only
|-- AI-POLICY.md       what must be recorded and disclosed about AI use
|-- CONTEXT.md         glossary of the project's terms
|-- latex/             the manuscript: paper.tex, paper.bib, figures
|-- ai/               agent output, kept apart from the paper
  
```

```
| |-- 2026-09-20-referee-report/    one dated folder per agent task
| |-- 2026-09-21-mathlib-fit/
| |-- ai-usage.md    master record of what AI did
| `-- tokens.md     master record of tokens used and their cost
|-- refs/           PDFs of the papers cited (never tracked)
|-- scratch/        disposable experiments (never tracked)
|-- lean/           the Lean formalization
`-- .gitignore      keeps build output, refs/, and scratch/ out of git
```

The `ai/` folder is where agents put their reports, so that their output never mixes with the paper until I decide it should, and the two master records there are what the paper’s AI statement is written from (Section 7). You do not need all of this on the first day. Start with `AGENTS.md`, `latex/`, `ai/` with its two records, and the `.gitignore`, and add the other files when you first need them; an agent will draft each one with you if you ask, and my `init-repo-docs` skill sets up the whole set.

4.2.3. *AGENTS.md*. `AGENTS.md` is a plain Markdown file that the agent reads at the start of every session. The agents.md site calls it “a README for agents” [Age]. Codex reads it natively [Ope26e]; Claude Code reads `CLAUDE.md` [Ant26l], so a one-line `CLAUDE.md` that points to `AGENTS.md`, or the import line `@AGENTS.md`, makes both harnesses read the same instructions. Put in it what a new collaborator would need on the first day and the rules you never want broken, and keep it short, because it is read on every step. Mine point to the other files rather than repeat them. Here is one to start from:

my-paper/`AGENTS.md`

```
# AGENTS.md: agent entry point for my-paper

A paper on <topic> by <you>, a graduate student; advisor <name>.

## The repository's contracts
- LAYOUT.md: what goes where.
- LATEX.md: latex/paper.tex is the paper; build with `latexmk -pdf paper.tex` in
  latex/.
- STYLE.md (writing), ADR.md (decisions made), CONTEXT.md (terms),
  AI-POLICY.md (what to record about AI use).

## Rules
- Do not edit latex/ unless asked. Drafts and reports go in ai/YYYY-MM-DD-<task>/.
- Cite only sources you have opened (refs/, a URL, or a DOI), with the
  theorem number or page of each claim used. Mark anything unchecked UNVERIFIED.
- If a question has a false premise, say so. Say "I don't know" rather than guess.
- Save every computation as a script next to its output.
- After each task, add a dated entry to ai/ai-usage.md and ai/tokens.md.
- Commit with a clear message. Never push without asking.

## Terminology
- <rules for this project's words, added as they come up>
```

The last heading is where project conventions accumulate; in the `pcf` repository, for instance, it records that an integral closed subscheme of codimension one is called an “integral effective divisor”, never a “prime”. An exercise that has a fresh agent critique your `AGENTS.md` is in Section A.

Instruction files and skill folders come in three scopes (Table 5). *Global* files live in your home folder (written `~`) and apply to every project, so personal rules go there; *project* files are committed

to the repository, so collaborators and their agents get them too; *personal* files are yours, for one repository, and stay out of git.

	Claude Code	Codex
Global (every project)	~/.claude/CLAUDE.md, ~/.claude/skills/	~/.codex/AGENTS.md, ~/.agents/skills/
Project (committed)	./AGENTS.md or ./CLAUDE.md, .claude/skills/	./AGENTS.md, .agents/skills/
Personal (one repository)	./CLAUDE.local.md, .claude/settings.local.json	AGENTS.override.md

TABLE 5. Where Claude Code and Codex look for instructions and skills [Ant26l, Ant26o, Ant26j, Ope26e, Ope26b].

4.2.4. *Spec-driven development.* In *spec-driven development* you write the specification first (what is wanted, the plan, and pass/fail tests for success), and agents implement against it and are checked against it. Mathematicians already work this way: the definitions and the statement come first, and a proof is correct relative to them. Table 6 is the dictionary.

Mathematics	Spec-driven development
definitions	the specification
statement of the theorem	acceptance tests
proof	implementation
referee	review gates and check-* skills

TABLE 6. Spec-driven development and mathematics.

GitHub’s open-source Spec Kit [Git25] `github/spec-kit` turns this into a workflow with one command per step: a *constitution* (the project’s principles, written once), then *specify*, *plan*, *tasks*, and *implement* [Del25]. In my repositories the constitution is the set of files in the tree above. Each `*.md` contract states rules with a mechanical pass/fail criterion, and a read-only check skill audits the repository against it and reports violations. The simplest pair is `LAYOUT.md`, a declared tree of where every kind of file belongs, audited by `check-layout`. Without one, agents scatter files, and a repository drifts until nobody knows which file is the real one.

4.3. **MCP.** Skills tell an agent how to do something; MCP servers give it new abilities. The Model Context Protocol [Mod26] is a standard way for an agent to call outside programs and data as typed tools. An MCP *server* is a small program that exposes a list of tools, each with a name, a description, and typed arguments; the agent calls them like functions and gets structured data back, instead of scraping web pages (Figure 18). Anthropic released MCP in November 2024 [Ant24], ChatGPT, Cursor, Gemini, Microsoft Copilot, and Visual Studio Code adopted it, and in December 2025 Anthropic gave it to the AAIF [Ant25a]. If a resource you use has an MCP server, it is usually the better way for the agent to reach it, and you can ask your agent to install one for you.

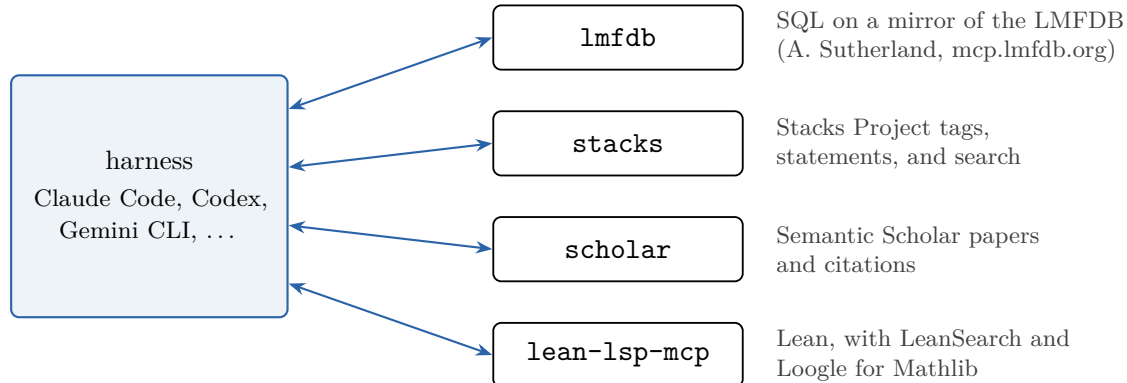


FIGURE 18. An agent connected to MCP servers.

The servers I use for mathematics are these. Andrew Sutherland’s `lmfdb` server gives read-only SQL access to a mirror of the LMFDB [Thea], at <https://mcp.lmfdb.org/mcp> [Sut26] AndrewVSutherland/lmfdb-mcp; its tools include `overview`, `describe_table`, `search_knows`, `run_sql`, and `export_query` for bulk downloads. `lean-lsp-mcp` [oOo26] oOo0o0o/lean-lsp-mcp connects the agent to Lean’s language server, with `LeanSearch` (search in natural language) and `Loogle` (search by type pattern) for Mathlib. `arxiv-mcp-server` [bla26] blazickjp/arxiv-mcp-server searches arXiv and reads papers. The `scholar` server (Semantic Scholar papers and citations) and the `stacks` server (Stacks Project tags, statements, and search) are in my public skills repository [Dup26a].

You do not write the SQL. Asked for the rank-2 elliptic curve over $\mathbb{Q}$ of smallest conductor, the agent finds the table with `overview` and `describe_table` and calls `run_sql`; the call and the server’s reply are the listings in Section 3.1. The answer, `389.a1`, comes from the database, not from the model’s memory, and Exercise A.13 has you repeat it.

5. MODELS, PRICES, AND PLANS

This section is about what agents cost: how usage is measured, how models compare on price and on benchmarks, where tokens are sold, and what a subscription plan buys compared with paying per token. Prices, scores, and limits change every few weeks, so every number here carries a date. I re-checked the vendors’ pricing and usage-limit pages on 25 October 2026.

5.1. Tokens. Model usage is measured in tokens, the pieces of text a model reads and writes (Section 2.2 explains how text is cut into tokens). Every step of an agent consumes *input* tokens, the text the model reads, and *output* tokens, the text it writes. Output costs five times as much (six for GPT-5.6 Terra) as input at both Anthropic and OpenAI. A model’s hidden “thinking” is billed as output, so a higher effort setting costs more per task; Anthropic and Google both say so in their documentation, and Anthropic adds that its current models always think [Ant26m, Goo26b].

Most of the tokens in an agent session are input. On every step the model is sent its whole context again, so a long session reads the same files over and over, and its cost grows roughly with the square of its length [Ald26]. Providers keep recently read text in a *cache*. Writing text into the cache costs a little more than ordinary input, and reading it back costs much less: a tenth of the input price for GPT-6 Astra and for Claude Fable 5, and a fortieth for Claude Fable 5.1 [Ant26n, Ope26h]. Re-reads therefore dominate the token count more than they dominate the bill. In the run of Table 10, 91% of the tokens were re-reads and about 1% were text the model wrote, while the bill was about 44% for reading new material, 34% for re-reads, and 23% for writing. Anthropic’s documentation puts it this way: “a one-line question in a session that has been open all day still

draws usage for the whole conversation” [Ant26m]. Short, focused sessions (Section 3.7) are the main way to spend less.

5.2. Ladders of models and their list prices. Every lab sells a ladder of models, from small and cheap to large and expensive. Anthropic’s is Haiku, Sonnet, Opus, and Fable; OpenAI’s is Luna, Terra, Sol, and Astra; Google’s flagship is Gemini 4 Argon. Version numbers drift independently within a ladder (in October 2026 OpenAI’s newest Terra is GPT-5.6 while Sol is at GPT-6.1), so nothing should be read into them. Prices are quoted in dollars per million tokens (Figure 19). The spread within a ladder is a factor of 100: Claude Haiku 5.5 and GPT-6 Luna cost 10 cents per million input tokens, and Claude Fable 5.1 and GPT-6 Astra cost 10 dollars [Ant26n, Ope26h]. The two labs have converged on almost identical prices. Gemini 4 Argon costs \$2 per million input tokens and \$10 per million output tokens through 30 November 2026, and twice that from 1 December [Goo26b]. Prices per unit of capability have been falling fast [CSOA25].

AI Anthropic			OpenAI		
Model	Input	Output	Model	Input	Output
Claude Haiku 5.5	\$0.10	\$0.50	GPT-6 Luna	\$0.10	\$0.50
Claude Sonnet 5.5	\$2	\$10	GPT-5.6 Terra	\$2	\$12
Claude Opus 5.5	\$4	\$20	GPT-6.1 Sol	\$2	\$10
Claude Fable 5.1	\$10	\$50	GPT-6 Astra	\$10	\$50

FIGURE 19. API list prices in US dollars per million input/output tokens, 22 October 2026 [Ant26n, Ope26h].

Different tasks call for different models. Moving a file from one folder to another is a job for the cheapest, fastest model, such as Claude Haiku 5.5. Hunting for an exotic Tate class on an abelian variety is a question at the research frontier, for the most capable model available, such as GPT-6 Astra, with everything checked afterwards. Most of what agents do (reading, searching, reformatting, and bookkeeping) does not need the frontier model. When my agents start subagents (Section 3.7), I tell them to pick the cheapest model that can do each job.

5.3. Benchmarks. Figure 20 plots sixteen models by one composite score, the Artificial Analysis Intelligence Index (v4.3.2, each model at its best effort setting), against the blended price $(3 \cdot \text{input} + \text{output})/4$ in dollars per million tokens, on a log scale [Art26b]. Moving right, price grows by a factor of about 100 while the score grows by about 30%. The frontier models, Claude Fable 5.1 and GPT-6 Astra, sit at the far right. GPT-6.1 Sol and Gemini 4 Argon reach almost the same score at a fifth of the price, Claude Sonnet 5.5 scores slightly higher at a fifth of the price, and Claude Opus 5.5 scores higher still at under half the price. Claude Haiku 5.5 and DeepSeek V4.1 Flash reach 80–85% of the frontier score at 1–3% of the price. Price per token is not price per task: at higher effort settings a model thinks longer and uses more tokens.



FIGURE 20. Intelligence Index against price, 22 October 2026 [Art26b].

On 25 October 2026 the live leaderboard put Claude Opus 5.5 first at 58 and Claude Sonnet 5.5 second at 56, with Claude Fable 5.1, GPT-6 Astra, and Gemini 4 Argon at 53; it also listed two models that are not in the figure, Meta’s Muse Wonder Full at 56 and Kimi K3.1 Pro at 55 [Art26b]. Epoch AI’s Capabilities Index, updated the same day, ranks Claude Opus 5.5 (168.8), GPT-6 Astra (168.8), GPT-6.1 Sol (168.4), Claude Fable 5.1 (166.5), and Gemini 4 Argon (166.5) at the top, with overlapping confidence intervals [Epo26a]. On Epoch’s index the top places in October 2026 belong to Anthropic and OpenAI models, and Google’s flagship has lagged them (Section 4.1.3).

There are dozens of benchmarks (OpenRouter alone runs 18 [Ope26o]), and changing the benchmark changes the ranking. In the same Artificial Analysis snapshot of 22 October, on AA-Omniscience, which rewards knowing facts and penalizes confidently wrong answers, DeepSeek V4.1 Flash and GPT-5.6 Terra score about zero and the small open Gemma 4 31B scores well below zero. On CritPt, a benchmark of research-level physics problems, the top seven models are bunched between 0.27 and 0.32 of problems solved [Art26b]. No single number says which model will do best on a given mathematical problem; finding out means trying the problem on more than one model.

5.4. Token markets. Underneath the chat windows there is a market in tokens. OpenRouter is a broker that resells almost every model through one account and ranks models by the tokens it processes for them [Ope26n]. In late October 2026 it handled roughly 150–200 trillion tokens a week, and the most used model was a cheap open-weight one, DeepSeek V4.1 Flash. In that market people pay per token and choose a model the way they would choose a cloud server.

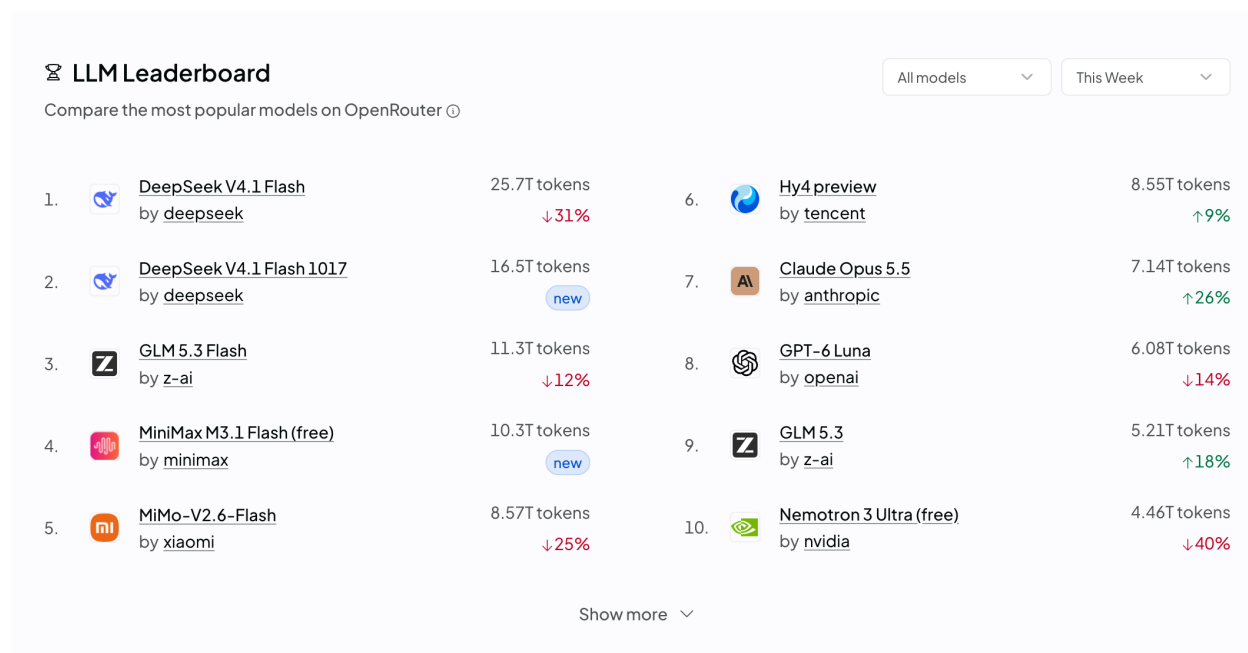


FIGURE 21. OpenRouter’s leaderboard by weekly tokens, 22 October 2026 [Ope26n].

So there are three ways to get tokens. The first is to buy them from a frontier lab (Anthropic, OpenAI, or Google), by subscription or per token. The second is to buy tokens of open-weight models served on someone else’s machines, through a broker such as OpenRouter. The third is to run an open model yourself, on a laptop or a department machine, with a program such as Ollama [Oll26i]. The open-weight models are behind the frontier on the indices above, the commercial harnesses are tuned for their own labs’ models [Ope26p], and anything much larger than a small model needs serious hardware to run at home; Section 2.8 covers what the second and third routes involve. The rest of this section is about the first.

5.5. Plans and API rates. A frontier lab sells tokens in two ways. Through the *API* (application programming interface, the door that programs use to call a model) you pay the list prices of Figure 19, exactly, for what you use. A *subscription*, or plan, is a flat monthly fee for a stream of tokens, regulated by a limit over each 5-hour window and a weekly limit, both of which the lab can change. A subscription is a token subscription: you pay a fixed price in advance for the right to use tokens later, up to the limits. Table 7 lists the plans.

Plan	Price per month	What you get
Claude Pro	\$20 (\$17 billed yearly)	Claude Code; at least 5 times Free’s usage per 5-hour session; Fable paid from usage credits
Claude Max	\$100 or \$200	5 or 20 times Pro’s usage per 5-hour session, a weekly limit across all models, and \$100–\$200 a month of API credits
ChatGPT Free and Go	\$0 and \$8	Codex with GPT-6 Luna only
ChatGPT Plus	\$20	Codex; per 5-hour window an estimated 5–45 messages with GPT-6 Astra or 15–160 with GPT-6.1 Sol
ChatGPT Pro	\$100, \$200, or \$500	No five-hour limit at present; the \$500 tier adds an “Ultrafast” mode
Google AI Pro	\$19.99	Entry rate limits in the Antigravity agent
Google AI Ultra	\$99.99 or \$199.99	Higher or highest rate limits in Antigravity
Claude Team	\$25 or \$125 per seat (\$20 or \$100 billed yearly)	Shared billing and administration for 2–150 people; Claude Code included
Claude Enterprise	\$20 per seat (yearly, 20 seats minimum) <i>plus usage at API rates</i>	Administration; usage billed per token
ChatGPT Business	\$25 per user (\$20 billed yearly)	The same Codex limits as Plus
Education plans	ask the vendor	University-wide agreements

TABLE 7. Subscription plans, 25 October 2026.

The table is taken from the vendors’ pages on 25 October 2026 [Ant26d, Ant26r, Ope26i, Goo26c]. OpenAI’s per-window message counts are its own estimates, and it adds that “weekly limits may also apply” [Ope26i]. Google describes its limits as refreshing “every 5 hours until you reach your weekly limit”, with Ultra at “5x or 20x higher than AI Pro limits” [Goo26a].

Buying a \$200 plan does not give you access to models or tools that can do more; it gives you more usage of the same ones. The higher tiers of each lab raise the limits. What else they add is about speed and billing, not capability: OpenAI’s \$500 tier adds the Ultrafast mode in Codex [Ope26i]; Max comes with monthly API credits; and the frontier model, Fable, is paid for from usage credits on Claude Pro, while Anthropic’s pricing page lists it on Max as “50% of weekly limits” [Ant26d]. Free plans do not include Claude Code [Ant26d].

For someone starting out, one \$20 plan (Claude Pro or ChatGPT Plus) used for real work for two weeks shows whether the limits are a problem; the \$100 tier is the next step if they are. A second plan from the other lab gives a second, independent model family, which is useful for checking the first.

Departments that buy Team or Enterprise accounts get administration and pooled billing, not more capable models. An Enterprise seat price does not include usage, which is billed at API rates on top [Ant26d], so the cost depends on how much people use it. Commercial plans are not used for training by default [Antb], which matters for unpublished work (Section 3.4).

What a plan is worth. At the moment a subscription buys far more tokens than the same money spent at API rates. Table 8 values one \$200 plan by the run whose ledger is in Table 9: one evening’s work that would have cost \$257.81 at API list prices used 10% of the plan’s weekly allowance.

The run at Fable 5 API list prices (Table 9)	\$257.81
It used 10% of the weekly allowance, so one week of the plan is worth	$10 \times \$257.81 = \$2,578.10$
A month is 52/12 weeks, so one month of the plan is worth	$\frac{52}{12} \times \$2,578.10 \approx \$11,172$
The plan costs [Ant26r]	\$200 a month
Ratio	$\$11,172/\$200 \approx 56$

TABLE 8. What a Max 20x plan is worth at API list prices.

The ratio is an order of magnitude, not a price. Anthropic does not publish its limits in tokens. It says only that Max 20x gives “20x the Pro plan’s usage allowance”, with a session limit that resets every five hours, and that Max plans “also have a weekly usage limit that applies across all models” [Ant26r]. My 10% is a reading off the usage meter, assumed to be the weekly one. At the newer Fable 5.1 prices, which charge \$0.25 instead of \$1 per million re-read tokens [Ant26n], the same run would cost about \$193, and the ratio would be about 42.

Other measurements give smaller ratios. Wokey, a gateway that resells usage from subscription accounts behind it, measured 17 Max 20x accounts in the week of 12–19 October 2026 and put a week of Max 20x at about \$1,643 at API prices (about \$7,121 a month, roughly 36 times the price). It also found that the weekly limit of Max 20x is about 9.7 times Pro’s, not 20 times, because the “20 times” refers to the 5-hour window [Wok26]. So the plan is worth somewhere between 30 and 60 times its price at API rates, on these two measurements.

The limits change often. Anthropic introduced weekly limits for Pro and Max on 28 August 2025 [Zef25]. In 2026 it ran a temporary 50% increase in Claude Code’s weekly limits through 13 September, and from 14 September it raised the standard weekly limits by 25% “permanently”, which it described as a 17% reduction from the boosted level [Par26].

Measuring your usage, and the API-key trap. In Claude Code, `/usage` shows subscribers how much of the 5-hour and weekly allowance they have used, with a breakdown of what used it (skills, subagents, and MCP servers); its dollar figure is computed locally at list prices and is meant for API users [Ant26m]. The program `ccusage` reads the session logs on your disk and prices them at list rates, for Claude Code, Codex, and Gemini CLI [ccu26] `ccusage/ccusage`:

```
| $ npx ccusage
```

Those dollar figures are what the tokens would have cost through the API, not what you paid. Exercise A.15 prices one of your own sessions this way.

If the environment variable `ANTHROPIC_API_KEY` is set in your shell, Claude Code ranks it above your subscription login: in an interactive session it asks once whether to use the key and remembers the answer, and in non-interactive mode (`claude -p`) it always uses the key [Ant26b]. Usage through the key is billed per token, and users have reported surprise charges this way [Cla26a, Cla26b]. `/status` shows which credential a session is using, and unsetting the variable falls back to the subscription:

```
| $ unset ANTHROPIC_API_KEY
```

Reimbursement. Grants offices ask whether a plan is a subscription to general-purpose software. The argument I use is that the free tier is the general-purpose product, while a paid plan buys computing for research, metered in tokens, like time on a computing cluster. Only the part used for the project’s work is charged to a grant, and the harness logs every request, so the records show what the tokens were spent on (Section 7.2). Rules differ between institutions and funders, so the grants office is the place to ask, before buying.

5.6. A ledger: the false-premise experiment. The numbers above come from one measured run. After a question from Ivan Cheltsov, on the evening of 23 August 2026 I ran Claude Fable 5 agents for about an hour and a half. First an agent read his preprint and six related works in birational geometry. Then, in four separate sessions, it was told, falsely, that “a Fable agent” had already proved an open conjecture, and asked to reconstruct the proof. All four sessions declined and said the conjecture was open, and two of them also found an error in the background I had supplied (Exercise A.3 runs the same test in your own field). The run was paid for by a \$200-a-month subscription. Table 9 is its ledger, computed from the session records on disk: *Calls* are requests to the model, *Written* and *Read* are output and input tokens, and costs are at Claude Fable 5 API list prices.

	Task	Calls	Written	Read	Cost	Outcome
0	Read the preprint and six related works (main agent plus 7 reader subagents)	275	0.33M	39.2M	\$90.74	Reading notes for the later steps
1	“Reconstruct the proof” of Mumford’s conjecture	40	0.12M	4.8M	\$19.42	Declined; said it is open
2	... of the Iskovskikh–Manin/Kollár conjecture on hypersurfaces	43	0.10M	5.2M	\$18.96	Declined; found an error in the prompt’s background
3	... of “unirational iff a rational point” for geometrically rational surfaces (2 subagents)	209	0.18M	19.5M	\$45.39	Declined; found an error in the prompt’s background
4	... of unirationality of degree-1 del Pezzo surfaces (3 verification subagents)	209	0.44M	25.2M	\$83.29	Declined; said it is open
	Total	776	1.17M	94.0M	\$257.81	

TABLE 9. The false-premise experiment, 23 August 2026.

Table 10 splits the same 95 million tokens by kind, at Claude Fable 5 API list prices (the subagents’ cache writes are cheaper, at \$12.50 per million). Most of the tokens are the model re-reading its own context on every call; most of the cost is reading, of new material (44%) and of the cached context (34%), not writing (23%). Without the ledger the cost of a run on a subscription is invisible; with it, the same records also feed the AI statement of a paper (Section 7).

Kind of token	Tokens	Price per million	Cost
Written by the model (output)	1.17M	\$50.00	\$58.50
New text read (cache writes)	7.16M	\$12.50–\$20.00	\$112.47
Earlier context re-read (cache reads)	86.82M	\$1.00	\$86.82
Other input	0.002M	\$10.00	\$0.02
Total	95.16M		\$257.81

TABLE 10. The tokens of Table 9 by kind.

6. AUTOFORMALIZATION

Autoformalization means having agents translate a paper’s definitions, statements, and proofs into Lean, a language in which a computer checks every proof. I learned most of what is in this

section from conversations with David Roe about his work on the absolute Galois group of $\mathbb{Q}_2$, when I asked him what he had done and how. The problem, to find an explicit presentation of $G_{\mathbb{Q}_2}$, was contributed by Roe to Epoch AI’s FrontierMath list of open problems, and Epoch lists it as solved with AI: the first solution was elicited by Roe from Claude Fable 5 on 9 June 2026, and a second came from GPT-5.5 Pro in a harness built by David Turturean on 24 June [Epo26b]. Roe and Turturean wrote it up, Turturean writing the manuscript, and each of them formalized the proof in Lean with coding agents [RT26]. Roe’s formalization has contained no `sorry` since 26 July 2026 [Roe26]. Roe pointed me to Chris Birkbeck’s *mathlib-quality* [Bir26], a set of agent skills for developing, proving, cleaning up, and preparing Lean code for Mathlib. I adapted it into a package that behaves more like *superpowers*⁵: a single routing skill, `using-leanpowers`, that sends each request to the right small skill (Section 6.9).

6.1. What Lean is. Lean [Lea26d] is a programming language whose type checker checks proofs. A theorem in a Lean file has a name, a statement, and, after `:=`, a proof; if the proof is wrong, the file does not compile. *Mathlib* [Theb] is the large community library of formalized mathematics that almost every Lean project builds on. Two Lean words come up constantly: `sorry` is a placeholder that marks a step as unproven and lets the file compile anyway (with a warning), and an `axiom` is a statement assumed without proof. A finished formalization has no `sorry` and uses only the three standard axioms (`propext`, `Classical.choice`, and `Quot.sound`) [Ili26].

Roe’s formalization shows how a large formalization is organized. The proof of the main theorem is complete in Lean relative to nine *literature axioms*: named Lean statements of published results that Mathlib does not yet contain, such as local reciprocity for $\mathbb{Q}_2$ (`localReciprocity`), local Tate duality (`tateDualityAt`), and the local Euler–Poincaré characteristic. Each is documented with its citation and its deviations from the cited formulation [Roe26]. The same proof is also available with the nine results bundled into one explicit hypothesis, `LiteratureInputs`, instead of axioms. The census then shrinks over time: by the repository’s record, several entries were proved inside the repository in July 2026 (the dyadic Hilbert symbol formula and local Kummer theory, among others), and an extra input needed for a second candidate presentation was first carried as an explicit hypothesis in the theorem statement and then proved on 26 July [Roe26]. That is the order of work this section follows: first formalize relative to named major theorems, stated precisely and listed, and then reduce those theorems to more basic results.

6.2. Lean has two modes. In *interactive mode* you write the proof in an editor (VS Code with the Lean extension), and a side panel, the *infoview*, shows the current goal after each step (a *tactic*): the hypotheses above the turnstile $\vdash$ and what is left to prove after it. Figure 22 shows the last step of a proof that $a \mid b$ and $b \mid c$ imply $a \mid c$: unpacking the hypotheses gives k with $b = ak$ and m with $c = bm$, the witness km is proposed, and one rewrite closes the goal (“No goals”). Agents work in this mode too: they read the goal and propose the next tactic.

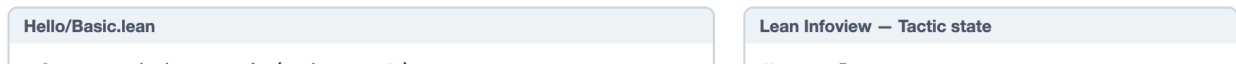


FIGURE 22. Interactive mode: the last step of a proof, with the infoview.

In *compile mode* the command `lake build` checks every file of the project in one batch, with no human in the loop. Either it reports `Build completed successfully` or it names the file, line, and goal it could not close. Figure 23 is the failure for the false statement $2 + 2 = 5$ (the file is `theorem two_add_two_eq_five : 2 + 2 = 5 := by norm_num`): the goal left over is `False`, at

⁵I heard about *superpowers* [Vin25] from Edgar Costa, who is perpetually up to date with what developers are using.

line 3. This is the mode a referee, or a *CI server* (a machine that rebuilds the project on every change), runs. Agents do the work in interactive mode; compile mode decides whether it is done.

```
$ lake build
...
× [9048/9050] Building Hello.Bad (69s)
error: Hello/Bad.lean:3:43: unsolved goals
└ False
Some required targets logged failures:
- Hello.Bad
error: build failed
```

FIGURE 23. Compile mode: `lake build` on a file claiming $2 + 2 = 5$.

6.3. An example: the pcf paper. My coauthors Alon Levy, Thomas Tucker, Zhuoran Wang, and I have a paper, *PCF all the way down* [DLTW26], whose main theorem is now formalized in Lean. This is how it went. On 20 September 2026 I asked an agent (Claude Opus 5 in Claude Code, running `using-leanpowers`) to formalize the paper, with Roe’s project as the model. The request, as the run’s record quotes it:

```
> formalize pcf.tex in lean ... a good model for to what extent it should be
    formalized is https://roed314.github.io/gq2/ where we assume a couple of Sorry's
    but get a formalization
```

The agent installed Lean and Mathlib, read the paper, and split it in two: the dynamical and combinatorial argument, and the geometric engine (the bounding theorem, proved from Zariski–Nagata purity and Abhyankar’s lemma). Mathlib had neither purity nor Abhyankar’s lemma, so the conclusion of the bounding theorem became a named interface, and the dynamical half was proved from it with no `sorry`. The agent fanned out four subagents, one file each, with instructions to delete and report anything they could not prove rather than weaken it, and wrote an axiom census, `AXIOMS.md`, following Roe’s pattern. Its report flags the point that matters most for Section 6.7: `#print axioms` showed only the standard three, but the assumed mathematics was carried in hypotheses and structure fields, which `#print axioms` cannot see.

The next four weeks reduced those interfaces. On 16 and 17 October agents proved Zariski–Nagata purity (Stacks Project Tag 0BMB) and Abhyankar’s lemma for a regular divisor (Tags 0EYG and 0EYH) in the development, and the characteristic-zero theorem became unconditional; the older theorems that take purity and Abhyankar’s lemma as hypotheses are kept beside it. On 21 and 22 October Codex and Claude Code sessions formalized the paper’s Theorem A in characteristic p (with prime-to- p monodromy) and for subvarieties of every codimension. The `lean/` folder now holds a root module `Pcf.lean`, the development under `Pcf/` (a subfolder `Stacks/` for the classical inputs, organized tag by tag, and one for the bounding theorem), `AXIOMS.md`, and the Palomar files described below. Each theorem is cited by the paper’s `LATEX \label`, never by number, because numbers change between versions.

Here is the build and the axiom check of the public copy (Section 6.4), run on 25 October 2026. The Mathlib build cache was already present, so `lake build` replays rather than recompiles. A build of the private repository from a clean clone on 16 October, after downloading the Mathlib cache, took under three minutes of wall-clock time.

```
$ cd lean
$ lake build
[7,307 lines of replayed messages omitted]
```

```
warning: PcfChallenge.lean:194:8: declaration uses `sorry`
warning: PcfChallenge.lean:200:8: declaration uses `sorry`
warning: PcfChallenge.lean:205:8: declaration uses `sorry`
Build completed successfully (4059 jobs).
```

The three `sorry`s are intended: `PcfChallenge.lean` is the statement file for Palomar, whose proofs live in another file. The axiom check is a three-line file outside the project:

`axioms.lean`

```
import Pcf
#print axioms Pcf.isPCFAllSubvarieties_charZero_of_downstairsPCF
#print axioms Pcf.isPCFAllSubvarieties_charP_of_downstairsPCF
#print axioms Pcf.Palomar.pcfAllTheWayDown_charZero_unconditional

$ lake env lean axioms.lean
'Pcf.isPCFAllSubvarieties_charZero_of_downstairsPCF' depends on axioms: [propext,
  Classical.choice, Quot.sound]
'Pcf.isPCFAllSubvarieties_charP_of_downstairsPCF' depends on axioms: [propext,
  Classical.choice, Quot.sound]
'Pcf.Palomar.pcfAllTheWayDown_charZero_unconditional' depends on axioms: [propext,
  Classical.choice, Quot.sound]
```

The paper says exactly what was and was not formalized, in its section on software and AI assistance (Section 7). The Lean statements are slightly more general than the paper’s: the variety is an integral, locally Noetherian scheme locally of finite type over the field and not assumed separated, and the map need not be a morphism over the field. The recursion is stated at every finite depth, and the fact that it stops after $\dim X$ steps is not stated. One part of a comparison proposition and some branch-locus inclusions of a lemma on towers are not formalized. Mathlib’s notions of codimension-one point, ramification index, and normalization were checked against the paper’s by agent review, not by a formal equivalence, and, as of the paper’s current version, no author has reviewed the formal statements [DLTW26]. The paper also names the models that wrote the Lean.

Figure 24 is the loop this followed: a natural-language proof, with its citations checked and an adversarial review, then a formalization, then the checks (build, axioms, fidelity). The arrow back matters. In the `pcf` project, for example, formalizing purity showed that one clause of Stacks Project Tag 0BMA, as stated, fails without a torsion-freeness hypothesis that its proof uses; I posted this as a comment on the tag, and the paper records the form that was proved [DLTW26].

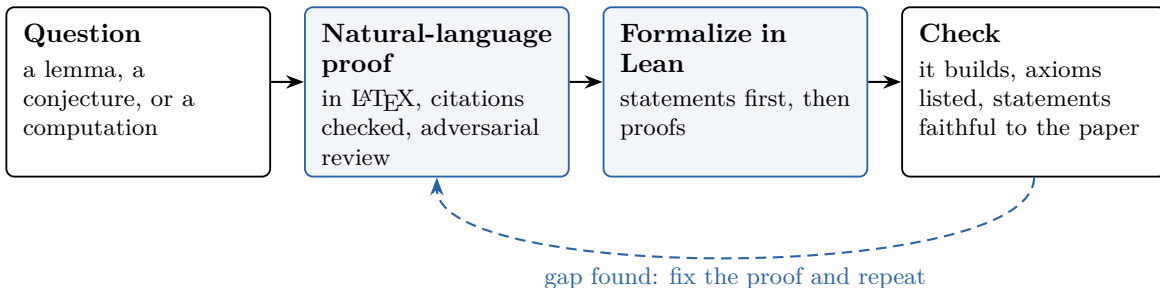


FIGURE 24. Prove in natural language, then formalize in Lean, and iterate.

6.4. A public repository for Palomar. The paper is not public yet, so the Lean code went into a separate public repository, [DTW26] `tdupu/pcfawd-formalization`, made for a submission to the

Palomar registry (Section 6.6). Palomar compares two files: a *Challenge* that states the theorems with Mathlib imports only and `sorry` as proofs, and a *Solution* that imports the development and proves them. A configuration file, `comparator.json`, names the compared theorems and the permitted axioms, and `formalization.yaml` carries the metadata: authors, a description of the statement, what it does not claim, and how AI was used. The README lists what is proved and gives the build commands. The commit history is the private repository’s history restricted to the published files, with commit messages edited only to remove private material, and with one squashed commit for the Theorem A work. The formalization was submitted on 17 October 2026; verification and review passed, but Palomar’s statement renderer rejected the project layout, which has since been changed, and it is not yet registered [DTW26].

6.5. Contributing to Tau Ceti. Tau Ceti [The26d] is a Lean library downstream of Mathlib whose code is written and reviewed by AI agents, while humans own the *roadmaps* [The26e]. It has three repositories: TauCeti for the AI-authored Lean, TauCetiRoadmap for the human-reviewed roadmaps, and TauCetiReview for the review rubrics and the machinery that runs them. A roadmap is a markdown file that specifies an area of mathematics, usually with suggested Lean signatures in a `Suggested.lean`. Tau Ceti’s instructions to agents allow a new definition or file only when it advances a specific roadmap target or supplies a prerequisite of one (improving existing code is always in scope), and agents never open pull requests or issues on TauCetiRoadmap themselves: people do. Code pull requests are reviewed by AI against the fixed rubrics, and the verdicts are posted as “block”, “changes requested”, or “approval” [The26d].

To work on part of a roadmap you register an *intention*, an issue on TauCetiRoadmap listing the targets you will take, and comment `claim` on it; a bot assigns it to you, and the claim expires after 30 days unless renewed. Automated workers do not author a target someone else has claimed [The26b]. Here is the claim on one of ours, issue #1456 [Dup26c]:

TauCetiRoadmap issue #1456, comments

```
tdupu: claim
github-actions[bot]: @tdupu you've claimed this task -- it expires **2026-11-22 21:39
UTC**.
```

Agents also follow a coordination contract in the TauCeti repository: they push to a pull-request branch only with `git push -force-with-lease` against the commit they observed, so no agent overwrites another’s work, and cooperating agents take short-lived claims on tasks to avoid duplicate work [The26c].

Converting an existing formalization follows the roadmap guide’s “porting” rules: coordinate with the authors of the existing work and get their agreement, specify the mathematics rather than the existing code, and improve it rather than copy it [The26b]. For `pcf`, an agent first made a read-only intake, listing which of our Lean files advance targets of merged roadmaps, which duplicate something already in Tau Ceti or Mathlib (and so should be replaced by it), and which need a new roadmap. We then wrote roadmap pull requests for local purity, purity of the branch locus, and Abhyankar’s lemma, and ported the code in small pull requests, one topic each. As of 25 October 2026, nineteen of our code pull requests and four roadmap pull requests have been merged, and four code and three roadmap pull requests are open ♠♠♠ Taylor: [counts from gh pr list, 25 Oct; recheck before publishing].

The skills I use for this are in the `taucetipowers` package ([Dup26b] `tdupu/agent-skills-public`): a router, `using-taucetipowers`, sends issue drafting to `create-issue-tauceti`, Lean work to `pr-pipeline-tauceti`, read-only triage to `tauceti-triage`, roadmap pull requests to `roadmap-pr-tauceti`, and contributing a whole existing development to `contribute-to-tauceti`. It reads the target repository’s current rules and templates, and it treats the build, the review state,

and any remote write as separate gates: it stops for my approval before filing an issue or opening a pull request.

6.6. Where formalizations go. Table 11 lists the venues as of October 2026, from each venue’s own description. Mathlib [Theb] is the canonical, human-curated library; it wants general definitions and theorems that fit its design and will be maintained. Tau Ceti [The26d] is described above. Lean Pool [IA26] is a maintained archive of independent formalizations, kept compiling by AI agents as Lean and Mathlib change; it admits completed projects with no **sorry** or **admit** and only the standard axioms [Ili26]. merely-true [mer25] is a sandbox with “no manual human review process”: pull requests from approved accounts merge when the automated build passes, and **sorry** and **axiom** are rejected; a theorem there says nothing about whether its statement means what you think it means. Palomar [Pal26] is a registry, not a library, incubated by the Lean FRO and ICARM: it records that one exact version of a repository proves one exact statement, after a kernel check and an AI check that the formal statement matches the informal claim. It does not endorse the result or replace peer review [Pal26, Tao26], but it gives you something durable to cite. The paper itself still goes on arXiv, or on Hexagon, a new repository in public beta run by the Hexagon Mathematics Foundation and announced by Benjamin Antieau [Hex26, Ant26s]. Hexagon does not accept Lean formalizations as part of a submission, but it links to Mathlib, Palomar, prove2.me, and Tau Ceti; it does not allow AI systems to be contributors. The venues combine: a paper can be on Hexagon and its main theorem a Palomar entry pointing at a Lean Pool project.

Venue	What it is	Who reviews
Mathlib [Theb]	the community library of general, reusable material	human maintainers review every pull request
Tau Ceti [The26d]	library downstream of Mathlib built against human-written roadmaps [The26e]	AI writes and reviews the code; humans write the roadmaps
Palomar [Pal26]	registry of one exact statement at one commit; something to cite	kernel check plus an AI fidelity check; not peer review
Lean Pool [IA26]	archive of completed, sorry -free projects, kept compiling	automated checks plus an LLM review of fit and significance; AI agents maintain the builds
merely-true [mer25]	sandbox; anything that builds	no human review of content: auto-merge when the build passes (contributor accounts approved manually)
Hexagon [Hex26]	repository for the paper itself (beta); hosts no Lean code	links out to Mathlib, Palomar, prove2.me, Tau Ceti

TABLE 11. Where a formalization, or the paper around it, can go.

What “Mathlib quality” means has been written down several times. Mathlib’s own style guide [Lea26b] and pull-request review guide [Lea26c] set the norms; the review guide asks whether a contribution is “general enough to support known future needs” and notes that “incorporating mathematics into mathlib places a burden to maintain what was added.” Mathlib’s contribution page now has a section on AI: a pull request that used AI must say so in its description and, if it contains a substantial amount of LLM-generated code, carry the **LLM-generated** label, the contributor must understand everything the AI wrote, and reviewers “will summarily close without comment any low quality PR produced using LLMs” [Lea26a]. Birkbeck’s mathlib-quality turned these norms into agent instructions: planning tickets with exact statements and verified Mathlib lemmas, a cleanup cadence, a Mathlib-fit assessment, and a self-review before a pull request [Bir26]. On the Lean Zulip, user202729 drew a flowchart for choosing a venue, with comments from Kim

Morrison, Filippo A. E. Nuccio, Chris Birkbeck, and Johan Commelin [use26] (Figure 25). It starts from code quality: beautiful, fully general code with a domain-expert maintainer may be Mathlib; anything else must first be free of `sorry` and `axiom`, and then goes to Tau Ceti if it is reusable and on a roadmap, and to Lean Pool if it is a one-off project. About LLM-written code aimed at Mathlib it says: “Sorry, we don’t trust LLM code. As of Sep 2026, LLM code are terrible.” It does not mention Palomar or merely-true, and in practice the venues are not exclusive.

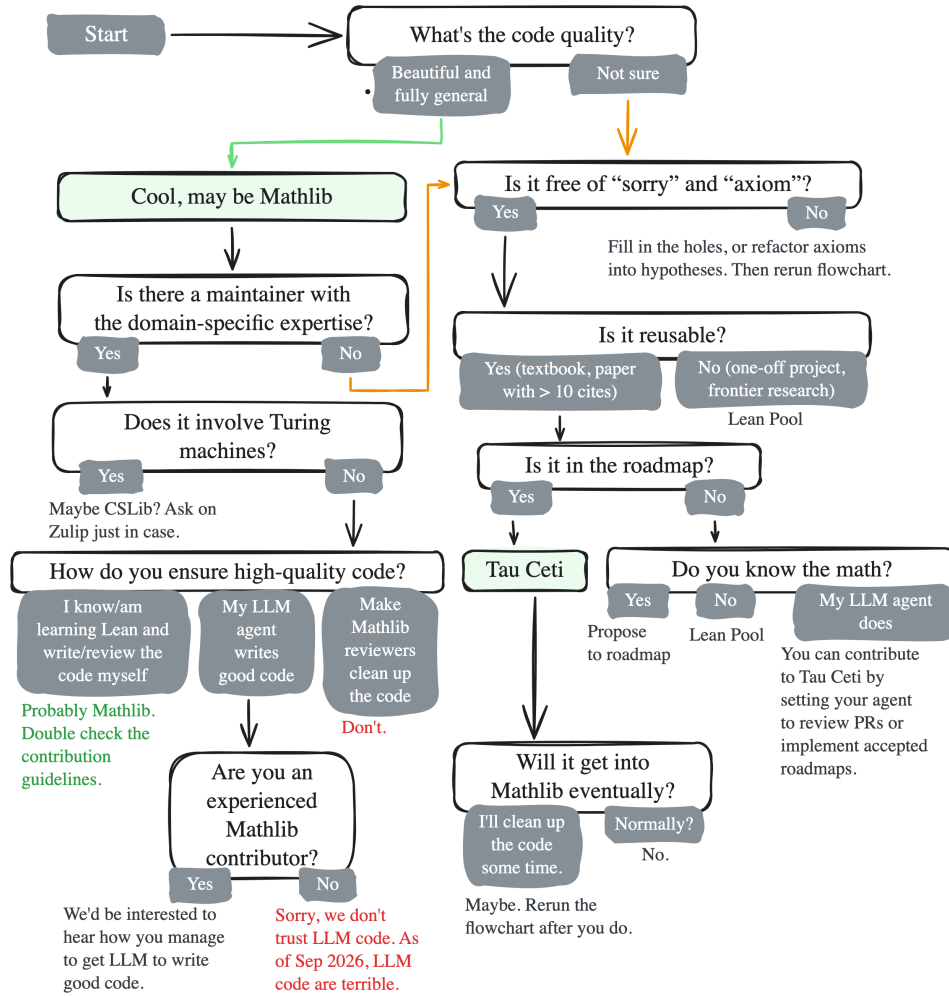


FIGURE 25. user202729’s flowchart from the Lean Zulip [use26].

Anthropic’s Lean proof of Fermat’s Last Theorem (13 million lines, standard axioms only, September 2026) [Ant26k] is a test case. By the rules above it could be registered on Palomar or archived in Lean Pool, but in my view it belongs in merely-true: it is far larger than anything a curated archive is set up to maintain, and what merely-true promises (it builds, with no `sorry` and no extra axioms) is exactly what has been checked.

6.7. Compiling is not fidelity. A compiled proof certifies that the Lean statement follows from the axioms it lists. It does not certify that the Lean statement is the theorem in your paper. Checking that the definitions and statements in Lean say what the paper says is called *fidelity*, and

it is a separate job. In October 2025 Terence Tao described an open problem in optimization solved with substantial help from a language model; its Lean formalization, done by another mathematician with AI help and expert supervision, had, “due to a human formalization error”, a hypothesis “slightly stronger than what was actually intended” [Tao25a]. It compiled, and it proved a slightly different theorem. In July 2026 an AI-generated Lean “proof” of the Collatz conjecture turned out to be “merely exploiting a bug in the Lean kernel (allowing you to prove anything)”, as David Madore reported [Mad26]: compiling is only as good as the checker and the axioms, which is why a finished project lists them. And a correct, machine-checked proof is not yet readable mathematics: John Baez describes a valid, computer-verified, LLM-generated proof that “was 60 pages long when it could have been 8, and contained a lot of weird idiosyncratic terminology” [Bae26]. Tao, about another episode: the Lean formalization “is used to supplement the human-generated proof; it is not a replacement for it” [Tao25b].

So a large Lean file offered as evidence that a paper is right raises four questions. Does it build? Does it have any `sorry`, and which axioms does it use (and, as the pcf report noted, which assumptions are hidden in hypotheses or structure fields)? Is there a *fidelity statement*, a list of the main definitions and theorems with the paper’s version next to the Lean version, so a human can check that they correspond? And is there a paper a human can read? A formalization with a couple of believable `sorry`s and a careful fidelity statement tells you exactly what has not been checked; one without a fidelity statement does not tell you what was proved. Exercise A.17 is an exercise on this.

6.8. First Proof. First Proof [ABH⁺26] is a benchmark run by mathematicians that measures the first step of Figure 24: AI systems get unpublished research lemmas and must produce natural-language proofs, with no human input, to the standard of the literature, graded by blind human referees (Figure 26). The results of the third batch (14 October 2026), and what they suggest about harnesses, are discussed with `frontier-dump` (Section 4.1.3). Only 4 of Astra’s 10 solutions had adequate citations (7 of 10 with ProofCouncil) [Fir26b].

Batch	When	What happened
1	Feb 5–13, 2026	10 unpublished lemmas from 11 mathematicians’ own research; answers encrypted for a week. DeepMind’s Aletheia self-reports 6/10 (majority expert view).
2	results Jun 10, 2026	Formal benchmark, blind human referees. Best system (ProofCouncil) 6 of 10 passing (First Proof report).
3	results Oct 14, 2026	5 systems, 24 h each, no human input, 30 referees at CMU’s ICARM. ProofCouncil 9/10 (\$1,888) · GPT-6 Astra alone 7/10 (\$36) · open-weight (Kimi K3) 0/10
Still weak		Citations: only 4/10 Astra solutions adequately cited (7/10 with ProofCouncil). Exposition weak.

FIGURE 26. The three First Proof batches of 2026 [ABH⁺26, Fir26b, FJhK⁺26].

6.9. leanpowers. `leanpowers` is my package of Lean skills (public copy at [Dup26b] `tdupu/agent-skills-public`). You type `/using-leanpowers` and the router sends each request to one of 27 small skills, in six groups: setting up a project, planning, writing, checking, improving, and contributing (Figure 27). Compiling and fidelity are separate gates: `lean-verify` checks the build, the `sorry`s, and the axioms, and `lean-fidelity` checks whether the Lean says what the paper says. There is a skill for each venue: `lean-mathlib-fit` and `lean-prepare-pr` for

Mathlib, `contribute-to-tauceti` for Tau Ceti, and `to-palomar` for Palomar. The package adapts Birkbeck’s `mathlib-quality` at version 0.73.3 [Bir26], splitting its workflows into composable skills and replacing its global hooks with scoped checks; the catalogue of all my skills is in Section B.

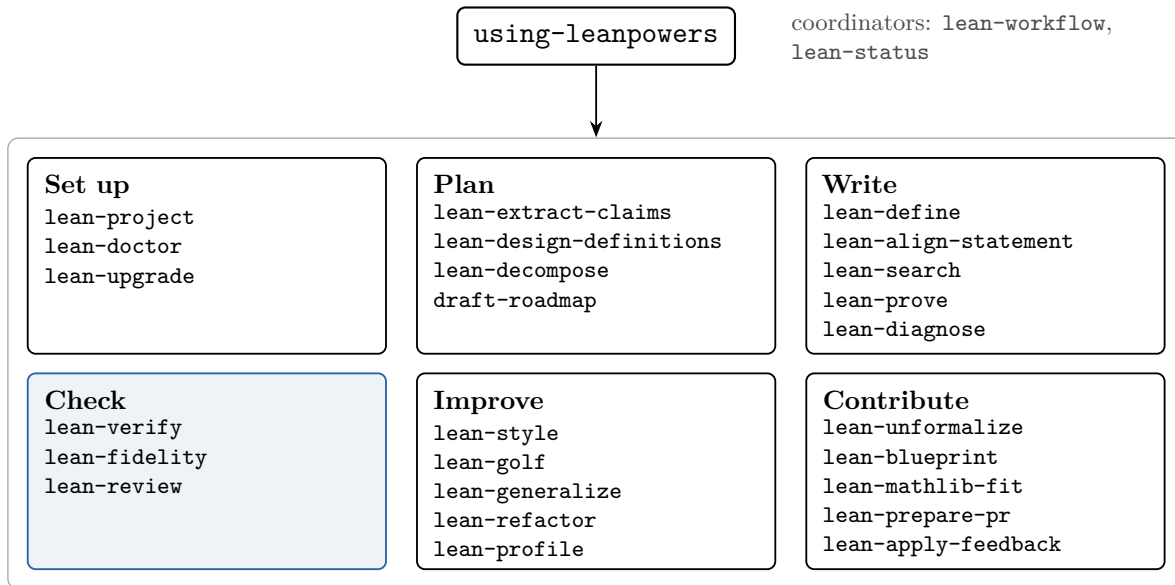


FIGURE 27. The `leanpowers` router and its skill groups.

7. AI USAGE STATEMENTS

When a company announces that its AI has proved a conjecture X , it makes two claims: a mathematical claim, that proof P proves X , and an empirical claim, that machine M produced P . The first is checked the way proofs have always been checked. The second is an experimental result, and an *AI usage statement* reports it, in miniature, whenever AI does real work toward one of your results. Models change and their outputs vary, so no one can re-run your session; what a statement can do is let a reader *inspect* what you did. The experimental sciences have asked for this for a long time, from Bacon’s request that experiments be reported “that men may be free to judge for themselves” [Bac58] and Boyle’s reports of his failed experiments [BG20] to the National Academy of Sciences’ call for “sufficient information to enable the informed reader to assess the validity of the publication’s conclusions” [NNI93]. Saying nothing is also a claim, namely that you did the work yourself.

7.1. An example: the pcf paper. The paper *PCF all the way down*, with Alon Levy, Thomas Tucker, and Zhuoran Wang [DLTW26], discloses its AI use in two places, and the coauthors gave permission to describe it here. The draft of 24 October 2026 is the one described.

The first place is a short subsection, “Software and AI assistance”, at the end of the introduction, after the outline of the paper. In one paragraph it names every model and harness used (GPT-5.4 in ChatGPT Classic; several GPT-5 and GPT-6 models in the Codex desktop application; Claude Fable 5, Fable 5.1, Opus 5, and Opus 5.5 in Claude Code), says what they helped with (proofs, counterexamples, formalization, and drafting), lists the examples and theorems that AI-assisted work contributed to, says which model drafted the subsection itself and that the authors had not yet reviewed that wording, and states that the human authors are responsible for every result and citation. It ends with a pointer to the full account. A reader who stops at the introduction knows which results to look at with this in mind, and the introduction is where readers look for

what a paper relies on; the AMS asks for disclosure “in the Materials and Methods (or similar section)” [Ame].

The second place is a section at the end, also called “Software and AI assistance”, with thirteen subsections, followed by an appendix on token accounting. Its first seven subsections are methods: how the manuscript was typeset and compiled from the authors’ notes, how the agents were managed (the contract files at the top of the repository and the skills that write them), how records were kept, how research questions were put to the strongest available model (`frontier-dump`, Section 4.1.3), where adversarial reviews were run, how computations were repeated, and how the statement itself was drafted. Each skill is cited to the public repository that holds it, so a reader can read the instructions the agents followed. The computations are reported as output of named software at named versions (SymPy 1.13.2 and Singular 4.4.1), as any computation in a paper would be.

The subsection on findings is the empirical claim. It tells the story in order: the second and third authors had reduced the theorem to an equality they could not prove; OpenAI’s GPT-5.4 in ChatGPT Classic found a counterexample to it; Anthropic’s Claude Fable 5 in Claude Code suggested the correct upper bound, with a proof by Zariski–Nagata purity followed by Abhyankar’s lemma; and OpenAI’s GPT-6 Astra in Codex, at reasoning effort `xhigh`, found an example showing that one of the two correction terms in that bound is necessary. Each finding names the model and the harness, and the reasoning setting where one was recorded; the first two say explicitly that none was recorded, and one later finding gives no setting without saying so. The reasoning setting is reported because the same model at different settings is, for practical purposes, a different instrument. After each finding, the paper says, the authors checked it by hand and the computations behind it were rerun before it was used further.

The provenance subsection asks whether the model’s idea was already known. It was: the authors had GPT-5.5 in Codex search for the purity-plus-Abhyankar strategy, and the paper lists where it appears, starting with SGA 1, Exposé X, and Grothendieck and Murre, with pinpoint locators. It says that this is a comparison of sources, not evidence that a model copied one. Credit is the reason: models are trained on the literature, so an argument a model hands you may already be in print, and only a search will tell you. The literature-review subsection reports a negative search the same way: a sweep for an earlier removal of a smoothness hypothesis that the paper removes found nothing, and the paper calls this absence claim weak because no independent cross-check was run; the same sweep turned up Jonsson’s 1997 thesis, which narrowed the paper’s novelty claim in dimension two.

The due-diligence subsection lists related questions that were put to frontier models (the rigid analytic and complex analytic analogues), with what was proved, what failed, and the remark that the accompanying supplement was written entirely by AI. This records that the obvious next questions were asked and what current models could do with them, which is the check of Section 7.3.

The formalization subsection names the formal theorems, the version of Mathlib (a tag and a revision), and the axioms the proofs use (`propext`, `Classical.choice`, and `Quot.sound`), and states that none depends on `sorry`. It also says what the formalization does *not* establish (listed in Section 6.3), names the models that wrote the Lean code, and gives the public repository. Compiling is not the same as saying the right thing (Section 6.7), so the subsection says exactly what was checked and by whom.

The appendix tabulates, for each task whose output the paper uses, the model and harness, the tokens by class (uncached input, cache writes, cache reads, and output), and the cost at the providers’ published per-token rates, with the rates and the date they were checked. These are API-equivalent charges, not billing; drafting and corrections with Claude Opus 5.5 in Claude Code, for example, came to \$15.92. Rows whose usage was not recorded say “not recorded”, and rows with no applicable rate are marked unpriced, “which is not a claim of zero cost”. The work has a cost, and a reader can judge it only if it is written down.

Throughout, the models are described as “origin metadata; they are neither authors nor mathematical authorities”, and the authors take responsibility for every statement and proof. The AMS says that AI tools cannot be authors [Ame]. A coarser style of report, DeepMind’s autonomy levels and human–AI interaction cards, is described in Section C.

7.2. Records. A statement like that cannot be written from memory at the end. The pcf repository keeps an `ai/` folder with one dated subfolder per job, holding what the agents produced, and two running records beside them: `ai-usage.md`, which records for each task the prompt, the models and harnesses, the work performed, each result with an account of how it was derived, the literature searched, and the negative findings; and `tokens.md`, which records the tokens and their cost. An `AI-POLICY.md` at the root of the repository states, as rules an agent can be checked against, what must be recorded. Figure 28 is one entry of `tokens.md`, shortened: a 19-minute review run by Claude Opus 5.5. It labels the cost as API-equivalent, says that the split between input and output tokens is assumed, and gives the range the cost could fall in.

ai/tokens.md (excerpt; six of twelve rows)

```
## 2026-10-21T17:22-04:00__frontier-dump__ramification-corollary-review

| Field | Value |
| --- | --- |
| Model / harness | `claude-opus-5-5` (self-reported by the worker as `claude-opus-5-5[1m]`); Claude Code CLI 2.1.306 |
| Tokens | 238,912 in total, as reported by the harness (`subagent_tokens`), with no breakdown by token class |
| Assumed split | output 25,000, from the worker's own estimate; input 213,912 for the remainder. Cache-write and cache-read tokens are not separable from this figure. |
| Cost (USD) |  $1.355648 = (213,912 \times 4 + 25,000 \times 20) / 1e6$ . Token charges only, at API-equivalent rates; this is not actual billing. |
| Bound | minimum 0.0477824, if every token is priced at the cache-read rate; maximum 4.77824, if every token is priced at the output rate |
| Duration | 00:19:18.5 (1,158,525 ms), with 45 tool uses |
```

FIGURE 28. One entry of the pcf paper’s cost record.

In my setup two skills are the only writers of these files (`update-ai-usage` and `update-tokens`), and others draft the statement from them and audit the draft (`write-materials-and-methods` and `latex-ai-statement`) [Dup26e]. The skills are not needed: an agent pointed at this section can set up the same records in any repository. The git history is a record too, with a date and a message for every change, so keep it rather than squashing it before submission. Before anything goes public, scan records, transcripts, and code for passwords, personal data, and collaborators’ unpublished work; scanners such as `gitleaks` catch much of it. Exercise A.19 drafts a statement from records like these.

7.3. Checking that a problem cannot be one-shot. Funders, hiring committees, and researchers are looking for problems that need work. A problem that a frontier model can solve in one attempt (*one-shot*) is not a good problem for a grant proposal, a research statement, or a thesis: there is little left to do, since one can one-shot it and be done. Whoever poses a problem should therefore know which kind it is. The check is cheap. Put each problem to the strongest model you can reach, once asking for a proof and separately asking for a disproof; the `frontier-dump` skill (Section 4.1.3) does both and keeps the record.

Do it safely: from an account with training turned off (Section 3.4), so that an unpublished idea does not become training data. Then say in the proposal or statement that you checked: when, with which model and harness, and what happened. A failed attempt is information for the reader, and it saves the next person, and the next model, from spending effort on the same attempt. Some work cannot be one-shot by its nature, such as large computations and databases, which need the computation itself. The due-diligence subsection of the pcf paper (Section 7.1) is an example of reporting such runs. One more check fits here: have an agent review the document adversarially against what the readers want (the solicitation, the job advertisement, or the fellowship’s criteria); my `pd-review-as` skill reads an NSF project description as a named kind of reviewer. Exercise A.18 is this check as an exercise.

7.4. Provenance runs. For each idea an AI produced, run a literature search with a similar model and harness for earlier references, and report what it found, *even when it found nothing*. I call these *provenance runs*. The pcf paper’s search for the purity-plus-Abhyankar strategy is one (Section 7.1). A provenance run finds candidates; a citation still requires opening the source, and the skills that enforce this are in the catalogue (Table 14).

7.5. Confidentiality and policies. Turn off training on your account before anything unpublished goes into a session (Section 3.4). The AMS journals tell referees not to upload papers under review to an AI tool “in any format, for any reason” [Ame]. To use AI tools on a paper you are refereeing at all, you need permission from both the journal and the author. A coauthor’s unpublished work is in the same position: ask before it goes into a model. The AMS requires authors who use AI tools to disclose which tools they used and how, and AI tools cannot be authors [Ame]. The Leiden Declaration, endorsed by the International Mathematical Union, says that responsibility “for the completeness and accuracy of citations to relevant prior work, remains exclusively with the human authors”, and that announcements through press releases or blog posts “cannot replace peer-review or community scrutiny” [Lei26]. For a thesis, find and follow your graduate school’s policy on AI use; for homework, qualifying exams, and anything graded, the instructor’s rules apply. Funders, universities, and hiring departments also publish rules on AI use in applications, and these change.

8. ORCHESTRATION

An agent is a model in a harness (Section 3). *Orchestration* is the next layer: a program that starts agents, gives them work, and tracks that work in a systematic way, outside any one agent’s context window, so that a crashed or exhausted agent can be replaced and nothing is lost. Once several agents run in parallel, the person running them becomes the bottleneck (switching terminals, remembering who is doing what, relaying messages), and an orchestrator takes over that bookkeeping. The morning report in the introduction (Section 1) shows what this looks like after one night.

Steve Yegge’s post introducing his orchestrator, Gas Town, is where I learned about this, and it is a fun read [Yeg26] (steve-yegge.medium.com/welcome-to-gas-town-4f25ee16dd04). Its successor, Gas City [Gas26] [gastownhall/gascity](https://gastownhall.com/gascity), is a toolkit for building orchestrators, and it lets each agent run on a different provider. My configuration of it for mathematics, `mathcity` [Dup26] `td upu/mathcity`, is public: it holds the roles, the workflows, the policies, and the skills that it runs.

8.1. Hooks. A *hook* is a shell command that the harness runs automatically at a fixed moment in an agent’s session [Ant26c], and hooks are one of the ways an orchestrator keeps agents on track. In Claude Code, a hook at session start can print the state of the project, and what it prints is added to the agent’s context; the superpowers plugin uses one to load its instructions into every new session. A hook before each prompt can add fresh information alongside the prompt. A hook before a tool runs can block the call, for instance a dangerous command, and one after a tool runs can rebuild a $\text{\LaTeX}$ file or run the tests every time a file is edited. A hook before compaction can

save a note of what matters before the harness summarizes the conversation, and a hook when the agent stops can check the work and keep the agent going. The difference from an instruction in `AGENTS.md` is that a hook is a program, so it runs every time, whereas the model may or may not follow an instruction (Figure 13). Hooks live in the harness’s settings file. Here is one of mine, which writes a handoff note before compaction:

`~/.claude/settings.json` (excerpt)

```
"hooks": {
  "PreCompact": [
    {
      "matcher": "",
      "hooks": [
        {
          "type": "command",
          "command": "bash ~/.claude/scripts/pre-compact-handoff-<repo>.sh"
        }
      ]
    }
  ]
}
```

There is no need to write these by hand: describe the behavior to the agent (“every time you edit a `.tex` file, recompile it and show me the errors”) and ask it to set up the hook.

8.2. Worktrees. Parallel agents in one repository overwrite each other’s files unless each has its own copy. A *git worktree* [Git26] git-scm.com/docs/git-worktree is a second working copy of the same repository on its own branch, sharing one history, so work done in it merges back through ordinary git. Codex and Claude Code can create worktrees for their agents (in Claude Code, `claude -worktree`). Here is the pcf repository on 25 October 2026, with the main copy and one worktree for a feature branch:

```
$ git worktree list
~/repos/pcf                84c97bc [main]
~/repos/pcf-branched-covers 37d2884 [feature/branched-covers]
```

9. COSTS, QUALITY, AND CREDIT

Four facts to end with.

Cost. OpenAI’s Navier–Stokes run reportedly used “about 10,000 agents concurrently, at a cost of millions of dollars”; as Javier Gómez-Serrano put it, “very few mathematicians will have resources of that scale” [Huc26]. What plans and API access cost an individual is in Section 5.

Open models. Open-weight models can be run on your own hardware, used in the same harnesses, and fine-tuned on your own data; the section on language models explains how (Section 2.8).

Quality. Paid applications have to make a profit. Cory Doctorow’s name for platforms degrading once users are locked in is *enshittification* [Doc25]. In January 2026 OpenAI announced that it would test advertising in the Free and Go tiers of ChatGPT [Ope26g].

Credit. When OpenAI announced its Navier–Stokes result on 8 September 2026, Tristan Buckmaster and Levent Alpöge, who had used OpenAI’s Codex while working on the problem, asked whether their sessions had reached OpenAI’s models [Buc26]. OpenAI’s answer that day (Figure 29) was that no specific user data was accessed, but “we cannot rule out that de-identified data derived from their usage of our products helped improve our models.” Two days later OpenAI updated its announcement to say that Buckmaster’s Codex prompts over the two preceding months “could

not have influenced the system in any way, including through training” [Ope26f]; the update says nothing about earlier use.

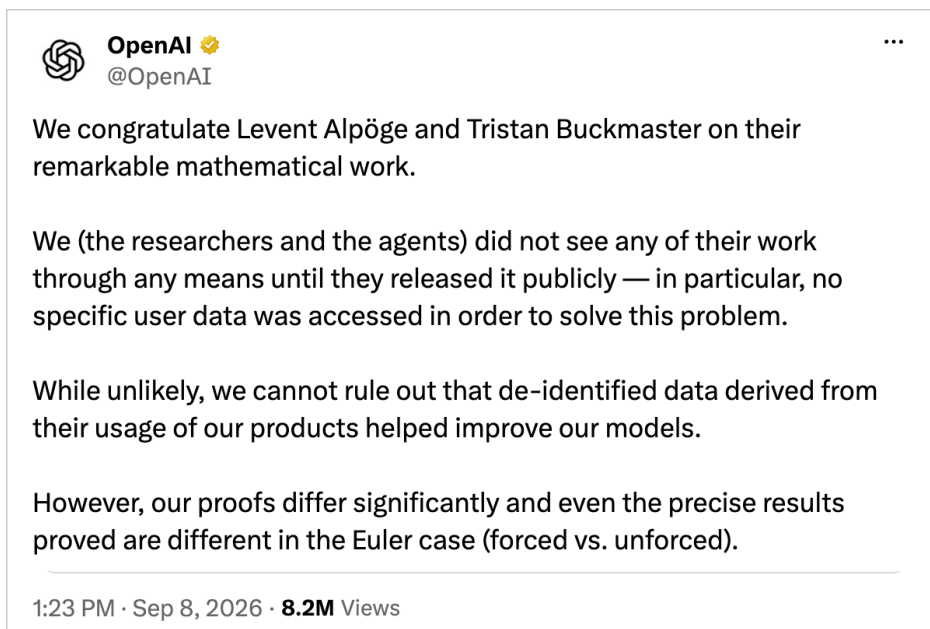


FIGURE 29. OpenAI on X, 8 September 2026 [Ope26l].

APPENDIX A. EXERCISES

The exercises are collected here, grouped by the section they belong to. Each one has a goal and a test for when it is done. Do them in a repository of their own, commit after each, and record what the agent did in `ai/ai-usage.md` as you would for real work. The two hello-worlds (Exercise A.5 and Exercise A.16) are the place to start: they show that the whole setup works before anything that matters depends on it.

A.1. Language models. These go with Section 2. A model samples each token from a probability distribution, so fluent output can be wrong; the first three exercises show where.

Exercise A.1 (Explain a lemma from a paper you know well). *Goal:* see where an explanation is good and where it is fluent but wrong, on mathematics whose answer you know. Put the PDF in `refs/` and ask: “Explain the proof of Lemma 3.2 in `refs/smith.pdf` to a first-year graduate student, step by step. Point out every step the paper leaves to the reader.” *Done when* you have marked each step of the explanation right, wrong, or vague.

Exercise A.2 (Transcribe handwritten notes into $\text{\LaTeX}$). *Goal:* learn where transcription errors hide. Take phone photos of a few pages of notes, put them in the folder, and ask for a $\text{\LaTeX}$ file using your own macros. *Done when* you have checked the transcription line by line against the photos; this is where a dropped sign or a wrong index slips in.

Exercise A.3 (A false-premise test). *Goal:* see whether your model builds on a falsehood you supply. In a fresh session, tell the agent that a statement in your field which you know to be false (or open) “was proved last year”, and ask it to reconstruct the proof. *Done when* you have recorded whether it declined, corrected the premise, or produced a “proof”; if the last, find the step where it went wrong.

A.2. Harnesses. This goes with Section 3: the harness is what lets the model run programs on your machine.

Exercise A.4 (Run a computation and save the script). *Goal:* the habit that every number in your paper comes from a saved script you can rerun. Ask: “Using Sage, compute the rank and torsion of the elliptic curves of conductor below 100. Save the script in `scripts/` and its output next to it, and check three answers against the LMFDB.” *Done when* the script is committed and you have rerun it yourself.

A.3. Skills. These go with the material on skills and prompts in Section 4 (Section 4.1).

Exercise A.5 (A hello-world skill). *Goal:* learn where skills live and how they load. Ask the agent: “make a skill called `hello` in `.agents/skills` that prints today’s date and the name of this repository”. Restart the harness and run `/hello` (or `$hello`). *Done when* the skill prints the date and the repository name, and you can point to its `SKILL.md`.

Exercise A.6 (One set of skills for every harness). *Goal:* one skills folder that Claude Code and Codex both read. Keep the skills in `~/.agents/skills` and make `~/.claude/skills` a symbolic link to it. *Done when* `ls -l ~/.claude/skills` shows a link to `~/.agents/skills`, and the `hello` skill of Exercise A.5 runs in each harness you have.

Exercise A.7 (Prove and disprove a lemma you already know). *Goal:* see how the strongest model behaves when asked both ways. Pick a lemma whose proof you know, and a small false variant of it (drop a hypothesis). Run `/frontier-dump "prove X"` and, separately, `/frontier-dump "disprove X"` on each (or ask the strongest model you have in two fresh sessions). *Done when* you have four reports and have written one line on each: did it prove the true lemma, refute the false one, and resist the wrong direction in the other two runs?

Exercise A.8 (Ask the strongest model about the Riemann Hypothesis). *Goal:* see what a frontier model does with a famous open problem, and practise recording a negative result. The Riemann Hypothesis is one of the Clay Mathematics Institute’s Millennium Problems, listed as unsolved [Cla]. Run `/frontier-dump "Prove the Riemann Hypothesis."` and, in a separate run, `/frontier-dump "Disprove the Riemann Hypothesis."` Expect neither run to settle it. A good report says that the problem is open, separates what is proved (partial results, with sources you can open) from what is conjectured, and lists in its refutations section any premise of the prompt it rejects. A report that claims a proof or a counterexample has failed, and finding the step where it fails is part of the exercise. Expect the runs to be long; look at what they cost in `tokens.md` afterwards. *Done when* both dated folders are committed, you have written one line on each report (what it claimed, and whether that claim survives your reading), and `ai/ai-usage.md` has an entry for each run giving the date, the model, the prompt, and the outcome “no progress” (or what went wrong), so the negative run is on record.

Exercise A.9 (An adversarial review of your own draft). *Goal:* hear a referee’s objections before a referee does. Ask: “Act as a hostile referee. Read `latex/paper.tex` and list every gap, unjustified step, and unclear definition, most serious first. Write the report in `ai/`.” Do this only with your own work, or with a collaborator’s draft when they agree; using AI on a paper you are refereeing needs the permission of both the journal and the author (Section 7.5). *Done when* you have sorted every point in the report into “right”, “wrong”, and “right but minor”.

Exercise A.10 (Repair one misbehavior). *Goal:* fix an agent by fixing its instructions. Take one thing your agent keeps doing that you dislike, make it happen on demand, find the line of `AGENTS.md` or the skill that causes it, and repair that line, by better context rather than a new rule if you can (`adjust-behavior` does this for you). *Done when* the reproduction no longer shows the behavior and the file is no longer than before, or you can say why it had to grow.

A.4. AGENTS.md and git. These go with Section 4.2.

Exercise A.11 (Let the agent drive git). *Goal:* learn git by watching an agent use it. In a test repository, ask the agent to commit a change, make a branch, commit something on it, switch back, and undo the last commit, explaining each command. *Done when* you can read `git log` and explain every entry, and could undo a bad agent edit on your own.

Exercise A.12 (Write your AGENTS.md and have it critiqued). *Goal:* a short AGENTS.md that matches your repository. Write one for a real project, starting from the sample in Section 4.2. Then open a fresh session and ask: “Critique this AGENTS.md: which rules are vague, which contradict each other, which describe files that do not exist, and what would a new collaborator still not know?” *Done when* every file it names exists, every rule is one you would want enforced, and it still fits on one screen.

A.5. MCP. These go with Section 4.3.

Exercise A.13 (Ask the LMFDB through its MCP server). *Goal:* get an answer from a database rather than from the model’s memory. Connect Andrew Sutherland’s LMFDB server [Sut26] AndrewSutherland/lmfdb-mcp. In Claude Code [Ant26e],

```
$ claude mcp add --transport http lmfdb https://mcp.lmfdb.org/mcp
$ claude mcp list
```

and check that `lmfdb` is listed as connected (inside a session, `/mcp` shows its tools). In the Claude app, add a custom connector with the same URL. Then ask, in plain language, “What is the rank-2 elliptic curve over $\mathbb{Q}$ of smallest conductor? Use the LMFDB.” *Done when* the agent has called the server (you can see the `run_sql` call and its result) and returned `389.a1`, with coefficients $[0, 1, 1, -2, 0]$, that is, $y^2 + y = x^3 + x^2 - 2x$, and you have checked the label on the LMFDB website.

Exercise A.14 (A literature search, with every citation checked). *Goal:* find out how often an agent’s references are wrong. Ask: “Find the papers that prove or use <a result you care about>.” For each, give the arXiv or DOI link, the theorem number, and a one-line summary, and download the arXiv PDFs into `refs/`.” If you have the arXiv, Semantic Scholar, or Stacks Project servers connected, the agent will use them. *Done when* you have opened every reference and recorded which ones were right, which were wrong in small ways, and which did not exist.

A.6. Models, prices, and plans. This goes with Section 5.

Exercise A.15 (Price one of your own sessions). *Goal:* see what your own work would cost at API list prices. After a real working session, run `npx ccusage` in a terminal (it reads the session logs on your disk [ccu26] `ccusage/ccusage`), and in Claude Code run `/usage` to see how much of your plan’s allowance you used. *Done when* you can say, for that session, how many tokens were read, how many written, what they would have cost through the API, and what fraction of your plan they used.

A.7. Autoformalization. These go with Section 6.

Exercise A.16 (A hello-world Lean project). *Goal:* know that formalization is there when you need it. Ask the agent to create a Lean project that imports Mathlib and checks one line. It takes a while to download. *Done when* `lake build` reports success on a file that uses Mathlib.

Exercise A.17 (Formalize one line and check fidelity). *Goal:* feel the difference between compiling and being right. In the project of Exercise A.16, ask the agent to state and prove in Lean a one-line fact you know (for instance, that the sum of two odd integers is even). Then read the Lean statement yourself, symbol by symbol. *Done when* the file builds with no `sorry`, and you have written a two-column fidelity statement (your sentence, the Lean statement) and checked that every hypothesis and quantifier matches.

A.8. **AI usage statements.** These go with Section 7.

Exercise A.18 (Check that a proposed problem cannot be one-shot). *Goal:* know, before you propose a problem, whether the strongest model solves it in one attempt. Take one problem from a grant proposal, a research statement, or a thesis plan. Use an account with training turned off (Section 3.4), so the problem does not leave your hands. Run `/frontier-dump "prove X"` and, separately, `/frontier-dump "disprove X"`, giving the problem and the sources a reader would need, and nothing else. *Done when* you have read both reports and know whether either run solved the problem, and `ai/ai-usage.md` records the date, the model, and what happened, so that one sentence in the proposal can say the problem was checked. If a run did solve it, you have a result rather than a proposal.

Exercise A.19 (Draft an AI statement from your records). *Goal:* check that your records are good enough to write from. For a paper, a thesis chapter, or a homework set where you used AI, ask the agent to draft the “Software and AI assistance” paragraph using only `ai/ai-usage.md`, `ai/tokens.md`, and the git history, and to list what it could not find there. *Done when* every sentence of the draft points to an entry in your records, and the list of gaps tells you what to start recording.

APPENDIX B. A CATALOGUE OF SKILLS

This appendix lists every skill named in the guide, with a link to a public copy. Most are in my repository [Dup26b] `tdupu/agent-skills-public` (GPL-3.0); the skills of the brief system used in orchestration are in [Dup26e] `tdupu/mathcity`; the rest belong to other people and link to their repositories. To install a skill, point your agent at its folder and ask it to install it. The public repository is a snapshot of a private working one, so fixes can lag behind. ♠♠♠ Taylor: [Eight links (check-wheel, coordinate-review, create-artifact, get-best-apis, get-best-models, revise-artifact, to-palomar, xkcd-927) work only after agent-skills-public is pushed; six of them are live in mathcity already.]

B.1. The starting point: superpowers. Superpowers, by Jesse Vincent [Vin25] `obra/superpowers`, is described with skills (Section 4.1.1). Its router, `using-superpowers`, makes the agent look for an applicable skill before doing anything, and the routers in Table 12 call it first.

B.2. Packages. A package is a router skill, `using-name`, together with the skills it routes to, so you need to remember one name per area. Table 12 lists the packages in `agent-skills-public` as of 25 October 2026. The `latexpowers` row deserves one more sentence: the package accepts a citation only after an agent has opened the source and found the claim at a specific locator, and it writes every AI task to `ai-usage.md` and `tokens.md`, against which `latex-ai-statement` audits the paper’s AI statement before submission (Section 7).

Package	What it does	Router
latexpowers	Write and check manuscripts, with the AI statement built in	using-latexpowers
mathpowers	Prove, refute, and search arXiv, the Stacks Project, the LMFDB, and Mathlib	using-mathpowers
leanpowers	Formalize in Lean 4 and Mathlib, with fidelity checks	using-leanpowers
lmfdb	Query the LMFDB, and move data between Magma, text files, and databases	search-lmfdb
docpowers	Document repositories and mathematical software	using-docpowers
testpowers	Choose and write tests (Sage, Macaulay2, Magma, Lean, and L ^A T _E X)	using-testpowers
grantpowers	Assemble an NSF proposal package	using-grantpowers
pdpowers	Write and panel-review the NSF Project Description	using-pdpowers
teachingpowers	Syllabus, course page, notes, and assessments	using-teachingpowers
taucetipowers	Contribute roadmaps, issues, and pull requests to Tau Ceti	using-taucetipowers
youtube-powers	Publish lectures on YouTube	youtube-powers

TABLE 12. Packages in `tdupu/agent-skills-public` [Dup26b].

B.3. Every skill in the guide. Tables 13 to 16 list the skills by what they are for, with a one-line summary taken from each skill’s own description.

Skill	What it does
<code>frontier-dump</code>	Send a question to the strongest model the harness can reach; save a dated report with a refutations section, <code>ai-usage.md</code> , and <code>tokens.md</code>
<code>grilling</code>	Interview you, question by question, to stress-test a plan
<code>grill-with-docs</code>	A grilling session that also writes decisions and a glossary
<code>domain-modeling</code>	Pin down a project’s terms and record its decisions
<code>critical-review</code>	Adversarial review of any artifact, with a verdict and action items
<code>doubt</code>	When a claim is doubted, a fresh agent told the claim is wrong looks for the error
<code>coordinate-review</code>	Alternate critic and reviser agents on an artifact until it is approved
<code>create-artifact</code>	Write an artifact from a specification (the creator in <code>coordinate-review</code>)
<code>revise-artifact</code>	Apply a review’s action items (the reviser in <code>coordinate-review</code>)
<code>fp-finder-skill</code>	Review and revise a skill to a fixed point; a revision must be shorter and still approved
<code>fp-finder-latex</code>	Bounded independent review and revision of a manuscript
<code>referee-report</code>	Line-by-line referee-grade review of your own manuscript, written to <code>ai/</code>
<code>revise</code>	Apply an accepted review to the manuscript item by item
<code>adjust-behavior</code>	Repair the instructions behind an agent’s bad behavior: reproduce, find the cause, repair, and verify
<code>behavior-repro</code>	Turn a complaint about an agent into a reproduction
<code>behavior-cause</code>	Trace a reproduced behavior to the files that produced it
<code>behavior-verify</code>	Show that an edit changed the behavior it was meant to fix
<code>check-wheel</code>	Before building, check whether something already does it
<code>xkcd-927</code>	Consolidate competing documents into one existing source instead of writing a new one

TABLE 13. Asking, reviewing, and repairing (Section 4.1).

APPENDIX C. OTHER REPORTING STYLES

Google DeepMind reports AI contributions to its own mathematics differently from the `pcf` statement described with AI usage statements (Section 7.1). Their paper *Towards Autonomous*

Skill	What it does
<code>math-workflow</code>	The shared process behind <code>latexpowers</code> and <code>mathpowers</code> , from prototype to integrated manuscript
<code>write-proposition</code>	Write one proposition and its proof, only from an evidenced claim
<code>contradiction-check</code>	Report any later claim that contradicts an earlier one
<code>track-down-reference</code>	Find the source of a claim and verify it by opening it, with a pinpoint citation
<code>check-citations</code>	Check every citation against the bibliography and flag those that are not pinpoint
<code>doc-provenance</code>	Add or audit sources, attribution, licenses, and dates in documentation
<code>purge-claudisms</code>	Remove stock machine phrasing from a draft
<code>update-ai-usage</code>	Append a dated entry to the repository’s <code>ai-usage.md</code>
<code>update-tokens</code>	Record tokens and their cost in the repository’s <code>tokens.md</code>
<code>latex-ai-statement</code>	Audit a paper’s AI statement against <code>ai-usage.md</code> and <code>tokens.md</code>
<code>write-materials-and-methods</code>	Draft the software and AI section of a paper from those records
<code>init-repo-docs</code>	Set up a research repository’s <code>AGENTS.md</code> , <code>LAYOUT.md</code> , <code>AI-POLICY.md</code> , and the other files
<code>check-layout</code>	Check a repository against its own <code>LAYOUT.md</code>
<code>pd-review-as</code>	Review an NSF Project Description as one named kind of reviewer
<code>search-arxiv</code>	Search arXiv and return metadata and BibTeX
<code>search-scholar</code>	Search Semantic Scholar for papers and citations
<code>search-stacks</code>	Retrieve and search Stacks Project tags
<code>search-mathlib</code>	Find Mathlib declarations by description, name, or type
<code>search-lmfdb</code>	Query the LMFDB through its MCP server (Section 4.3)
<code>get-best-models</code>	Recommend an open-weight model that fits your hardware
<code>get-best-apis</code>	Build a live table of model benchmarks and API prices (Section 5)

TABLE 14. Writing, sources, and records (Section 7).

Skill	What it does
<code>lean-verify</code>	Check the build, the <code>sorrys</code> , and the axioms before calling Lean work checked
<code>lean-fidelity</code>	Check that Lean declarations say what the paper says
<code>lean-mathlib-fit</code>	Decide whether a declaration belongs in Mathlib or is already there
<code>lean-prepare-pr</code>	Prepare a Lean contribution for review
<code>to-palomar</code>	Package a formalization as a Palomar registry entry
<code>tauceti-triage</code>	Check existing Lean work against Tau Ceti’s roadmaps and open pull requests
<code>contribute-to-tauceti</code>	Contribute an existing formalization to Tau Ceti end to end
<code>roadmap-pr-tauceti</code>	Draft a Tau Ceti roadmap pull request
<code>create-issue-tauceti</code>	Draft an issue for Tau Ceti or its roadmap
<code>pr-pipeline-tauceti</code>	Take a change through a Tau Ceti pull request
<code>mathlib-quality</code>	Chris Birkbeck’s plugin for bringing Lean code up to Mathlib standards [Bir26]; <code>leanpowers</code> adapts it

TABLE 15. Lean and Tau Ceti (Section 6).

Mathematics Research [FTB⁺26] introduces Aletheia, a research agent built on Gemini Deep Think, and uses two devices.

The first is a pair of axes, borrowed from the levels used for self-driving cars (Figure 30). Autonomy runs from primarily human (with secondary AI input), through human–AI collaboration,

Skill	What it does
brief-prep	Turn a piece of work into a brief, a decision ready for a human, with every gate passed
create-brief	Write the brief file
catch-no-brainer	Classify a brief as routine or as needing a human decision
present-briefs	Present briefs one at a time, with the next one ready
present-it	Put the context for one decision into the conversation
adjudicate-brief	Record a decision so that it survives across sessions
check-claims	Find the claims an action depends on that have no source

TABLE 16. The brief system in `tdupu/mathcity` (Section 8).

to essentially autonomous; significance runs from Level 0 (negligible novelty) to Level 4 (landmark breakthrough). All of their results sit at Level 2 (publishable research) or below.

	Primarily Human (secondary AI input)	Human-AI Collaboration	Essentially Autonomous
<i>Level 0: Negligible Novelty</i>			Erdős-652, 654, 1040 (Feng et al., 2026b)
<i>Level 1: Minor Novelty</i>			Erdős-1051 (Feng et al., 2026b)
<i>Level 2: Publishable Research*</i>	Complexity Bounds (ACGKMP26) Arithmetic Volumes (FYZ26)	Generalized Erdős-1051 (BKKKZ26) Independence Polynomials (LeeSeo26)	Eigenweights (Feng26)
<i>Level 3: Major Advance</i>			
<i>Level 4: Landmark Breakthrough</i>			

Table 1 | Classification of all AI-assisted mathematics results encompassed in this work. Detailed descriptions of the categories, with discussion and examples, can be found in §6.1.

*Works listed as Level 2 in this table have been submitted for publications.

FIGURE 30. Autonomy and significance levels [FTB⁺26, Table 1]. © Google DeepMind, CC BY-NC-ND 4.0.

The second is a “Human-AI Interaction card”, which three of their papers carry: an outline transcript of who prompted what, with a link to the raw prompts and outputs [Goo26e] `google-deepmind/superhuman`. Figure 31 is the card for their eigenweights paper: the human asked for type A, then type C, then type D, and the model did the rest. Their report on the first First Proof batch uses the same style [FJhK⁺26].

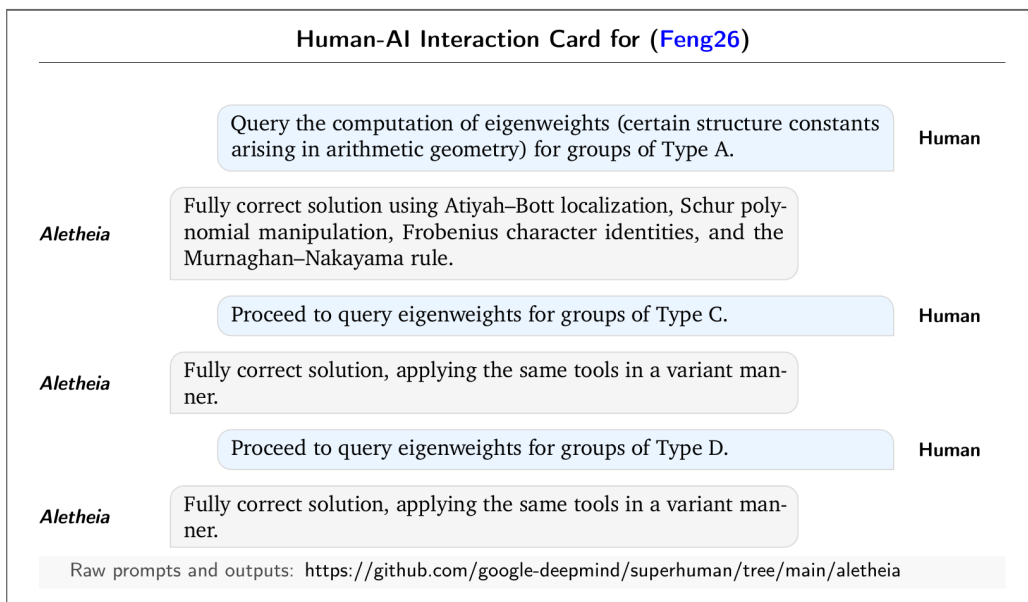


FIGURE 31. An interaction card [FTB⁺26, §6.2]. © Google DeepMind, CC BY-NC-ND 4.0.

The levels and cards are coarse and quick to read: for a whole result, how much the machine did and how much the result matters, with the detail in the published prompts and outputs. The pcf statement is finer grained: it goes finding by finding, names the model, harness, and setting for each, and adds the provenance search for each AI idea, negative results, the formalization’s axioms, and the cost. The two can be combined, with a level and a card at the top and the finding-by-finding record below.

APPENDIX D. HOW THIS DOCUMENT WAS MADE

This document was made in five steps, all with Claude Opus 5.5 (Anthropic) in the Claude Code harness [Ant26a], using the superpowers skills [Vin25] and my mathcity skills [Dup26e] `tdupu/mathcity`.

The first draft was written in September 2026 by a coordinating session with parallel subagents. The token usage of those sessions was not recorded, so no cost is given for it; this is a gap in the record, not a claim that the work was free. In October 2026 the document was extended with the material of the talk: eight parallel subagents each drafted one section from my talk materials, my notes, and the sources they cite, and a coordinating session merged them.

On 24 October 2026 eight independent reviewer agents, one per section, were told to assume the text wrong until a source showed otherwise. They checked 626 factual claims (numbers, prices, dates, quotations, attributions, and software behavior) against the cited sources, re-ran checks where possible, and found about 106 that needed correction; a coordinating session applied the corrections. The same day seven worker agents restructured the document for graduate students: six rewrote groups of sections from the fact-checked text, and one cleaned old titles out of the images. Their inputs were the fact-checked text, questions people had asked me by email (anonymized), and a sourced survey of public public posts about AI in mathematics.

On 25 October 2026 an agent running the `grill-with-docs` skill interviewed me about every section of the restructured text and recorded my decisions as they were made, 38 in all, together with my notes on the draft. The main decisions were to follow the order of the talk, to keep the text informational, to write prose rather than lists, to redraw diagrams in TikZ, and to show work at the

computer verbatim. Worker agents then rewrote the document in parallel from the fact-checked text, the decisions, and the talk’s slides and presenter notes, one group of sections each, while a figure worker drew the diagrams, and a coordinating session assembled the result and had it read again against the sources. In all, this round used 20 agents besides the coordinating session: research and editing agents during the grill, eight writers and a figure worker for the rewrite, and two final reviewers.

Step (October 2026)	Agents	Calls	Written	Read	Cost
Drafting from the lecture material	8	160	0.17M	14.9M	\$10.52
Fact-check	8	426	0.28M	44.9M	\$21.46
Restructure (incl. research and final read)	11	211	0.21M	20.1M	\$14.26
Coordinating session (all three steps)	1	177	0.14M	58.8M	\$18.30
Revision: agents (25 October)	20	729	0.54M	81.6M	\$40.85
Revision: coordinating session	1	173	0.13M	115.9M	\$31.83
Total		1,876	1.46M	336.2M	\$137.22

TABLE 17. Tokens used to make the October versions of this document.

Table 17 is summed from the session records on disk. Costs are at API list prices [Ant26n], though the work ran on a subscription; the columns are as in Table 9.

The agents drafted text from my notes and the talk, searched for and read sources, and prepared the bibliography. I chose the topics and the structure and revised the text. Every reference and quotation was checked against its source by an agent on 26 September 2026, and the factual claims were checked again on 24 October 2026 as described above. Most of this checking was done by agents; I have not checked every line myself. I am responsible for the content.

I thank Edgar Costa, from whom I learned much of this.

REFERENCES

- [ABH⁺26] Mohammed Abouzaid, Andrew J. Blumberg, Martin Hairer, Joe Kileel, Tamara G. Kolda, Paul D. Nelson, Daniel Spielman, Nikhil Srivastava, Rachel Ward, Shmuel Weinberger, and Lauren Williams, *First proof*, 2026.
- [Age] Agentic AI Foundation, *AGENTS.md: a simple, open format for guiding coding agents*.
- [Age26a] Agent Skills, *Agent skills overview*, 2026, Accessed 2026-10-24.
- [Age26b] ———, *Agent skills: Specification*, Open standard, originally developed by Anthropic, 2026, Spec repository and reference validator **skills-ref**: <https://github.com/agentskills/agentskills>.
- [Age26c] Agentic AI Foundation, *Projects*, 2026, Ten hosted projects as of 2026-10-24. Accessed 2026-10-24.
- [Ald26] Martin Alderson, *Watch out for cache read costs*, August 2026, Accessed 2026-10-24.
- [Ame] American Mathematical Society, Committee on Publications, *Use of artificial intelligence*, AMS journals policy, adapted from the COPE position statement (February 2023).
- [Anta] Anthropic, *How do i change my model improvement privacy settings?*, Anthropic Privacy Center, Updated 3 August 2026.
- [Antb] ———, *Is my data used for model training?*, Anthropic Privacy Center, Updated 18 August 2026.
- [Ant24] ———, *Introducing the Model Context Protocol*, November 2024, Accessed 2026-10-24.
- [Ant25a] ———, *Donating the Model Context Protocol and establishing the Agentic AI Foundation*, December 2025, Accessed 2026-10-24.
- [Ant25b] ———, *Equipping agents for the real world with Agent Skills*, Anthropic engineering blog, October 2025, Update of 18 December 2025: published as an open standard. Accessed 2026-10-24.
- [Ant25c] ———, *Updates to consumer terms and privacy policy*, August 2025.
- [Ant26a] ———, *Advanced setup – Claude Code docs*, 2026, Installation, system requirements, authentication.
- [Ant26b] ———, *Authentication*, Claude Code documentation, 2026, Section “Authentication precedence”. Accessed 2026-10-25.
- [Ant26c] ———, *Automate actions with hooks – Claude Code docs*, 2026, Hooks guide; full reference at <https://code.claude.com/docs/en/hooks>.

- [Ant26d] ———, *Claude plans and pricing*, 2026.
- [Ant26e] ———, *Connect Claude Code to tools via MCP*, Claude Code documentation, 2026, Gives `claude mcp add -transport http <name> <url>`, `claude mcp list`, and `/mcp`.
- [Ant26f] ———, *Context windows*, Claude Platform Docs, 2026, Accessed 2026-10-24.
- [Ant26g] ———, *Create custom subagents – Claude Code docs*, Claude Code documentation, 2026, “Each subagent runs in its own context window”; “returns only the summary”.
- [Ant26h] ———, *Effort*, Claude Platform documentation, 2026.
- [Ant26i] ———, *Explore the context window – Claude Code docs*, Claude Code documentation, 2026, Automatic compaction, `/compact`, `/clear`, `/context`, and what survives compaction.
- [Ant26j] ———, *Extend Claude with skills*, Claude Code documentation, 2026.
- [Ant26k] ———, *Formalizing Fermat’s Last Theorem*, September 2026, Code: <https://github.com/anthropics/fermats-last-theorem>. Accessed 2026-10-24.
- [Ant26l] ———, *How Claude remembers your project – Claude Code docs*, 2026, CLAUDE.md locations, `@imports`, native AGENTS.md support (v2.1.277+).
- [Ant26m] ———, *Manage costs effectively*, Claude Code documentation, 2026, Accessed 2026-10-24.
- [Ant26n] ———, *Pricing*, Claude Platform documentation, 2026.
- [Ant26o] ———, *Settings files and precedence – Claude Code docs*, 2026.
- [Ant26p] ———, *Terminal guide for new users – Claude Code docs*, Claude Code documentation, 2026.
- [Ant26q] ———, *Tool use with Claude*, Claude Platform Docs, 2026, The `tool_use` block (`type`, `id`, `name`, `input`) and the `tool_result` round trip.
- [Ant26r] ———, *What is the Max plan?*, Claude Help Center, 2026.
- [Ant26s] Benjamin Antieau, *Hexagon*, Blog post, October 2026, Accessed 2026-10-24.
- [App26a] Apple, *Mac Studio – technical specifications*, 2026.
- [App26b] ———, *MacBook Pro – technical specifications*, 2026.
- [Art26a] Artificial Analysis, *Kimi K3: Intelligence, performance & price analysis*, 2026, Kimi K3 (Max): Intelligence Index 44; open weights.
- [Art26b] ———, *LLM leaderboard: Comparison of AI models from OpenAI, Anthropic, Google, SpaceXAI & others*, 2026, Snapshot of 22 October 2026 (Intelligence Index v4.3.2, AA-Omniscience, CritPt); accessed 2026-10-24.
- [Art26c] ———, *Open weights models: Intelligence, performance & price*, 2026, Intelligence Index v4.3.2; 11 open-weight models listed.
- [Bac58] Francis Bacon, *Preparative towards a natural and experimental history (Parasceve ad historiam naturalem et experimentalem)*, *aphorism IX*, The Works of Francis Bacon (James Spedding, Robert Leslie Ellis, and Douglas Denon Heath, eds.), vol. 4, Longman, London, 1858, First published 1620 with the Novum Organum.
- [Bae26] John C. Baez, *Post on Mathstodon*, Mathstodon, September 2026.
- [BBCV21] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković, *Geometric deep learning: Grids, groups, graphs, geodesics, and gauges*, 2021.
- [BG20] Dorothy Bishop and Eoin Gill, *Robert Boyle on the importance of reporting and replicating experiments*, Journal of the Royal Society of Medicine **113** (2020), no. 2, 79–83.
- [Bir26] Chris Birkbeck, *mathlib-quality: a Claude Code skill plugin for bringing Lean 4 code up to Mathlib standards*, GitHub repository, 2026, MIT license. Version 0.73.3, commit 7146cacdeb08b44e1ce15d63a95b421a8861f28d, is the version `using-leanpowers` was ported from.
- [bla26] blazickjp, *arxiv-mcp-server: an MCP server for searching and reading arXiv papers*, GitHub repository blazickjp/arxiv-mcp-server, 2026, Accessed 2026-10-25.
- [BMR⁺20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, et al., *Language models are few-shot learners*, 2020.
- [Böl26] Can Bölük, *Oh my pi (omp)*, GitHub repository can1357/oh-my-pi, 2026.
- [Buc26] Tristan Buckmaster, *Statement [on finite-time blowup results with L. Alpöge and interactions with OpenAI]*, NYU Courant, September 2026.
- [ccu26] ccusage contributors, *ccusage: usage analysis for Claude Code and other coding agents*, GitHub repository, 2026, Accessed 2026-10-24.
- [Cla] Clay Mathematics Institute, *Riemann hypothesis*, Millennium Problems, status “Unsolved”.
- [Cla26a] Claude Code users, *Claude code defaults to API key billing despite active Max subscription*, GitHub issue #44669, [anthropics/claude-code](https://github.com/anthropics/claude-code), April 2026, Accessed 2026-10-24.
- [Cla26b] ———, *Env ANTHROPIC_API_KEY silently routed all sessions to API billing despite active Max subscription*, GitHub issue #78491, [anthropics/claude-code](https://github.com/anthropics/claude-code), July 2026, Accessed 2026-10-24.
- [CSOA25] Ben Cottier, Ben Snodin, David Owen, and Tom Adamczewski, *LLM inference prices have fallen rapidly but unequally across tasks*, Epoch AI, Data Insights, March 2025.

- [Dee24] DeepSeek-AI, *DeepSeek-V3 technical report*, 2024.
- [Dee26] ———, *DeepSeek-V4.1-Flash*, Hugging Face model cards deepseek-ai/DeepSeek-V4.1-Flash and deepseek-ai/DeepSeek-V4.1-Flash-1017, 2026, 552B backbone parameters, 8B (prefill) / 16B (decode) activated; 1M context; 890 bytes per token global KV cache; MIT license.
- [Del25] Den Delimarsky, *Spec-driven development with AI: Get started with a new open source toolkit*, The GitHub Blog, September 2025, Accessed 2026-10-24.
- [DG⁺25] DeepSeek-AI, Daya Guo, et al., *DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning*, Nature **645** (2025), no. 8081, 633–638, Published 17 September 2025. Preprint arXiv:2501.12948.
- [Die26] Thomas Dietert, *Slop is a skill issue: The engineering around coding agents*, Talk, MIT EECS/CSAIL Agentic Coding in Practice Seminar Series (host: Adam Chlipala; series organizer: Saman Amarasinghe), May 2026, Slides and video: <https://people.csail.mit.edu/saman/acpss/>.
- [DLTW26] Taylor Dupuy, Alon Levy, Thomas Tucker, and Zhuoran Wang, *PCF all the way down*, Draft, October 2026; section “Software and AI assistance” and appendix “Token accounting”, 2026.
- [Doc25] Cory Doctorow, *Enshittification: Why everything suddenly got worse and what to do about it*, MCD / Farrar, Straus and Giroux, New York, October 2025, Published 7 October 2025; UK edition Verso Books.
- [DPHZ23] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer, *QLoRA: Efficient finetuning of quantized LLMs*, 2023.
- [DTW26] Taylor Dupuy, Thomas Tucker, and Zhuoran Wang, *pcfawd-formalization: a Lean 4/Mathlib formalization of “PCF all the way down”*, GitHub repository, 2026, Created 2026-10-16; commit 5730ad1 of 2026-10-23 used here. Apache-2.0.
- [Dup26a] Taylor Dupuy, *agent-skills-public: MCP servers (stacks, scholar)*, GitHub repository tdupu/agent-skills-public, folder mcp, 2026, Accessed 2026-10-24.
- [Dup26b] Taylor Dupuy, *agent-skills-public: Some skills that are useful to mathematicians*, GitHub repository, 2026, Accessed 2026-10-24.
- [Dup26c] Taylor Dupuy, *[Intention]: ModularCurves — §4D strict-henselization universal property (08HQ)*, TauCe-tiRoadmap issue #1456, drafted by an AI agent, October 2026.
- [Dup26d] ———, *Issues #34–#36: Ollivier standard bases, partial differential algebra, and Mansfield differential Gröbner bases*, GitHub issues, repository tdupu/magma-diff-alg, September 2026, Opened by an agent 2026-09-11 02:11–02:12 UTC; issues 35 and 36 at the same address. Accessed 2026-10-25.
- [Dup26e] ———, *mathcity: a Gas City pack for mathematics research*, GitHub repository, 2026.
- [Epo26a] Epoch AI, *Epoch capabilities index (ECI)*, 2026, Scores from https://epoch.ai/data/eci_scores.csv, updated 25 October 2026: Claude Opus 5.5 168.83, GPT-6 Astra 168.81, GPT-6.1 Sol 168.37, Claude Fable 5.1 166.49, Gemini 4 Argon 166.45. Accessed 2026-10-25.
- [Epo26b] ———, *A presentation of the absolute Galois group of $\mathbb{Q}_2$* , FrontierMath: Open Problems, problem contributed by David Roe, 2026, Listed as solved (AI): first solution elicited by Roe with Claude Fable 5 on 2026-06-09; second by GPT-5.5 Pro in a harness by David Turturean, 2026-06-24; progress update 2026-07-06.
- [Fir26a] First Proof Project, *First proof on GitHub: repositories batch-1, batch-2, and batch-3*, GitHub organization 1stproof, 2026, batch-3 holds problems, human solutions, AI solutions, referee reports, raw outputs, submitted code, and costs. Accessed 2026-10-25.
- [Fir26b] ———, *Frontier AI systems solve up to 9/10 problems in First Proof benchmark; open-weight entrant solves none*, Third Batch press release, October 2026, Accessed 2026-10-24.
- [Fir26c] ———, *OpenAI prompt for System A (GPT-6 Astra, reasoning effort max, standard mode, web search enabled)*, Third Batch, 2026, Accessed 2026-10-25.
- [Fir26d] ———, *Third batch*, 2026, Lists the five systems tested: GPT-6 Astra, Anthropic Siege, IMProofBench ProofCouncil, MerLean, Humanize. Accessed 2026-10-24.
- [FJhK⁺26] Tony Feng, Junehyuk Jung, Sang hyun Kim, Carlo Pagano, Sergei Gukov, Cheng-Chiang Tsai, David P. Woodruff, Adel Javanmard, Aryan Mokhtari, Dawsen Hwang, Yuri Chervonyi, Jonathan N. Lee, Garrett Bingham, Trieu H. Trinh, Vahab Mirrokni, Quoc V. Le, and Thang Luong, *Aletheia tackles FirstProof autonomously*, Google DeepMind report, March 2026, Raw prompts and outputs: <https://github.com/google-deepmind/superhuman/tree/main/aletheia>. Accessed 2026-10-24.
- [FTB⁺26] Tony Feng, Trieu H. Trinh, Garrett Bingham, et al., *Towards autonomous mathematics research*, 2026, v1 10 Feb 2026, v3 6 Mar 2026; Google DeepMind.
- [G⁺24] Aaron Grattafiori et al., *The Llama 3 herd of models*, 2024.
- [Gas26] Gas Town Hall, *Gas city: orchestration-builder SDK for multi-agent coding workflows*, GitHub repository, 2026.
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT Press, Cambridge, MA, 2016.

- [GEB⁺24] Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, et al., *FrontierMath: A benchmark for evaluating advanced mathematical reasoning in AI*, 2024.
- [Gem25] Gemini Team, Google, *Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities*, 2025, Version 6, 19 December 2025.
- [GGW22] Joel Gibson, Georg Gottwald, and Geordie Williamson, *Machine learning for the working mathematician*, Seminar, University of Sydney, first half of 2022, 2022.
- [Git25] GitHub, *Spec kit: toolkit for spec-driven development*, GitHub repository, 2025, Accessed 2026-10-24.
- [Git26] Git project, *git-worktree: Manage multiple working trees*, Git documentation, 2026, Accessed 2026-10-24.
- [Goo25] Google DeepMind, *Advanced version of Gemini with Deep Think officially achieves gold-medal standard at the International Mathematical Olympiad*, Google DeepMind blog, July 2025, Accessed 2026-10-22.
- [Goo26a] Google, *Gemini apps limits & upgrades for Google AI subscribers*, Gemini Apps Help, 2026, Accessed 2026-10-25.
- [Goo26b] ———, *Gemini developer API pricing*, Google AI for Developers, 2026, Gemini 4 Argon \$2/\$10 per million tokens through 30 November 2026, \$4/\$20 from 1 December 2026; output price “including thinking tokens”. Accessed 2026-10-25.
- [Goo26c] ———, *Google AI plans*, 2026, AI Plus \$4.99, AI Pro \$19.99, AI Ultra from \$99.99 (and \$199.99) per month. Accessed 2026-10-25.
- [Goo26d] Google DeepMind, *Gemma 4*, Hugging Face model cards google/gemma-4-31B-it and google/gemma-4-26B-A4B-it, 2026, 31B: 30.7B parameters; 26B A4B: 25.2B total, 3.8B active; 256K context; Apache 2.0.
- [Goo26e] ———, *superhuman: Aletheia prompts and outputs*, GitHub repository, 2026, Accessed 2026-10-24.
- [Hex26] Hexagon Mathematics Foundation, *About Hexagon*, Website (public beta), 2026, Accessed 2026-10-24.
- [HMS⁺26] Thomas Hubert, Rishi Mehta, Laurent Sartran, et al., *Olympiad-level formal mathematical reasoning with reinforcement learning*, *Nature* **651** (2026), no. 8106, 607–613, Published 12 November 2025.
- [Hor25a] Dex Horthy, *Getting AI to work in complex codebases (advanced context engineering for coding agents)*, HumanLayer, GitHub repository, August 2025, Introduces “frequent intentional compaction” and the research-plan-implement workflow; recommends keeping context utilization in the 40–60% range.
- [Hor25b] ———, *No vibes allowed: Solving hard problems in complex codebases*, Talk, AI Engineer Code Summit, New York, November 2025; video uploaded 2 December 2025, November 2025, Source of the term “the dumb zone”.
- [HSW⁺21] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, et al., *LoRA: Low-rank adaptation of large language models*, 2021.
- [HTH25] Kelly Hong, Anton Troynikov, and Jeff Huber, *Context rot: How increasing input tokens impacts LLM performance*, Technical report, Chroma, July 2025, Accessed 2026-10-24.
- [Huc26] Grace Huckins, *What OpenAI’s latest controversy tells us about the future of math*, MIT Technology Review, September 2026, Accessed 2026-10-24.
- [IA26] Vasily Ilin and Justin Asher, *Lean Pool*, GitHub repository, 2026, Apache-2.0 license.
- [Ili26] Vasily Ilin, *Lean Pool: An AI-maintained archive of formalized mathematics*, September 2026.
- [Iso24] Phillip Isola, *Lecture 8: Transformers*, MIT 6.7960 Deep Learning, Fall 2024 (P. Isola, S. Beery, and J. Bernstein), MIT OpenCourseWare, 2024.
- [JWSM25] Daniel Jaroslawicz, Brendan Whiting, Parth Shah, and Karime Maamari, *How many instructions can LLMs follow at once?*, 2025, Introduces the IFScale benchmark; project page <https://distylai.github.io/IFScale>.
- [KMP⁺24] Eldar Kurtic, Alexandre Marques, Shubhra Pandit, Mark Kurtz, and Dan Alistarh, “give me BF16 or give me death”? accuracy-performance trade-offs in LLM quantization, 2024, ACL 2025.
- [L⁺24] Nathan Lambert et al., *Tulu 3: Pushing frontiers in open language model post-training*, 2024.
- [Lam26] Lambda, *AI cloud pricing*, 2026, On-demand instances, per GPU-hour.
- [Lea26a] Lean Community, *Contributing to Mathlib: what to contribute, and use of AI*, 2026.
- [Lea26b] ———, *Library style guidelines*, 2026.
- [Lea26c] ———, *Pull request review guide*, 2026.
- [Lea26d] Lean FRO, *Lean programming language*, 2026.
- [Lei26] *Leiden declaration on artificial intelligence and mathematics*, June 2026, Endorsed by the International Mathematical Union.
- [lla23] llama.cpp contributors, *Performance of llama.cpp on Apple Silicon M-series*, GitHub discussion ggml-org/llama.cpp #4167, 2023, Community-maintained table, updated through 2026.
- [LTL⁺25] Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, et al., *Goedel-Prover-V2: Scaling formal theorem proving with scaffolded data synthesis and self-correction*, 2025.
- [Mac26] MacTech, *Apple introduces new Mac Studio with M5 Max and M5 Ultra*, August 2026.

- [Mad26] David Madore, *Post on an AI-generated Lean “proof” of Collatz that exploited a kernel bug*, Bluesky, July 2026.
- [Mel] Herman Melville, *Moby dick; or, the whale*, Project Gutenberg eBook #2701.
- [mer25] merely-true contributors, *merely-true: a Lean project for AI-generated mathematics*, GitHub repository, 2025, Apache-2.0 license.
- [Met24] Meta, *Llama 3.2 3b*, Hugging Face model card meta-llama/Llama-3.2-3B, 2024, Accessed 2026-10-22.
- [Mod26] Model Context Protocol, *Model context protocol*, 2026, Accessed 2026-10-24.
- [Moo26a] Moonshot AI, *Kimi-K3*, Hugging Face model card moonshotai/Kimi-K3, 2026, 2.8T total, 104B activated parameters.
- [Moo26b] ———, *Kimi K3 license*, 2026, MIT-style; separate agreement for Model-as-a-Service operators above US\$20 million revenue; attribution above 100 million monthly users.
- [Mun11] Randall Munroe, *Standards*, xkcd, comic #927, July 2011, Image <https://imgs.xkcd.com/comics/standards.png>. Licensed CC BY-NC 2.5, see <https://xkcd.com/license.html>.
- [MYZ13] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig, *Linguistic regularities in continuous space word representations*, Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), 2013, pp. 746–751.
- [NNI93] National Academy of Sciences, National Academy of Engineering, and Institute of Medicine, *Responsible science: Ensuring the integrity of the research process, volume II*, National Academy Press, Washington, DC, 1993, The quoted passage is from a reprinted MIT report, “Report of the Committee on Academic Responsibility”, ch. 9.
- [Oll26a] Ollama, *Claude code*, Ollama documentation, integrations, 2026, Accessed 2026-10-24.
- [Oll26b] ———, *Cloud*, Ollama documentation, 2026.
- [Oll26c] ———, *deepseek-v4.1-flash*, Ollama model library, tags, 2026.
- [Oll26d] ———, *gemma4*, Ollama model library, 2026, Accessed 2026-10-24.
- [Oll26e] ———, *glm-5.3*, Ollama model library, tags, 2026.
- [Oll26f] ———, *glm-5.3-flash*, Ollama model library, tags, 2026.
- [Oll26g] ———, *gpt-oss*, Ollama model library, tags, 2026.
- [Oll26h] ———, *kimi-k3*, Ollama model library, tags, 2026.
- [Oll26i] ———, *Ollama*, 2026.
- [Oll26j] ———, *qwen3.6*, Ollama model library, tags, 2026.
- [Oll26k] ———, *qwen3.8*, Ollama model library, tags, 2026.
- [oOo26] oOo0oOo, *lean-lsp-mcp: Lean theorem prover MCP*, GitHub repository oOo0oOo/lean-lsp-mcp, 2026, Accessed 2026-10-24.
- [Ope24] Open Source Initiative, *The open source AI definition, version 1.0*, 2024.
- [Ope25a] OpenAI, *gpt-oss-120b*, Hugging Face model card openai/gpt-oss-120b, 2025, Apache-2.0.
- [Ope25b] ———, *gpt-oss-20b*, Hugging Face model card openai/gpt-oss-20b, 2025, Apache-2.0.
- [Ope26a] ———, *Advanced configuration – Codex*, ChatGPT Learn documentation, 2026, Custom model providers; -oss. Accessed 2026-10-24.
- [Ope26b] ———, *Build skills – Codex*, 2026, Formerly developers.openai.com/codex/skills.
- [Ope26c] ———, *Codex CLI*, ChatGPT Learn documentation, 2026, Formerly developers.openai.com/codex.
- [Ope26d] ———, *Codex CLI*, GitHub repository openai/codex, 2026, Apache-2.0; install instructions in README.
- [Ope26e] ———, *Custom instructions with AGENTS.md – Codex*, 2026, Formerly developers.openai.com/codex/guides/agents-md.
- [Ope26f] ———, *On the Navier–Stokes millennium prize problem*, September 2026, Announced 8 September 2026; updated 10 September 2026.
- [Ope26g] ———, *Our approach to advertising and expanding access to ChatGPT*, January 2026, Official page returned HTTP 403 to automated fetch; contents confirmed via Willison2026ChatGPTAds and news coverage.
- [Ope26h] ———, *Pricing*, OpenAI API documentation, 2026, Standard tier, prompts up to 272K tokens; accessed 2026-10-24.
- [Ope26i] ———, *Pricing – Codex*, 2026.
- [Ope26j] ———, *Reasoning models*, OpenAI API documentation, 2026.
- [Ope26k] ———, *tiktoken: a fast BPE tokeniser for use with OpenAI’s models*, GitHub repository openai/tiktoken, 2026, Version 0.14.0, encoding o200k_base. Accessed 2026-10-24.
- [Ope26l] ———, *“we congratulate Levent Alpöge and Tristan Buckmaster on their remarkable mathematical work. . .”*, Post on X (@OpenAI), September 2026, Accessed 2026-10-24.
- [Ope26m] OpenCode, *OpenCode: the open source coding agent*, GitHub repository anomalyco/opencode (formerly sst/opencode), 2026, MIT license.

- [Ope26n] OpenRouter, *AI model rankings*, 2026, Usage data through 21 October 2026; screenshot 22 October 2026; accessed 2026-10-24.
- [Ope26o] ———, *Benchmarks*, 2026, Screenshot 22 October 2026; accessed 2026-10-24.
- [Ope26p] ———, *Claude code integration*, OpenRouter documentation, 2026, Accessed 2026-10-24.
- [Ope26q] ———, *Model endpoints: per-provider pricing and quantization*, OpenRouter API, /api/v1/models/{model}/endpoints, 2026, Queried for deepseek/deepseek-v4.1-flash-1017, z-ai/glm-5.3, moonshotai/kimi-k3 and others.
- [OWJ⁺22] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al., *Training language models to follow instructions with human feedback*, 2022.
- [Pal26] Palomar Registry, *Palomar: a registry of verified formal results*, Website and GitHub organization, 2026, Source: <https://github.com/PalomarRegistry>.
- [Par26] Mayank Parmar, *Anthropic is cutting Claude Code’s current weekly limits by 17 percent*, BleepingComputer, August 2026, Accessed 2026-10-24.
- [PH22] Mary Phuong and Marcus Hutter, *Formal algorithms for transformers*, 2022.
- [Poc26] Matt Pocock, *grill-with-docs (SKILL.md)*, In the GitHub repository mattpocock/skills, 2026, Accessed 2026-10-24.
- [Qwe26a] Qwen Team, *Qwen3.5-397B-A17B*, Hugging Face model card Qwen/Qwen3.5-397B-A17B, 2026, Accessed 2026-10-23.
- [Qwe26b] ———, *Qwen3.6-35B-A3B*, Hugging Face model card Qwen/Qwen3.6-35B-A3B, 2026, 35B total, 3B activated; 262,144-token context; Apache-2.0.
- [Qwe26c] ———, *Qwen3.8-27B*, Hugging Face model card and config.json, Qwen/Qwen3.8-27B, 2026, 27.8B parameters (BF16); 262,144-token native context; 16 full-attention layers, 4 KV heads, head dimension 256; Apache-2.0.
- [Roe26] David Roe, *gq2-lean: a presentation of $g_{1/2}$ in Lean*, GitHub repository, 2026, Lean 4 and Mathlib; library free of `sorry` since 2026-07-26; nine literature axioms documented in `docs/literature-axioms.md`.
- [RSS⁺25] Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, et al., *DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition*, 2025.
- [RT26] David Roe and David Turturean, *A presentation of the absolute Galois group of $\mathbb{Q}_2$* , Project site with manuscript, interactive paper and two Lean formalizations, 2026, Manuscript version of 26 July 2026.
- [RWC⁺19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever, *Language models are unsupervised multitask learners*, OpenAI technical report, 2019.
- [Sha48] Claude E. Shannon, *A mathematical theory of communication*, Bell System Technical Journal **27** (1948), no. 3–4, 379–423, 623–656, Part 2: doi:10.1002/j.1538-7305.1948.tb00917.x. Section numbers as in the reprint at <https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf>.
- [Sha51] ———, *Prediction and entropy of printed English*, Bell System Technical Journal **30** (1951), no. 1, 50–64.
- [SMM⁺17] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean, *Outrageously large neural networks: The sparsely-gated mixture-of-experts layer*, 2017.
- [Sut19] Richard S. Sutton, *The bitter lesson*, Essay, Incomplete Ideas (blog), March 2019.
- [Sut26] Andrew V. Sutherland, *lmfdb-mcp: MCP connector that provides LLMs read-only SQL access to a mirror of the LMFDB*, GitHub repository AndrewVSutherland/lmfdb-mcp, 2026, Hosted at <https://mcp.lmfdb.org/mcp>; GPL-3.0. Accessed 2026-10-24.
- [Tao25a] Terence Tao, *Post on Mathstodon*, Mathstodon, October 2025.
- [Tao25b] Terence Tao, *Post on Mathstodon about a Lean proof supplementing a human proof, and literature reviews*, Mathstodon, October 2025, Accessed 2026-10-24.
- [Tao26] ———, *Palomar – a registry of Lean verified mathematics*, Blog post, *What’s new*, August 2026, Accessed 2026-10-24.
- [Thea] The LMFDB Collaboration, *The L-functions and modular forms database*.
- [Theb] The mathlib Community, *Mathlib: the math library of Lean 4*, GitHub repository, Apache-2.0 license.
- [The25] The Linux Foundation, *Linux foundation announces the formation of the Agentic AI Foundation (AAIF), anchored by new project contributions including Model Context Protocol (MCP), goose and AGENTS.md*, Press release, December 2025.
- [The26a] ———, *Agentic AI foundation announces four new projects: Agent Skills, Open Responses, OpenEnv, and OpenSandbox*, Press release, San Jose, October 2026, Accessed 2026-10-24.
- [The26b] The Tau Ceti Project, *Contributing to TauCetiRoadmap*, CONTRIBUTING.md and the “Coordinating work: intentions and claims” and “Porting existing work” sections of the README, 2026.
- [The26c] ———, *Tau Ceti agent coordination contract (v1)*, COORDINATION.md in the TauCeti repository, 2026.

- [The26d] ———, *Tau Ceti: an AIs-welcome Lean library downstream of Mathlib*, GitHub repository, 2026, Apache-2.0 license.
- [The26e] ———, *TauCetiRoadmap: human-controlled roadmaps for Tau Ceti*, GitHub repository, 2026.
- [Tur23] Richard E. Turner, *An introduction to transformers*, 2023, Version 6, 20 January 2026.
- [U.S24] U.S. National Science Foundation, *Award DMS-2401570: Intersection theory for differential equations (PI T. A. dupuy)*, 2024, Algebra and Number Theory / EPSCoR co-funding; 1 June 2024 – 31 May 2027.
- [use26] user202729, *Where should my formalization go? (flowchart)*, Excalidraw drawing posted on the Lean Zulip, with comments from Kim Morrison, Filippo A. E. Nuccio, Chris Birkbeck and Johan Commelin, 2026, Accessed 2026-10-23.
- [Vin25] Jesse Vincent, *superpowers: an agentic skills framework and software development methodology*, GitHub repository, 2025, MIT license.
- [Vos26] Laurie Voss, *Models got an order of magnitude better at following instructions in one year*, LinkedIn article, May 2026, Re-runs and extends the IFScale benchmark of Jaroslawicz et al. to $N = 10,000$ instructions.
- [VSP⁺17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin, *Attention is all you need*, 2017, Advances in Neural Information Processing Systems 30 (NeurIPS 2017).
- [Wil25] Simon Willison, *Agent skills*, simonwillison.net, December 2025, Update of 20 December 2025 on Codex. Accessed 2026-10-24.
- [Wok26] Wokey, *Claude usage limits (october 2026): Pro & max weekly limits, measured*, Wokey gateway; data from subscription accounts behind the gateway, 2026, Measured 12–19 October 2026 on 17 Max 20x and 474 Pro accounts; updated 19 October 2026. Accessed 2026-10-24.
- [Xia26] Xiaomi MiMo, *MiMo-V2.6-Pro-RL*, Hugging Face model card XiaomiMiMo/MiMo-V2.6-Pro-RL, 2026, 1.02T total, 42B activated parameters; 1M context; MIT license.
- [Yeg26] Steve Yegge, *Welcome to Gas Town*, Blog post, Medium, January 2026, Published 2026-01-01, last modified 2026-01-14. Figure 2, “The 8 Stages of Dev Evolution To AI”, is an image generated with Nano Banana.
- [Z.a26a] Z.ai, *GLM-5.3*, Hugging Face model card zai-org/GLM-5.3, 2026, GLM-5.3 license.
- [Z.a26b] ———, *GLM-5.3-Flash*, Hugging Face model card zai-org/GLM-5.3-Flash, 2026, 320B total, 18B active parameters; MIT license.
- [Z.a26c] ———, *GLM-5.3 license*, 2026, MIT-style; security review required for Model-as-a-Service operators above US\$10 billion revenue.
- [Zec26] Mario Zechner, *Pi: AI agent toolkit*, GitHub repository earendil-works/pi, 2026, MIT license; unified LLM API, agent loop, TUI, coding agent CLI.
- [Zef25] Maxwell Zeff, *Anthropic unveils new rate limits to curb Claude Code power users*, TechCrunch, July 2025, Weekly limits effective 28 August 2025.

DEPARTMENT OF MATHEMATICS AND STATISTICS, UNIVERSITY OF VERMONT

GAATI LABORATORY, UNIVERSITÉ DE LA POLYNÉSIE FRANÇAISE