# Sketchpad: certified exact arithmetic with a small vision-language model

Mostafa 'Neo' Mohsenvand
MIT Media Lab

31 October 2026

**Abstract**

Language models are accurate on most long-arithmetic problems, but their accuracy is estimated on samples and carries no guarantee for an unseen input; tool use is exact but moves the computation outside the network. We show that exactness at all input lengths becomes a finite, checkable property of a network when each of its steps reads only a bounded view of an external working memory. In Sketchpad, Qwen3.5-0.8B with a low-rank adapter executes arithmetic procedures on a rendered sheet. Each forward pass reads a three-cell view and emits one step: a digit, a carry or borrow, and the next operation. A fixed runtime moves the view and writes no digit of a result. We prove in Lean that the model is then exact on all inputs of any length if and only if it is exact on the finite set of views the runtime presents, that no finite test suffices when steps also read earlier steps, and that the guarantee tolerates bounded numerical perturbation and extends to compositions of procedures. All 2,446 views pass on the deployment device, a laptop GPU running the model in a web browser. The certified model performs signed addition and subtraction, long multiplication and division, square roots and greatest common divisors at arbitrary length. It solved 200 of 200 pre-registered problems in the browser, and 220 of 220 sums of up to 1,000 digits and products of up to 20 digits, of which GPT-6.1 Sol, the strongest of ten other models, solved 217. A symbolic runtime built on these procedures solves, expands and factors polynomials with every number computed by the model, and a multiplicative gate on the adapter recovers the base model exactly when the sheet is not in use.

## 1 Introduction

Language models now answer most long-arithmetic problems correctly, but this accuracy is measured on samples. Chain-of-thought prompting[1,2] and changes to positional encoding and data format[3–9] extend the lengths at which transformers add and multiply correctly. In each case, correctness on an unseen input remains an estimate, and the errors that remain are silent. In our comparison, GPT-6.1 Sol answered 217 of 220 problems correctly; its three errors, on sums of 300 and 1,000 digits, were completed answers well within the token budget, in the same form as the correct ones. Tool use[10–14] is exact, but places the computation outside the network.

A guarantee for all inputs requires that behaviour on unboundedly many inputs be fixed by finitely many facts. This holds when each step of a computation reads only a bounded view of an external memory and no other state passes between steps. The steps then range over a finite set of views, and a network that is correct on each view is correct on every run of any length. The observation is classical: it underlies Moore's analysis of experiments on finite automata[15], the verification of learned recursive programs[16] and a recent certificate for universal computation by language-model decoding[17]. It also has a converse. If a step may read the steps before it, as in chain of thought, no finite test suffices.

Sketchpad makes this hypothesis hold by construction for a pretrained vision-language model. A runtime lays out a problem on a rendered sheet and shows the model one view at a time: three cells of the sheet. In one forward pass, the model reads the view through its vision encoder and outputs the step's results and the next operation. The runtime writes the outputs and moves the view, and nothing else passes between steps (Fig. 1a). The procedures are the classical pencil-and-paper ones, and a low-rank adapter[18] trained on their 2,446 views teaches them to Qwen3.5-0.8B.

We prove in Lean that exactness on the views is equivalent to exactness at all lengths, and that the restriction to views is necessary (Section 3). We then certify the condition on the hardware that runs the model, a laptop GPU executing it in a web browser in 16-bit precision. All 2,446 views pass with a smallest margin of 3.02 logits, so the certificate tolerates any change of the logits smaller than 1.51. With these procedures the

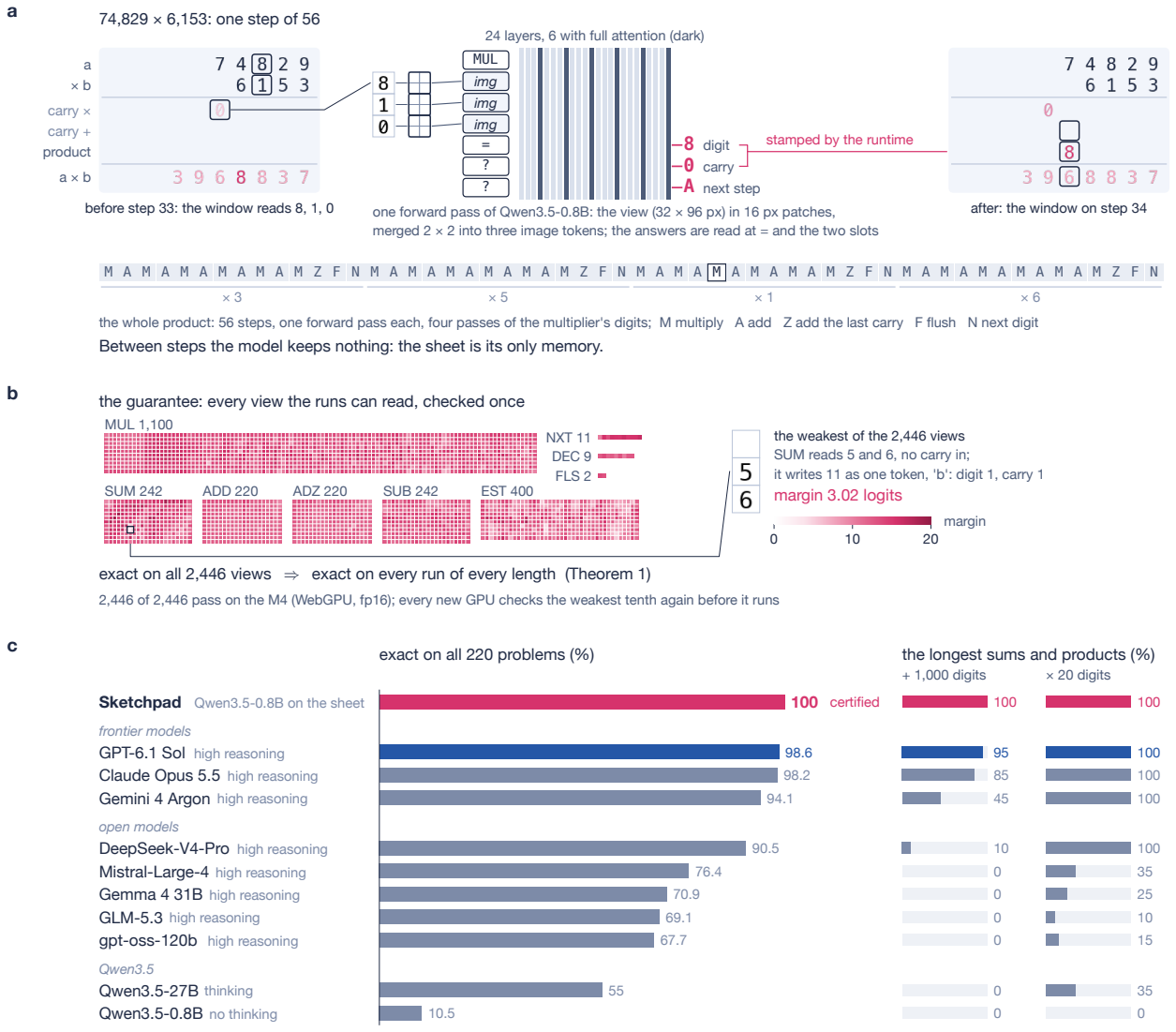



**Fig. 1 | Overview. a**, One step of the long multiplication 74,829 × 6,153 (step 33 of 56). The view (three cells, 32 × 96 pixels) enters Qwen3.5-0.8B as three image tokens after the operation token `MUL`; the outputs, read at `=` and two placeholder positions, are the digit, the carry and the next operation. Below: the operation sequence of the full product. **b**, Margins of all 2,446 views on the deployment device (Apple M4, WebGPU, 16-bit), grouped by table. The smallest margin, 3.02 logits, belongs to the addition view 5 + 6 with no incoming carry. **c**, Exact answers on the pre-registered comparison of 220 problems (sums of 10–1,000 digits and products of 2–20-digit numbers, 20 per length; Methods), overall and on the longest sums and products. Sketchpad in rose, the strongest alternative in cobalt; Qwen3.5-0.8B is shown without chain of thought, its better setting. Per-length results: Extended Data Fig. 1.

model adds, subtracts, multiplies and divides integers of any length and computes square roots and greatest common divisors. It answered all 220 problems of a pre-registered comparison in which the strongest of ten other models answered 217 (Section 4). Because certified procedures compose, a symbolic runtime built on them solves, expands and factors polynomials with every number computed by the model (Section 5). Finally, a scalar gate on the adapter restores the base model exactly, and a dialogue adapter learns to call the sheet when a reply needs exact computation; it stays within 2 points of the base model on instruction following and MMLU but loses 5.8 points on fresh grade-school word problems (Section 6). The guarantee covers the model's steps and is relative to the runtime, whose layouts, outer loops and reference procedures are conventional code; Section 3 states what they must satisfy.

## 2 The sheet machine

Sketchpad has three parts: a sheet that holds all state, a model that maps a view of the sheet to the outputs of one step, and a runtime that moves the view and writes the outputs. The design serves one requirement: the input of each step must come from a finite set that can be listed in advance.

**Model.** Qwen3.5-0.8B[19] combines a vision encoder with a 24-layer language model in which six layers use full attention and eighteen use Gated DeltaNet[20]. The only trained component is a rank-16 LoRA adapter on 150 projection matrices, with 10.2 million parameters (1.2% of the model). We call it the arithmetic adapter. Section 6 derives from it a dialogue adapter, and preliminary adapters appear only as contrasts. Device measurements were made on the GPU of an Apple M4 laptop, which we call the M4.

**Views.** The sheet is a grid of 32-pixel cells, each blank or a digit, drawn from a fixed glyph set. A view is three cells, rendered as a strip of $32 \times 96$ pixels that the vision encoder reads as three image tokens. Except in quotient estimation, the three cells form one column of the sheet. A step's prompt is an operation token, the view, `=` and two placeholder tokens, and the model's outputs are read at these three positions in one forward pass. The prompt contains nothing else. We call this the pinned-sheet policy; under it, the model's output is a function $f$ of the view alone.

**Tables.** For each operation, a table lists every view the operation can meet, with its correct outputs: column addition (`SUM`, 242 views), the five operations of long multiplication (1,553), subtraction (`SUB`, 242), estimation of a quotient digit (`EST`, 400) and decrement of an estimate (`DEC`, 9). With $\mathcal{O}$ the nine operations and $G$ the eleven glyphs, the rows form the set

$$V = \bigsqcup_{o \in \mathcal{O}} V_o, \qquad V_o \subseteq \{o\} \times G^3, \qquad |V| = 242 + 1{,}553 + 651 = 2{,}446, \tag{1}$$

whose terms count addition, multiplication and the calculator tables `SUB`, `EST` and `DEC` (Extended Data Table 1).

**Runtime.** For each procedure, a runtime of a few hundred lines lays out the operands, extracts views, calls the model and writes its outputs. It copies digits, as when a digit is brought down in division, but writes no digit of a result. In multiplication, the model's next-operation output determines the control flow. In division and square roots, the runtime fixes the outer loop and the model's outputs decide its branches. Independent procedures run in lockstep and share forward passes (Methods).

**Training.** We train the adapter directly on the tables, with cross-entropy, a hinge loss on the logit margin and a distillation penalty toward the base model on plain text[21] (Methods). Rows whose margin falls below 2 are replayed, and training stops when all rows have margins of at least 3. The tables are learned, not supplied by the base model. Without the adapter, Qwen3.5-0.8B passes 12 of the 2,446 views, all of them `SUM` views that copy a single digit; with eight solved views of the same table added to the prompt, it passes 158 of 2,380 (Methods).

## 3 Certification

Because each step is a function of its view, a check of the 2,446 views is a check of every run. This section makes the statement precise, shows that the restriction to views is necessary, and carries the check from exact arithmetic to the hardware.

At output position $k$ of a view $v$, let $z_k(v) \in \mathbb{R}^{\Sigma}$ be the model's logits over the vocabulary $\Sigma$ and $T_k(v)$ the correct token. The margin of $v$ is the smallest lead of a correct token over any other token,

$$m(v) = \min_k \Big( z_k(v)_{T_k(v)} - \max_{a \neq T_k(v)} z_k(v)_a \Big), \tag{2}$$

and $v$ passes if $m(v) > 0$. The model's output $f(v)$, the argmax at each position, then equals $T(v)$ under any tie rule. A certificate records $m(v)$ for all $v \in V$.

Let $C$ be the configurations of a procedure (sheet contents, view position and control state), $\mathcal{V} \supseteq V$ all views and $A$ the outputs. At configuration $c_t$, the runtime either halts, $\mathrm{view}(c_t) = \perp$, or presents the view $\mathrm{view}(c_t) \in \mathcal{V}$ and writes the outputs:

$$c_{t+1} = \mathrm{paint}\big(c_t,\ f(\mathrm{view}(c_t))\big). \tag{3}$$

The reference run uses the table $T\colon \mathcal{V} \to A$ in place of $f$.

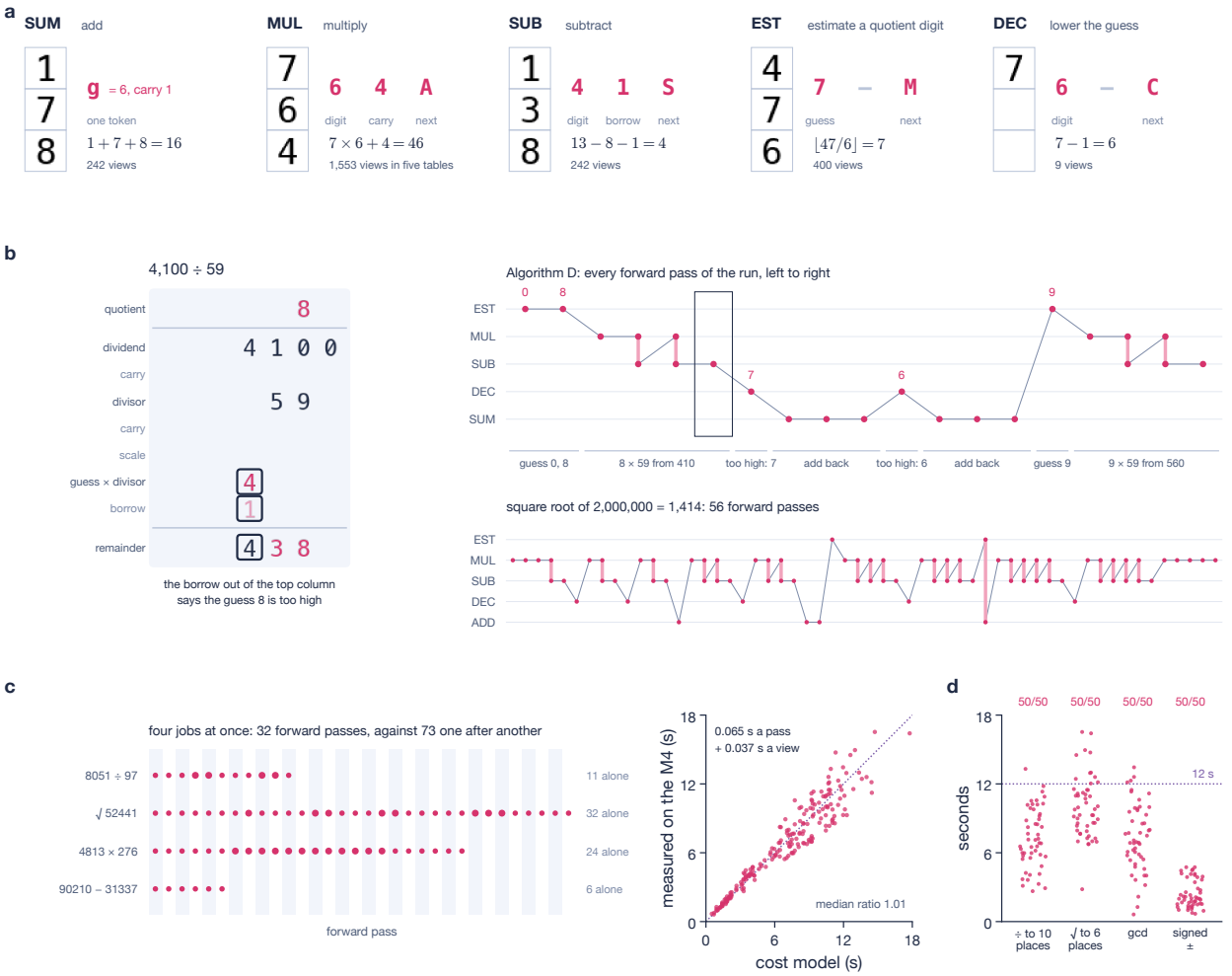



**Fig. 2 | The sheet machine.** **a**, One view and its outputs for each calculator table. **b**, Left: the division 4,100 ÷ 59; the borrow out of the leading column indicates that the estimate 8 is too high. Right: the operation sequence of the same division (Knuth's Algorithm D) and of the square root of 2,000,000 (56 forward passes). **c**, Batching four independent procedures: 32 forward passes instead of 73. Measured time against the cost model for the 200 problems of the browser test (median ratio 1.01). **d**, Time per problem in the browser test, all 200 exact.

**Theorem 1** (Exactness on views). *Let $R_T(I)$ be the set of views presented on reference runs from a set $I$ of initial configurations, and assume that* $\mathrm{paint}(c,\cdot)$ *is injective whenever* $\mathrm{view}(c) \neq \bot$. *The model's run from every $c_0 \in I$ equals the reference run at every step if and only if $f(v) = T(v)$ for all $v \in R_T(I)$. In particular, if $R_T(I) \subseteq V$ and $m(v) > 0$ for all $v \in V$, every run of every length is exact; this direction does not need injectivity.*

*Proof sketch.* If $f = T$ on $R_T(I)$ and the runs agree at $c_t$, then $\mathrm{view}(c_t) \in R_T(I) \cup \{\bot\}$ and (3) gives the same $c_{t+1}$; induction on $t$ completes sufficiency. Conversely, if the runs agree and $v = \mathrm{view}(c)$ for a configuration $c$ of a reference run, then $\mathrm{paint}(c, f(v)) = \mathrm{paint}(c, T(v))$, and injectivity gives $f(v) = T(v)$.

Theorem 1 separates what the network must get right from what the runtime must get right. The network's part, $f = T$ on $V$, is finite, and we certify it below. The runtime's part is that reference runs present only table views, $R_T(I) \subseteq V$, and compute the intended functions. The first follows from the layouts and normalisations of Section 4, and the runtime asserts it at each step; the second is the correctness of the classical procedures. Neither is formalised in Lean (Methods).

The restriction to views is necessary. If a step's context also contains earlier steps, as in chain of thought, no finite test suffices.

**Theorem 2** (Necessity of the pinned-sheet policy). *Suppose instead that each step's context retains the earlier steps. There is a two-state table and a family of decoders, each a correct lookup plus one attention head, such that for every finite set of test contexts some member passes every row of the table, and every*

*test context that begins with the prompt and ends in a row, with positive margins, yet errs on a longer run. Under the pinned-sheet policy the same member is exact at every length.*

*Proof sketch.* On a context with $n$ tokens after a sink token, $k$ of them events, the head writes $4Rk/(e^c+n)$ to the logit of the wrong state, where $R$ is the lookup's margin and $c$ the head's score on the sink. With $e^c = 3M+5$, for $M$ the length of the longest test context, the write exceeds $R$ only on longer contexts (Appendix A). This is an instance of Moore's result that no finite experiment identifies a machine with an unbounded number of states[15].

The theorems assume exact arithmetic, but the model runs in floating point, and the logits on the deployment path differ from those of any reference computation. The differences can be large: a 4-bit quantised decoder moved some margins of a preliminary adapter from $+9.4$ to $-8.9$. Margins make small differences harmless.

**Theorem 3** (Robustness to perturbation). *Let $z, \tilde{z} \in \mathbb{R}^O$ over a finite set $O$, and $y \in O$. If $\|z-\tilde{z}\|_\infty \le \tau$ and $z_y - \max_{a\neq y} z_a > 2\tau$, then $y$ is the unique maximiser of $\tilde{z}$. Hence, if $R_T(I) \subseteq V$, $m(v) > 2\tau$ for all $v \in V$, and at every step the logits on the execution path for each view $v \in V$ are within $\tau$ of $z_k(v)$ at every output position, every run from $I$ is exact.*

*Proof sketch.* If $a \neq y$ maximised $\tilde{z}$, then $\tilde{z}_a \le z_a + \tau < z_y - \tau \le \tilde{z}_y$. In a run, induction on $t$ as for Theorem 1: while the run agrees with the reference run, its view lies in $V$ and each output is the certified one. A cached-prefix implementation of the pinned step computes the logits of the full prompt exactly (Lemma 4), so it is a path with $\tau = 0$ in exact arithmetic.

We therefore certify on the deployment path itself. In 32-bit PyTorch, the smallest margin over the 2,446 views is 3.96. On the deployment path, ONNX Runtime Web on WebGPU[22–24] in 16-bit precision, all views pass, with smallest margins of 3.02 for addition, 6.52 for multiplication and 4.61 for the calculator tables (Fig. 3c). Per table, these differ from the server's smallest margins by less than one logit, whereas the 4-bit decoders of preliminary adapters deviate by up to 3 (Fig. 3d). The full check takes 3.5 minutes on the M4. By Theorem 3, a margin of 3.02 tolerates any change of the logits smaller than 1.51. The certificate scores one view per forward pass, whereas inference batches up to 32 views, and batching changes the 16-bit logits. With every view placed in random batches of 2, 4 and 32 for the arithmetic adapter, and of 4 and 32 for the dialogue adapter of Section 6, no logit moved by more than 1.09 and no margin by more than 0.64. The batched paths tested therefore satisfy Theorem 3 with $\tau = 1.09$, below half the smallest margin of 3.02. On a different GPU, the client recomputes the certificate on request; by default it recomputes the lowest-margin tenth of each table. Appendix A gives complete proofs and the corresponding Lean 4 theorems.

# 4 Arithmetic

Theorem 1 applies to a procedure only if every view it reaches is a table row. Signed addition, subtraction and schoolbook multiplication meet this by their layout, since each column holds two digits and a carry or borrow; the borrow out of the final column of a subtraction determines the sign. Division needs more, because a quotient digit is estimated from leading digits alone. Following Knuth's Algorithm D[25], both operands are first multiplied on the sheet by $\lfloor 10/(d_0+1) \rfloor$, where $d_0$ is the divisor's leading digit. The quotient is unchanged, and the scaled divisor $b$ has a leading digit $d$ of at least 5. For a partial remainder $r < 10b$ with leading digits $r_1 r_2$, `EST` returns

$$\hat{q} = \min\Big(9, \Big\lfloor \frac{10r_1 + r_2}{d} \Big\rfloor\Big), \qquad q \le \hat{q} \le q+2, \tag{4}$$

where $q = \lfloor r/b \rfloor$ is the true digit; the bounds are Knuth's Theorems A and B for base 10 and $d \ge 5$, and the inputs range over the 400 rows of the table. A borrow out of the multiply-subtract pass $r - \hat{q}\,b$ shows that the estimate is too large: `DEC` lowers it and the divisor is added back, at most twice.

Square roots use the digit-by-digit method on $c^2 a$, with $c \in \{1, 2, 4, 5\}$ chosen so that the root, if it has more than one digit, begins with 25 to 49. Twice the partial root then begins with a digit of at least 5, and the same estimation table serves every later digit (Methods). All scalings run on the sheet, and $\lfloor \sqrt{a} \rfloor = \lfloor \lfloor \sqrt{c^2 a} \rfloor / c \rfloor$. Greatest common divisors use Euclid's algorithm.

**Browser test.** We pre-registered 200 problems: 50 each of division to 10 decimal places, square roots to 6 places, greatest common divisors of integers up to 999, and signed sums and differences of up to 20 digits. Run one at a time on the M4, all 200 answers were exact. Median times were 7.0 s for division, 9.8 s for square roots, 7.0 s for greatest common divisors and 2.0 s for sums. Of the 200 problems, 184 met the pre-registered target of 12 s, and the slowest took 16.5 s (Fig. 2d; Extended Data Table 2). Measured times follow a cost model linear in forward passes and views, with a median ratio of 1.01 (Fig. 2c; Methods).

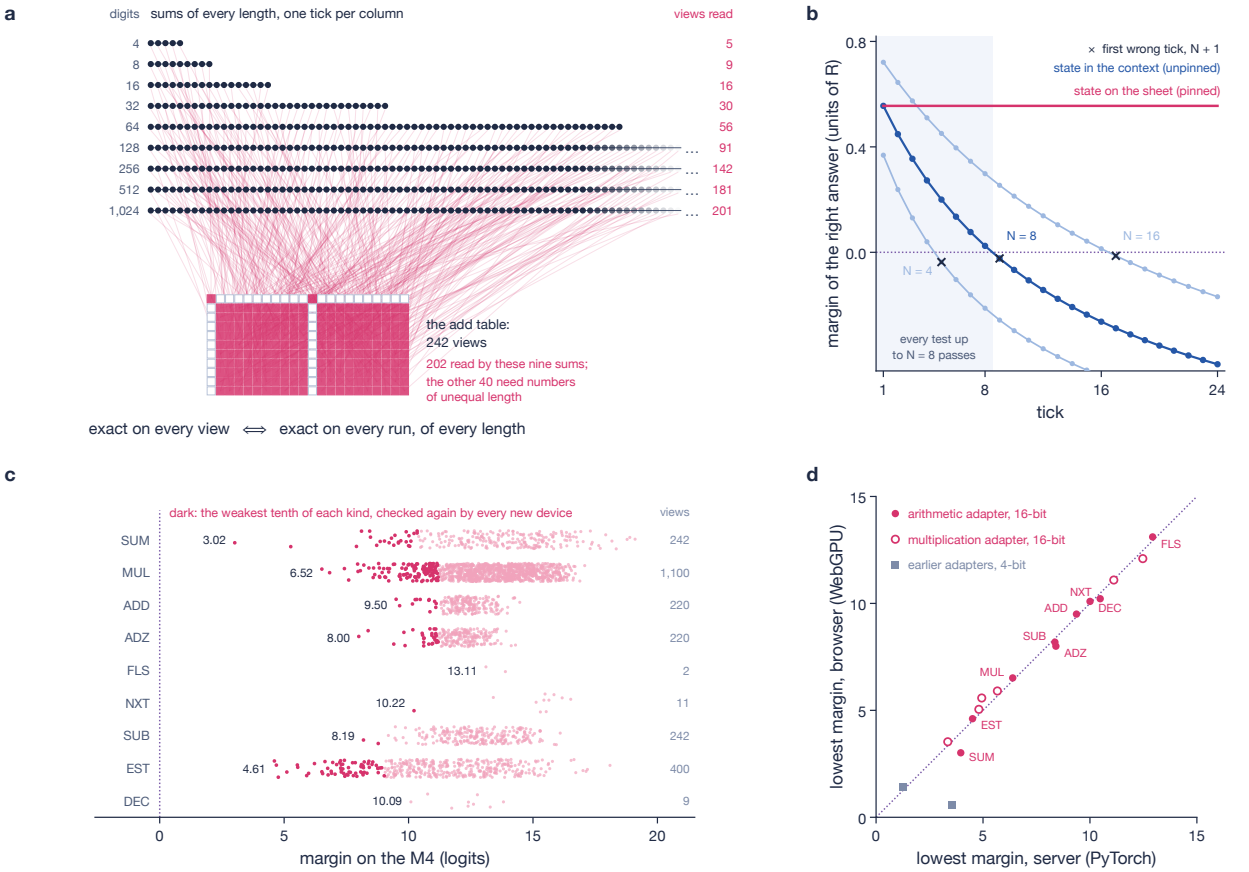



**Fig. 3 | Certification.** **a**, Sums of increasing length read only views from the 242-view addition table; nine sums of 4 to 1,024 digits reach 202 of them. **b**, The counterexample of Theorem 2: with state in the context, a decoder passes all tests up to $N$ steps and fails at step $N+1$ ($N = 4$, 8, 16; the sink score satisfies $e^c = 2N + 3$, Appendix A); under the pinned-sheet policy the margin is constant. **c**, Margins of all 2,446 views on the M4, by operation; dark points mark the lowest tenth, which is rechecked on each new device. **d**, Smallest margin per operation on the server (PyTorch, 32-bit) against the browser (WebGPU): 16-bit decoders lie on the identity line; 4-bit decoders of preliminary adapters deviate by up to 3 logits.

**Comparison.** We compared Sketchpad with ten language models on 220 pre-registered problems: sums of 10 to 1,000 digits and products of 2- to 20-digit numbers, 20 per length, with the same prompt and a budget of 32,768 output tokens (Methods). Sketchpad answered all 220. GPT-6.1 Sol, Claude Opus 5.5 and Gemini 4 Argon answered 217, 216 and 207, exact on all products, with all of their errors on sums of 300 or 1,000 digits. Five open-weight models answered between 149 and 199, failing mostly by exhausting the token budget on long sums and long products. Qwen3.5-27B answered 121, and Qwen3.5-0.8B, the base of Sketchpad, 23 without chain of thought and 3 with it (Fig. 1c; Extended Data Fig. 1 and Table 5).

# 5 Algebra

An algebraic derivation is a sequence of calls to certified procedures, whose inputs are transcribed from the problem and from earlier results. Its correctness then reduces to three conditions.

**Theorem 4** (Composition). *Consider $K$ calls with results $r = (r_1, \ldots, r_K)$, in which the input $x_i = q_i(r_{<i})$ of call $i$ depends only on the results $r_{<i} = (r_1, \ldots, r_{i-1})$ of earlier calls and the answer $g(r)$ only on $r$; let $q_i^*$, $T_i^*$ and $g^*$ be the intended inputs, functions and combining program, with results $r_i^* = T_i^*(q_i^*(r_{<i}^*))$. If every call returns its table's output on the input it received, $r_i = T_i(x_i)$, as Theorem 1 guarantees for a certified procedure, then a wrong answer requires, for some $i \le K$, a wrong table, $T_i(q_i^*(r_{<i}^*)) \neq T_i^*(q_i^*(r_{<i}^*))$, or a wrong transcription, $q_i(r_{<i}^*) \neq q_i^*(r_{<i}^*)$, or else a wrong combining program, $g(r^*) \neq g^*(r^*)$, and for any distribution of transcripts*

$$\Pr\left[g(r) \neq g^*(r^*)\right] \le \Pr[\text{some table wrong}] + \Pr[\text{some transcription wrong}] + \Pr\left[g(r^*) \neq g^*(r^*)\right]. \quad (5)$$

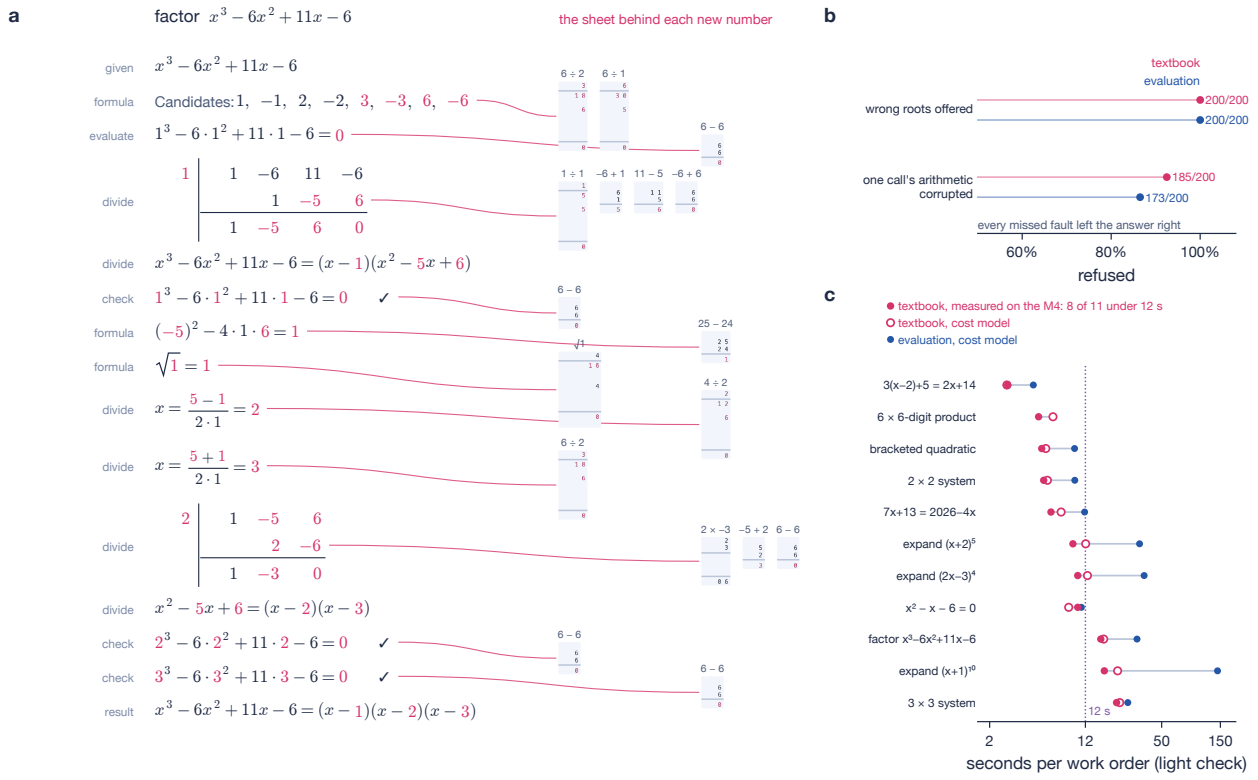



**Fig. 4 | Algebra. a**, Factorisation of $x^3 - 6x^2 + 11x - 6$. Numbers in rose were computed by the model; each is linked to the column operation that produced it. **b**, Rejection rates under fault injection: incorrect roots (200 of 200 for both routes) and one corrupted arithmetic result (185 of 200 for the textbook route, 173 of 200 for evaluation). **c**, Time per benchmark problem: textbook route measured on the M4 (rose) and under the cost model (open rose); evaluation route under the cost model (cobalt).

The certificate excludes the first failure. The other two, transcribing a problem into calls and combining their results, belong to the symbolic runtime, which is conventional code, and we guard them by checking each answer before it is returned.

The runtime follows the textbook procedures. It expands products and collects terms, solves linear equations by rearrangement and quadratics by the quadratic formula, expands powers by the binomial theorem with coefficients from Pascal's rule, factors by the rational root theorem and synthetic division, and solves linear systems by elimination (Fig. 4a). Irrational roots are returned as radicals with square factors removed, for example $3 \pm \sqrt{5}$. Rational roots are checked by substitution and irrational pairs by Vieta's relations in integer arithmetic, both exactly. Expansions are evaluated at $x = 1$, which detects any single wrong coefficient but not errors that cancel there (Methods). Under fault injection, the checks rejected 200 of 200 incorrect roots and 185 of 200 runs with one corrupted arithmetic result; in the other 15 runs the corruption did not change the answer (Fig. 4b).

Computer algebra reduces algebra to arithmetic in another way, by evaluation at points followed by interpolation[26] or Kronecker substitution[27]. Under the cost model, the textbook route is faster on ten of eleven benchmark problems and equal on the eleventh, a plain multiplication: 2.8 s against 4.5 s for a linear equation, 12.1 s against 33.1 s for $(x + 2)^5$ and 16.9 s against 31.5 s for factoring $x^3 - 6x^2 + 11x - 6$. Measured on the M4, eight of the eleven finish within the 12-s target; $(x + 1)^{10}$, a cubic factorisation and a $3 \times 3$ system take 16 to 22 s (Fig. 4c; Extended Data Table 3).

# 6 Gating

Fine-tuning for arithmetic damages other abilities. A preliminary arithmetic adapter changed perplexity on plain text by only 0.65%, yet it lowered strict instruction-following accuracy on IFEval[28] by 16.4 points, an instance of catastrophic forgetting[29,30]. We therefore multiply every adapter branch by a scalar graph input

$g$:

$$y = xW + g\, s\, (xA^\top)B^\top, \qquad y\big|_{g=0} = xW, \tag{6}$$

where $W$ is the frozen weight, $A$ and $B$ are the adapter matrices and $s$ is the LoRA scale (Fig. 5a). At $g = 0$ each branch contributes an exact floating-point zero, provided it is finite, so the gated model is the base model; we confirmed this bit for bit for logits, hidden states and cached states (Methods). At $g = 1$, the certificate on the M4 reproduces all 2,446 margins.

The gate is set by the runtime, not by the model. Switching it within a reply, so that the base model writes the prose and the adapter only the sheet's program, leaves the decision to compute with the base model, whose recall is 0.809 (Methods). We therefore keep $g = 1$ throughout a conversation and trained the dialogue adapter, starting from the arithmetic adapter, to decide by itself: it emits a short program for the sheet when a reply requires exact computation; generation then pauses, and the runtime executes the program and returns its results into the reply (Methods). Training combines supervised data with self-distillation[21] and on-policy distillation[31,32] toward the gated-off model, which limits forgetting[33]. Reinforcement learning with GRPO[34] follows, against an execution-based reward and with a KL penalty toward the supervised checkpoint, and every stage keeps the table objective.

Against criteria fixed before training (Methods), the dialogue adapter invokes the sheet on held-out phrasings with precision 0.994 and recall 0.989, against 0.971 and 0.809 for the base model prompted with four examples. It writes the correct computation for 92.8% of word problems, 96.7% of equations and 98.3% of expressions embedded in prose. On the benchmark items used during development, it stayed within 1.6 points of the base model on IFEval ($-1.5 \pm 2.0$), GSM8K[35] ($-1.6 \pm 3.4$) and MMLU[36] ($-1.4 \pm 1.6$), each less than one standard error from the criterion of $-2$. Because it was the fifth candidate scored on those items, we rescored it on items not used before, against predictions fixed in advance. MMLU held, at $-2.0 \pm 1.1$ on 1,000 questions, but GSM8K did not: $-5.8 \pm 1.7$ on the 1,069 remaining test problems, so the retention criterion is not confirmed for multi-step word problems. The loss appears both where the adapter calls the sheet and where it answers directly, which places it in multi-step reasoning rather than in the sheet. The same weights at $g = 1$ execute the arithmetic procedures: on the M4, all 2,446 views pass, with smallest margins of 6.67 for addition, 6.53 for multiplication and 4.56 for the calculator tables. Across recipes, IFEval and GSM8K trade off against each other (Fig. 5b). On-policy distillation removes most of the trade-off, and reinforcement learning reintroduces it unless anchored to the supervised checkpoint (Extended Data Table 4).

# 7 Mechanism

The certificate establishes what the model outputs on each view, not where in the network the output is decided. We applied the logit lens[37,38] to the arithmetic adapter at the first output position: with $h_\ell$ the residual state after depth $\ell$, and $W_U$ and norm the final unembedding and normalisation,

$$\hat{p}_\ell = \mathrm{softmax}\left(W_U\, \mathrm{norm}(h_\ell)\right). \tag{7}$$

The correct output becomes and remains the top token at a median depth of 19 for addition, 20 for multiplication and 18 for the addition steps of the multiplier; up to depth 15 it is the top token for at most 13% of views in any of these tables (Fig. 6a). At the final layer, states cluster by output, and sums with a carry (10–19) separate from those without (Fig. 6b).

To test whether this structure is causal, a class-mean intervention[39,40] moves the state of a view with output $a$ toward a target output $a'$,

$$h_\ell \leftarrow h_\ell + \mu^\ell_{a'} - \mu^\ell_a, \tag{8}$$

where $\mu^\ell_a$ is the mean of $h_\ell$ over the table's views with output $a$. Applied after depth 20, the end of the fifth full-attention layer, it changes the output to $a'$ in 79 of 80 addition steps and 80 of 80 multiplication steps; applied at any depth up to 15, it changes none (Fig. 6c). For multiplication the transition is abrupt: no step changes at depth 19 and every step at depth 20, so the digit is decided in the fifth full-attention layer. For addition it is gradual, with 31 and 38 of 80 steps changed at depths 18 and 19, and the same layer completes a decision that the preceding layers have begun, consistent with the median decision depth of 19 under the logit lens.

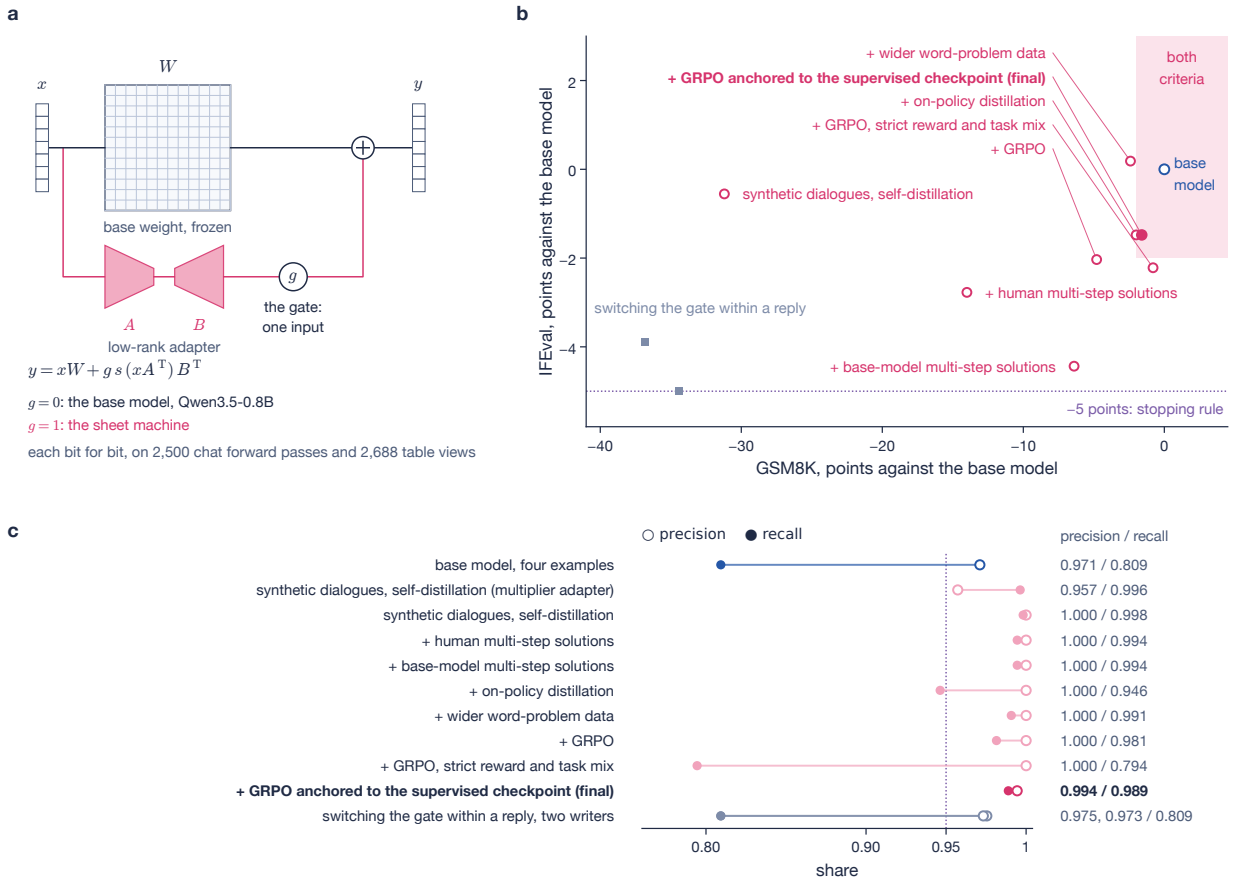



**Fig. 5 | Gating.** **a**, A scalar input $g$ multiplies every adapter branch; at $g = 0$ the model is identical to the base model, at $g = 1$ it executes the sheet procedures. **b**, IFEval against GSM8K, in points relative to the base model on the development items, for the dialogue fine-tunes and for switching the gate within a reply (on fresh items the dialogue adapter's GSM8K difference is −5.8); the shaded region satisfies both criteria, and the dotted line at −5 points is the pre-registered stopping rule. **c**, Precision and recall of invoking the sheet on held-out problems: base model with four examples (cobalt) and fine-tunes (rose).

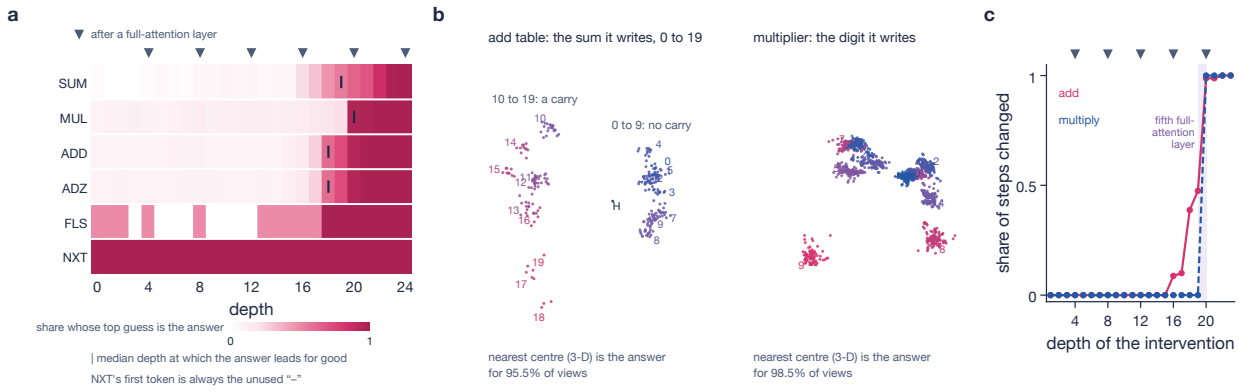



**Fig. 6 | Mechanism.** Arithmetic adapter. **a**, Logit lens: fraction of views whose top token at each depth equals the final output, by operation; ticks mark the median decision depth, triangles the ends of full-attention layers. **b**, Final-layer states projected onto three discriminant directions, coloured by output, for addition (0–19) and multiplication digits; nearest-centre shares are measured in this projection (Methods). **c**, Fraction of 80 steps whose output a class-mean intervention changes to its target, against the depth at which it is applied, for addition and multiplication; the shaded band is the fifth full-attention layer.

# 8 Discussion

A pretrained vision-language model of 0.8 billion parameters, restricted to a bounded view of its working memory, can be certified exact on inputs of every length by checking 2,446 views on the device that runs it. On the longest sums of our comparison, where the strongest frontier models made silent errors, its answers carry this guarantee, and the same certified procedures support symbolic algebra and sit behind an exact gate in a dialogue model.

The certificate covers the model's steps. The runtime that sequences them, the transcription of a problem into calls and the translation of a word problem into a computation are conventional code or uncertified model behaviour; the dialogue adapter mistranslates about 7% of held-out word problems, and on fresh grade-school problems it reasons less well than the base model, 5.8 points below it on GSM8K. In algebra, the answer checks guard these parts without being complete: they verify the roots found, not that all roots were found, and evaluation at a point is a spot check. The guarantee holds for the execution path on which the margins were measured, and on a new GPU it requires the full check, of which the client by default repeats only the lowest-margin tenth. Inputs must be rendered in the model's glyph set, the algebra is limited to polynomials of low degree in one variable and linear systems of up to three equations, and a step takes about 0.1 s on a laptop GPU.

Theorem 2 bears on methods that estimate accuracy on samples. When a step reads the steps before it, as in chain of thought and in most formats studied for length generalisation, no finite test certifies the model, however long the tested inputs. Context disciplines such as the inductive scratchpad[41] and PENCIL[42] bound what a step reads in order to generalise. The pinned-sheet policy, like the decoding of Lewandowski et al.[17], is the limit in which a step reads only a bounded view, and Theorem 1 shows that in this limit a finite check suffices. The price is a runtime that sequences the steps, and the gain is a guarantee that does not depend on which inputs were tested.

# Methods

## Model and adapter

The language model of Qwen3.5-0.8B[19] has 24 layers with hidden size 1,024; layers 4, 8, …, 24 (numbered from 1) use full attention (8 query and 2 key-value heads), the others Gated DeltaNet[20]. The LoRA adapter (rank 16, scale 2) is applied to the query, key, value and output projections of the 6 attention layers (24 matrices), the two input projections and the output projection of the 18 Gated DeltaNet layers (54) and the three projections of each of the 24 feed-forward blocks (72): 150 matrices and 10,223,616 parameters, against 873,438,784 in the base model. For deployment the model is exported to ONNX, with a 16-bit vision encoder and language model and 4-bit token embeddings (16-bit, looked up inside the decoder graph, for the dialogue adapter); the adapter is folded into the decoder graph. Two additional graph inputs implement the gate and the steering intervention, and one additional output returns hidden states.

## Views, prompts and tables

Glyphs are fixed $32 \times 32$ sprites of blank and the ten digits, so the set of possible views is finite and each view is an exact image. The prompt of a step is its operation token, the $32 \times 96$ image, `=` and two placeholder tokens, padded with further placeholders to a common length. Since the model is causal, padding after the output positions does not affect the outputs (Lemma 4), so steps of different operations can share a batch. The tables are exhaustive over each operation's views; for example, `MUL` covers every multiplicand digit or blank, every multiplier digit and every carry (blank, 0–8; $11 \times 10 \times 10 = 1{,}100$ views). The view of `EST` is assembled from three cells that do not share a column: the leading remainder digit $r_1$, the next remainder digit $r_2$ and the divisor's leading digit $d$, in the column of $r_2$. A blank $r_1$ is written as 0 before the view is read, so `EST` covers $r_1 \in \{0, \dots, d\}$, $r_2 \in \{0, \dots, 9\}$ and $d \in \{5, \dots, 9\}$, $10 \sum_{d=5}^{9} (d+1) = 400$ views; `DEC` reads the guess and two blank cells. That the procedures reach no other view follows from the layout and, for division and square roots, from normalisation; the runtime checks it at every step, and a reference implementation checked it on 1,000 random problems per procedure. No view outside the tables occurred in any experiment.

## Training

Each training step draws a batch $R$ of rows from every table, together with rows replayed from the last certificate. With $z_k(v;\theta)$ the logits at output position $k$ of row $v$, $K_v \in \{1,3\}$ the number of output positions and $m_k(v;\theta)$ the margin at position $k$, the term inside the minimum in (2), the loss is

$$\begin{aligned}\mathcal{L}(\theta) = \frac{1}{|R|}\sum_{v\in R}\frac{1}{K_v}\sum_{k=1}^{K_v}\Big[-\log\operatorname{softmax}\big(z_k(v;\theta)\big)_{T_k(v)} + \max\big(0,\, h - m_k(v;\theta)\big)\Big] \\ + \frac{\lambda}{n}\sum_{j=1}^{n}\mathrm{KL}\big(p_0(\cdot \mid x_{\le j}) \,\big\|\, p_\theta(\cdot \mid x_{\le j})\big),\end{aligned} \tag{9}$$

with hinge target $h = 4$ and $\lambda = 1$, where $x$ is a passage of $n = 128$ tokens of plain text and $p_0$ the base model[21]. Every 100 steps the full certificate is computed and rows with margins below $h/2 = 2$ are replayed until the next; training stops when every row has a margin of at least 3. The arithmetic adapter was initialised from a preliminary adapter certified on the addition and multiplication tables, with the subtraction, estimation and decrement tables added. It passed every row at step 1,200 and was stopped at step 1,600 (19.8 minutes on one H100 GPU; learning rate on a cosine schedule from $2 \times 10^{-4}$ to $2 \times 10^{-5}$ over 4,000 steps). Its plain-text perplexity is 1.0048 times that of the base model.

## Certification

Margins are computed over the full vocabulary of 248,320 tokens. The server certificate uses PyTorch in 32-bit floating point. The device certificate runs the deployed files (transformers.js 4.2.0[23], ONNX Runtime Web on WebGPU) through the inference code path and batched prompt form, and records every margin with the device, library version, file hashes and glyph set (Appendix B). Padding the addition prompt to the batched form changed no margin by more than 0.0018. Two device certificates, with and without the gate, agree on all margins to $10^{-3}$. The certificate is bound to the model files by their hashes (Appendix B). The batch check scores each of the 2,446 views alone and inside a batch of randomly chosen other views, in the padded prompt form of inference, and records the largest change of any logit over the full vocabulary and of the view's margin. For each batch size, one random partition covers all views once: sizes 2, 4 and 32 for the arithmetic adapter, and 4 and 32 for the dialogue adapter. The check samples batch compositions; it does not bound all of them.

The base-model control scores the same 2,446 views with the base model, in 32-bit PyTorch and the prompt form of the server certificate. Zero-shot, it passes 12 views, all of them `SUM` views, and 1.1% of output positions are correct. With eight solved views of the same table (fixed seed), each followed by its outputs, prefixed to every query, it passes 35 of the 2,380 views not among the examples, with outputs read at the placeholder positions, and 158 when each output after the first is read after the correct earlier outputs, the form the examples show (45.8% of positions); no table passes completely in any setting.

## Procedures and timing

Independent procedures, such as the products of an algebra step, run in lockstep, with one view from each in a shared batch. On the M4, a computation of $n_{\text{pass}}$ forward passes over $n_{\text{view}}$ views takes

$$t \approx \alpha\, n_{\text{pass}} + \beta\, n_{\text{view}}, \qquad \alpha = 0.065 \text{ s}, \quad \beta = 0.037 \text{ s}, \tag{10}$$

so batching reduces $n_{\text{pass}}$ and leaves $n_{\text{view}}$ unchanged.

Subtraction ends on a row that reports a final borrow, in which case the operands are swapped and the sign recorded. In division and square roots, a multiply-subtract pass of the divisor (or trial divisor) by a guess runs `MUL` one column ahead of `SUB`, two views per forward pass, with separate rows for the carry and the borrow. The runtime checks each next-operation output against the pass (`A` within it, `Z` at its end for `MUL`, `S` for `SUB`), and the borrow `SUB` writes out of the top column is the branch: no borrow keeps the guess, a borrow lowers it with `DEC`. In division the divisor is then added back with `SUM`, and the carry out of the top column, which cancels the borrow, decides whether to lower again; in square roots the pass is repeated from the saved remainder. A guess of 0 from `EST` comes with the next-operation `N` and skips the pass. In square roots, the first root digit is at most 4, and when it is 2 the second is at least 5; the runtime starts these at 4 and 9, the largest values the scale allows, so that the first is lowered at most twice and the second at most four times, and every other digit at the estimate of `EST`. Each root digit is thus fixed by the model's multiply-subtract

and `DEC` steps; the runtime copies a lowered digit into the trial divisor, and a carry out of the trial divisor's last digit is added to the next column with `ADD`, whose running-total cell is then blank. Greatest common divisors repeat division with remainder until the remainder the model writes is zero. In square roots, the trial divisor for the next digit, twice the partial root, is formed on the sheet by adding the last digit to the previous trial divisor with `ADD`.

For square roots, write $a = A \cdot 100^k$ with $1 \le A < 100$. The runtime takes $c = 4$ if $A < 1.5625$, $c = 2$ if $A < 6.25$, $c = 1$ if $A < 25$ and $c = 5$ otherwise, comparing the first six digits of $a$, aligned to pairs, with 15625, 62500 and 250000. Then $c\sqrt{A}$ lies in $[2.5, 5)$ if $A < 25$ and in $[25, 50)$ otherwise, so the root of $c^2 a$ begins with 25 to 49 when it has more than one digit and is a single digit from 2 to 4 otherwise; we also verified this for every radicand from 1 to 999,999. Division scales by $f = \lfloor 10/(d_0 + 1) \rfloor \in \{1, 2, 3, 5\}$, and $f(d_0 + 1) \le 10$ keeps the scaled divisor at the same length. The scalings use the multiplication procedure; the remainder of a division is unscaled by multiplying by $10/f$ and dropping the last digit ($f = 2, 5$) or by an exact division by 3, and the root by multiplying by $10/c$ or $100/c$ and dropping one or two digits. The runtime's choice of scale factors and initial guesses affects the number of steps but not the result, since every guess is at least the true digit and is lowered only by certified steps. The cost model was fitted on batches of 1 to 32 views on the M4.

The browser test drew its 200 problems from a fixed generator before any run: $a \div b$ to 10 places with $a \le 999$ and $2 \le b \le 99$; $\sqrt{a}$ to 6 places with $2 \le a \le 999$; $\gcd(a, b)$ with $10 \le a, b \le 999$; and signed sums and differences of 1 to 20 digits. It ran in Chrome on an Apple M4, one problem at a time, with independent views of the same problem sharing a forward pass; every view was a table row. Answers were checked against arbitrary-precision arithmetic. The test was repeated with the gated adapter, with identical views and outputs.

## Algebra

The algebra runtime parses a problem, constructs the derivation and obtains every number through the calculator interface; independent calls are batched. Each line of the derivation records the model calls that produced its numbers. The default check substitutes rational roots into the collected equation and evaluates expansions at $x = 1$; the strict check substitutes into the original expressions and evaluates expansions at $x = -1, 0, 1, 2$. For equations, both modes also confirm the collection of terms by evaluating each side before and after collection at $x = y = z = 1$; this does not detect a coefficient moved between variables, which the strict mode's substitution into the original equations does. Evaluation at fixed points is a spot check: an incorrect expansion of degree $D$ passes if the difference from the correct one vanishes at every checked point, which a single wrong coefficient cannot achieve at $x = 1$. For an irrational pair, the runtime removes square factors from the discriminant by trial division by the primes up to 47 and divides $p$, $q$ and $d$ by their greatest common divisor, so the radical is fully reduced when the discriminant is below $53^2 = 2{,}809$. The pair, written $(p \pm q\sqrt{m})/d$ with an integer $p$ and positive integers $q$, $m$ and $d$, is accepted only if

$$2ap = -bd \quad \text{and} \quad a\,(p^2 - q^2 m) = c\,d^2, \tag{11}$$

which are Vieta's relations $x_1 + x_2 = -b/a$ and $x_1 x_2 = c/a$ multiplied by $ad$ and $ad^2$; for $a \neq 0$ they hold if and only if the pair are the two roots. The check concerns the values of the roots; that the form is reduced rests on the reduction above. Fault injection planted an incorrect root (200 problems per route) or corrupted one arithmetic result (200 per route), with fixed seeds, and the rejection rates are the same in both check modes; a further 200 perturbed radical pairs were all rejected. The reference test suite (264 tests, 2,200 generated problems per route and check mode) uses exact tables in place of the model.

## Gating

The gate multiplies each adapter branch before it is added to the frozen projection. At $g = 0$ the branches are still evaluated, which adds the adapter's 1.2% of parameters to each forward pass. The identity test compared the base, ungated and gated graphs bit for bit under ONNX Runtime 1.31.1 (the Python package) on CPU, on 100 chat prompts (first user turns of the No Robots test split[43]; one prefill and 24 greedy steps each, 2,500 forward passes) and on 2,688 table views: the 2,446 views and the 242 addition views again in the padded prompt form. Every logit, hidden state and cached state of the gated graph equals that of the base model at $g = 0$ and that of the ungated adapter at $g = 1$, and greedy completions of the 100 chat prompts are identical.

## Dialogue fine-tuning

The training data comprise about 3,150 synthetic dialogues, about 30% of which should not invoke the sheet, and 547 problems from the GSM8K training split, solved by the base model with each calculation replaced by a call to the sheet. A call is a Markdown code block labelled `sheet`, with one calculator expression or equation per line; generation stops when the block closes, the runtime executes the block and appends one result line per expression, and generation resumes, with at most two blocks per reply. Every training step, in the supervised, distillation and reinforcement-learning stages alike, includes a batch of table rows under the objective (9), with low-margin rows replayed, and every stage was required to end with all 2,446 rows passing. The held-out set contains 780 items from unseen templates. Invocations are scored by execution. Criteria were fixed before training: precision and recall of invocation of at least 0.95; correct invocations for at least 90% of word problems and 95% of equations; IFEval, GSM8K (250 test problems) and MMLU (500 test questions; both subsets drawn with seed 0) within 2 points of the base model. GSM8K is evaluated at $g = 1$ with the sheet in the loop; in these evaluations, and in the reward, the blocks are executed by a reference implementation of the procedures in exact integer arithmetic rather than by the model, which by Theorem 1 returns the same results. The dialogue adapter was the fifth candidate scored on the same benchmark items. We therefore rescored it and the base model on the 1,069 GSM8K test problems and 1,000 MMLU test questions not used before, with predictions and a stopping rule fixed in advance (GSM8K $-1.5$ within $[-4.5, +1.5]$; MMLU $-1.0$ within $[-3.0, +1.0]$; failure below $-2$ by more than one standard error). GSM8K fell outside its band: 0.491 against 0.549 (139 problems gained, 201 lost). The adapter called the sheet on 619 problems and solved 327, where the base model solved 368 of the same; on the other 450 it solved 198 against 219. MMLU stayed inside its band, at 0.463 against 0.483. The benchmark differences are reported with the standard error of a paired difference, $\sqrt{n_+ + n_-}/n$ for $n_+$ and $n_-$ items gained and lost out of $n$. As an alternative to training the decision, we switched the gate within a reply: the base model, prompted with four examples, writes the prose at $g = 0$, and an adapter writes only the block at $g = 1$; two preliminary dialogue adapters served as block writers (Fig. 5b,c). The switch is exact, but invocation inherits the base model's recall of 0.809, and the sheet prompt lowered the base model's GSM8K accuracy by 34 to 37 points. Reinforcement learning used GRPO with groups of eight samples per task and four tasks per step, a reward that requires a finished reply quoting the executed result and penalises repeated or excess invocations, and a KL penalty toward the supervised checkpoint on every generated token; it ran for 200 steps on one H100 GPU.

## Mechanism

The analyses use the arithmetic adapter. Depth $\ell$ counts layers from the input: $h_0$ is the embedding and $h_\ell$ the output of the $\ell$-th of the 24 layers, so the full-attention layers end at depths 4, 8, …, 24. The logit lens (7) is computed at every depth at the first output position, for the 1,795 addition and multiplication views. The decision depth of a view is the first depth after which the final output remains the top token. Fig. 6b projects the final-layer states, after the final normalisation and with the table's mean subtracted, onto an orthonormal basis of the three leading Fisher discriminant directions of the output classes, with the within-class covariance shrunk 90% toward a multiple of the identity; the nearest-centre shares are measured in this projection. The class-mean intervention (8) acts at the first output position on 80 views per table drawn without replacement with a fixed seed, excluding the halting view of `SUM` and the `MUL` views with a blank multiplicand digit, and the target is drawn uniformly from the table's other outputs; the class means are taken over the same eligible views (241 `SUM` views and the 1,000 `MUL` views with a multiplicand digit). The projection of Fig. 6b uses all 242 `SUM` and 1,100 `MUL` views.

## Baselines

The comparison was pre-registered with its scoring rule. Problems were generated with a fixed seed; every model received the prompt "What is a + b? Give the exact integer. Put the final answer in \boxed{}." (with × for products). The answer was the last boxed integer, and a reply that exhausted the budget or gave no answer was scored as wrong. Qwen3.5-0.8B and Qwen3.5-27B (FP8 release) ran under vLLM with the recommended sampling settings and budgets of 32,768 tokens with chain of thought and 4,096 without. GPT-6.1 Sol (`gpt-6.1-sol`, reasoning effort high), Claude Opus 5.5 (`claude-opus-5-5`, effort high) and Gemini 4 Argon (`gemini-4-argon`, thinking level high) were accessed through their APIs on 31 October 2026 with default sampling and a budget of 32,768 output tokens including reasoning. Five open-weight models, DeepSeek-V4-Pro (`deepseek-v4-pro:0813`), Mistral-Large-4, Gemma 4 31B, GLM-5.3 and gpt-oss-120b, ran through the Ollama cloud API on 31 October 2026 with reasoning enabled at the highest available

setting and the same budget; Kimi K3 was piloted and excluded on cost. Each model was run once per problem, so a difference of a few problems between two models is within sampling variation. Sketchpad ran all 220 problems in batched lockstep in PyTorch, 64,700 views in 1,001 forward passes.

## Related work

**Neural arithmetic.** The Neural GPU learned binary addition and multiplication that generalise beyond the training length[44,45], and specialised arithmetic units followed[46,47]. For transformers, positional encodings and data formats determine length generalisation[3–9], and fine-tuned models reach high accuracy on bounded arithmetic[48,49], with division and gcd remaining difficult[50–52]. Neural cellular automata and differentiable transducers trained from scratch add and multiply far beyond their training lengths[53–55]. Mechanistic analyses find addition circuits correct on almost all inputs[56]. Exactness has been proved under idealised training[57], for small recurrent networks that add inputs of any length after training with a description-length objective[58], and for a purpose-built transformer parameterisation that evaluates Boolean circuits of any depth[59]. Sketchpad differs in using a pretrained vision-language model, in covering division, square roots and gcd, and in certifying the trained weights by enumeration, where these works estimate accuracy on samples or prove exactness for idealised, small or purpose-built networks.

**Procedures in language models.** Neural programmer-interpreters learned grade-school addition as a sequence of steps on a scratch pad read through pointers[60], the architecture whose generalisation Cai et al. later verified[16], and a controller moving over a two-dimensional grid learned multi-digit addition that extends to thousands of digits[61]. CAEF fine-tunes Llama-3.1-8B to execute arithmetic operators step by step on a text tape, with one adapter per operator[62]. Other work embeds non-learned arithmetic modules[63,64], compiles programs into transformer weights[65–68], trains transformers to predict program execution traces[69], or lets a language model emulate an interpreter[70].

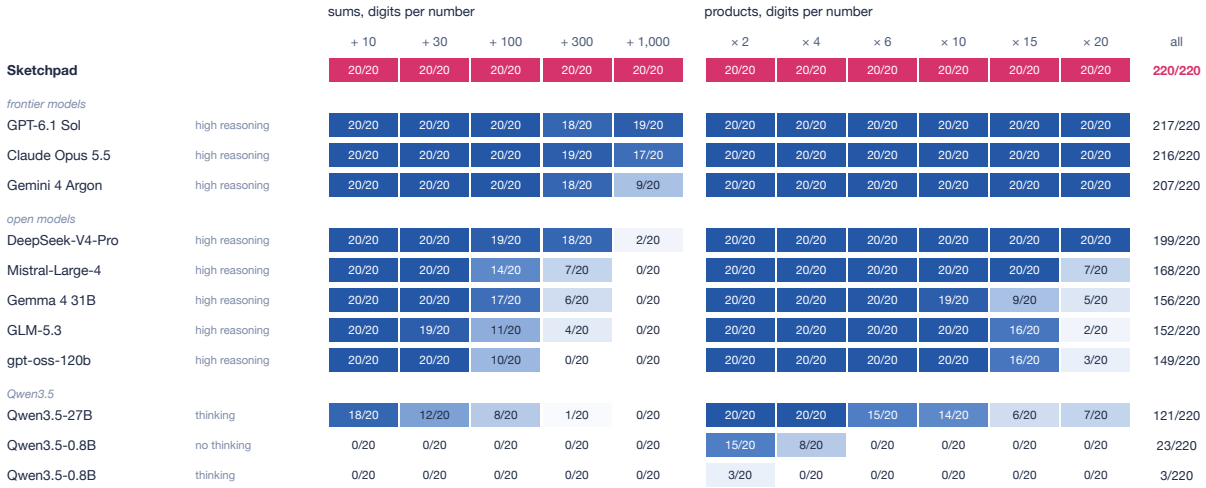
**Tool use.** Toolformer[10], PAL, Program of Thoughts, ToRA, ReTool and ToolRL[11–14,71] and recursive language models[72] delegate computation to external interpreters.

**Verification.** Neural network verification typically bounds outputs over continuous input regions[73]; here the input set is finite, as in computer-assisted proofs of a small transformer's accuracy on a task of fixed input size[74], except that a finite set of views covers inputs of every length. Our theorems specialise results on the identification of finite machines[15,75], the verification of learned recursive programs[16] and certified universal computation by language models[17,76]. Lewandowski et al. likewise restrict each decoding step to a fixed prompt and a local pair of symbols, and check a pretrained model's output on all 1,857 rules of a universal system by exact match at temperature zero; Sketchpad trains a small model to meet the condition and certifies margins on the deployment path, which Theorem 3 makes robust to that path's numerical differences. Discrete neural algorithmic reasoning forces a processor through finite predefined states, so that its learned algorithms can be proved correct on any input[77]; it trains processors from scratch, where Sketchpad certifies a pretrained model. The context discipline resembles the inductive scratchpad[41] and PENCIL[42]. Certification from examples can be hard in general and length-generalisation bounds can be uncomputable[78,79]; the pinned-sheet policy avoids these obstacles by making the input set finite.

**Visual input and scratchpads.** Pixel-based language models[80–82] and visual context compression[83,84] read rendered text. A small network learned addition and subtraction of seven-digit numbers, but not multiplication, from pictures of numbers[85], and vision-language models still fail at multiplication from images[86]. Visual scratchpads support multimodal reasoning[87–91]. Visual Sketchpad[87] shares the name of our system, which refers to Sutherland's Sketchpad[92].

**Algebra.** Transformers have been trained for symbolic integration[93], polynomial simplification[94] and Gröbner bases[95], and large models solve school algebra with chain of thought[96,97]. None of these is exact by construction.

**Continual learning.** Learning without Forgetting[21], progressive networks[98], elastic weight consolidation[30] and adapter routing[99] address forgetting; on-policy distillation and the analysis of Shenfeld et al. relate forgetting to divergence from the base model on its own outputs[31–33]. Our gate is a hard switch in the spirit of progressive networks. Activated LoRA likewise applies an adapter only to tokens after its invocation, reusing the base model's cache for earlier tokens[100], and modality-gated adapters leave inputs of other modalities bit-for-bit unchanged, as their authors verify by exact-equality tests[101]. In Sketchpad the gate is a graph input set by the runtime.




**Extended Data Fig. 1 | Exact answers by length.** Problems answered exactly, of 20 per length, for every system in the comparison: sums of 10 to 1,000 digits (left) and products of 2- to 20-digit numbers (right).

**Extended Data Table 1 | Tables.** Each row is one view an operation can encounter. Margins are the minimum over the table, in logits, on the server (PyTorch, 32-bit) and on the M4 (WebGPU, 16-bit), for the arithmetic adapter. `SUM` has one output position ($K_v = 1$), the other operations three; – marks an output that is always the same placeholder. The five multiplication tables, `MUL` to `NXT`, total 1,553 views. Next-operation codes: `A` add the partial product, `Z` end of pass, `M` multiply (or, after `EST`, proceed with the guess), `F` flush, `N` next digit (after `EST`, the guess is 0), `H` halt, `B` final borrow, `S` continue subtraction, `C` correct the remainder.

| Operation | View (top to bottom) | Outputs at `=`, `?`, `?` | Rows | Server | M4 |
|---|---|---|---|---|---|
| `SUM` | carry (blank, 1), top digit (blank, 0–9), bottom digit (blank, 0–9) | sum and carry as one token (0–19), or `H` | 242 | 3.96 | 3.02 |
| `MUL` | multiplicand digit (blank, 0–9), multiplier digit (0–9), carry (blank, 0–8) | digit, carry, `A` or `Z` | 1,100 | 6.39 | 6.52 |
| `ADD` | carry (blank, 1), product digit (0–9), running-total digit (blank, 0–9) | digit, carry, `M` | 220 | 9.37 | 9.50 |
| `ADZ` | as `ADD`, at the end of a pass | digit, carry, `F` | 220 | 8.41 | 8.00 |
| `FLS` | carry (blank, 1), blank, blank | final carry, –, `N` | 2 | 12.93 | 13.11 |
| `NXT` | next multiplier digit (blank, 0–9), blank, blank | –, –, `M` or `H` | 11 | 10.48 | 10.22 |
| `SUB` | borrow (blank, 1), top digit (blank, 0–9), bottom digit (blank, 0–9) | digit, borrow, `S`; top and bottom blank: –, –, `H` (no borrow) or `B` (borrow) | 242 | 8.36 | 8.19 |
| `EST` | leading remainder digits $r_1$ (0–$d$) and $r_2$ (0–9), leading divisor digit $d$ (5–9) | $\hat{q} = \min(9, \lfloor (10r_1 + r_2)/d \rfloor)$, –, `M` if $\hat{q} > 0$, else `N` | 400 | 4.51 | 4.61 |
| `DEC` | estimate $q$ (1–9), blank, blank | $q - 1$, –, `C` | 9 | 10.00 | 10.09 |
| **All** | | | **2,446** | **3.96** | **3.02** |

## Data and code availability

The model, adapter, certificate and analysis data are available on Hugging Face (`mohsenvand/sketchpad-0.8b-preview`). The runtime, the algebra, the baselines and the sources of this paper are available under the Apache 2.0 licence at https://github.com/NeoVand/sketchpad. The Lean proofs are in `paper/lean/` of the same repository (Appendix A).

**Extended Data Table 2 | Browser test.** 50 problems per set on the Apple M4 (WebGPU, 16-bit decoder), one problem at a time. Passes are fewer than views because independent views of one problem, such as the `MUL` and `SUB` views of a multiply-subtract pass, share a forward pass.

| Set | Exact | Median s | Max s | Over 12 s | Median passes / views | Cost model median s |
|---|---|---|---|---|---|---|
| $a \div b$ to 10 places ($a \le 999$, $b \le 99$) | 50 | 7.0 | 13.3 | 1 | 65 / 90 | 7.6 |
| $\sqrt{a}$ to 6 places ($a \le 999$) | 50 | 9.8 | 16.5 | 11 | 86 / 137 | 10.7 |
| $\gcd(a, b)$ ($a, b \le 999$) | 50 | 7.0 | 13.4 | 4 | 61 / 84 | 7.1 |
| signed $\pm$, up to 20 digits | 50 | 2.0 | 4.7 | 0 | 19 / 19 | 1.9 |

**Extended Data Table 3 | Cost of the algebra benchmark.** Views, forward passes after batching and seconds under the cost model, default check, for the textbook and evaluation routes. The first eleven problems are those of Fig. 4c. The last three have irrational roots, which the textbook route returns as radicals and the evaluation route only as decimal bounds, so they have no comparable evaluation entry and were not measured on the M4.

| Problem | Textbook views / passes / s | Evaluation views / passes / s |
|---|---|---|
| `solve 3(x-2)+5=2x+14` | 36 / 22 / 2.8 | 75 / 27 / 4.5 |
| `solve 7x+13=2026-4x` | 97 / 62 / 7.6 | 165 / 88 / 11.8 |
| `solve 2x+3y=7, x-y=1` | 75 / 48 / 5.9 | 194 / 41 / 9.8 |
| `solve (2x-3)(x+5)=x^2+1-16` | 79 / 43 / 5.7 | 177 / 50 / 9.8 |
| `solve x^2-x-6=0` | 102 / 77 / 8.8 | 162 / 79 / 11.1 |
| `123456 * 654321` | 96 / 46 / 6.5 | 96 / 46 / 6.5 |
| `expand (x+2)^5` | 159 / 95 / 12.1 | 458 / 248 / 33.1 |
| `expand (2x-3)^4` | 172 / 94 / 12.5 | 492 / 274 / 36.0 |
| `factor x^3-6x^2+11x-6` | 195 / 149 / 16.9 | 483 / 210 / 31.5 |
| `expand (x+1)^10` | 322 / 154 / 21.9 | 2,359 / 845 / 142.2 |
| `solve 2x+3y-z=1, x-y+2z=3, 3x+y+z=8` | 323 / 167 / 22.8 | 538 / 102 / 26.5 |
| `solve x^2-2=0` | 86 / 65 / 7.4 | |
| `solve 2x^2+3x-1=0` | 144 / 88 / 11.0 | |
| `solve x^2-6x+4=0` | 133 / 96 / 11.2 | |

# Extended Data

# A Proofs

The theorems are proved in the Lean 4 proof assistant [102] with Mathlib [103] (version 4.35.0), without `sorry` or additional axioms; `#print axioms` reports only `propext`, `Classical.choice` and `Quot.sound`, or a subset. The formalisation models the network as a deterministic function of its context, or as a causal decoder whose layer at a position reads only earlier positions, with arbitrary tie-breaking in the argmax and exact arithmetic; floating point enters only through $\tau$ in Theorem 3. The proofs below follow the Lean proofs, whose declarations are listed in Extended Data Table 6 (in the namespace `Polyformer.Pinned` unless prefixed). Lemmas 1–4 and Theorem 4a are formalised in general, and Theorems 1, 4b and 3 for the sheet machine (`Sheet.lean`) and for a token form, in which the state is a token and each step reads a state token and an event token after a fixed prompt (`Pinned.lean`, `Compose.lean`). In the sheet form of Theorem 4b, each call runs a sheet procedure for a given number of steps from a configuration built from its input and reads its result from the final configuration. The hypotheses $R_T(I) \subseteq V$ and the correctness of the reference procedures are assumptions of these theorems, not formalised for the concrete layouts. `Paper.lean` names every statement by its number in this paper.

## Definitions and lemmas

**Definition 1** (Sheet machine). *A sheet machine consists of sets $C$ of configurations, $\mathcal{V}$ of views and $A$ of outputs, a view map* $\mathrm{view}\colon C \to \mathcal{V} \cup \{\bot\}$ *and a paint map* $\mathrm{paint}\colon C \times A \to C$. *For $g\colon \mathcal{V} \to A$, the step*

**Extended Data Table 4 | Dialogue fine-tuning.** Held-out templates with the gate on and no examples, against the base model prompted with four examples (invocation and translation) or with none (benchmarks). IFEval, GSM8K and MMLU in points relative to the base model (0.536, 0.516 and 0.476), on the same 541, 250 and 500 items for every recipe; for the dialogue adapter, with the standard error of the paired difference. The last row rescores the dialogue adapter on GSM8K and MMLU items not used before (Methods). GSM8K replies at $g = 1$ invoke the sheet, executed by the reference implementation (Methods); –, not measured.

| Recipe | Precision / recall | Word | Equations | Prose | IFEval | GSM8K | MMLU |
|---|---|---|---|---|---|---|---|
| criterion | $\geq 0.95$ / $\geq 0.95$ | $\geq 0.90$ | $\geq 0.95$ | $\geq 0.95$ | $\geq -2$ | $\geq -2$ | $\geq -2$ |
| base model, four examples | 0.971 / 0.809 | 0.789 | 0.583 | 0.667 | 0 | 0 | 0 |
| synthetic dialogues, self-distillation (from a preliminary multiplication adapter) | 0.957 / 0.996 | 0.872 | 0.967 | 0.983 | −4.8 | – | −6.6 |
| synthetic dialogues, self-distillation | 1.000 / 0.998 | 0.894 | 0.975 | 1.000 | −0.55 | −31.2 | −5.4 |
| + human multi-step solutions | 1.000 / 0.994 | 0.856 | 0.975 | 0.992 | −2.8 | −14.0 | −2.4 |
| + base-model multi-step solutions | 1.000 / 0.994 | 0.817 | 0.958 | 0.992 | −4.4 | −6.4 | −3.4 |
| + on-policy distillation | 1.000 / 0.946 | 0.678 | 0.958 | 0.983 | −1.5 | −2.0 | −2.2 |
| + wider word-problem data | 1.000 / 0.991 | 0.856 | 0.958 | 0.975 | +0.2 | −2.4 | −2.2 |
| + GRPO | 1.000 / 0.981 | 0.861 | 0.958 | 0.992 | −2.0 | −4.8 | −0.6 |
| + GRPO, strict reward and task mix | 1.000 / 0.794 | 0.378 | 0.933 | 0.975 | −2.2 | −0.8 | −1.0 |
| **+ GRPO anchored to the supervised checkpoint (dialogue adapter)** | **0.994 / 0.989** | **0.928** | **0.967** | **0.983** | **−1.5 ± 2.0** | **−1.6 ± 3.4** | **−1.4 ± 1.6** |
| dialogue adapter, fresh items (1,069 GSM8K, 1,000 MMLU) | – | – | – | – | – | **−5.8 ± 1.7** | **−2.0 ± 1.1** |

$\sigma_g \colon C \to C$ *is*

$$\sigma_g(c) = \begin{cases} \mathrm{paint}\big(c, g(v)\big) & \text{if } \mathrm{view}(c) = v \in \mathcal{V}, \\ c & \text{if } \mathrm{view}(c) = \bot, \end{cases} \tag{12}$$

*and the run of length $n$ from $c_0$ is $(c_0, \sigma_g(c_0), \ldots, \sigma_g^n(c_0))$. The model's run and the reference run are those of $\sigma_f$ and $\sigma_T$. For $I \subseteq C$,*

$$\mathrm{Reach}_T(I) = \{\sigma_T^n(c_0) : c_0 \in I,\ n \geq 0\}, \qquad R_T(I) = \{\mathrm{view}(c) : c \in \mathrm{Reach}_T(I)\} \setminus \{\bot\}.$$

**Definition 2** (Runs over events). *Let $S$ be a set of states, $E$ a set of events and $F \colon S \times E \to S$. For $s_0 \in S$ and a word $w$ over $E$, $F^w(s_0)$ is the left fold, $F^\varepsilon(s_0) = s_0$ and $F^{ue}(s_0) = F(F^u(s_0), e)$, and the trace of $F$ on $w = e_1 \cdots e_n$ is $\big(F^{e_1 \cdots e_i}(s_0)\big)_{i=0}^{n}$. For $B \subseteq E$, $\mathrm{Reach}_F^B(s_0) = \{F^u(s_0) : u \in B^*\}$.*

**Extended Data Table 5 | Comparison.** Problems answered exactly, of 20 per cell; in brackets, the number of replies that exhausted the token budget. Qwen3.5-27B is the FP8 release; all other models ran with reasoning at their highest available setting, accessed on 31 October 2026. The errors of GPT-6.1 Sol and Claude Opus 5.5 were completed replies well within the budget, differing from the correct sum in 1 to 637 digits; 9 of the 13 errors of Gemini 4 Argon exhausted the budget. Most errors of the open-weight models exhausted the budget, except those of Gemma 4 31B, 46 of whose 64 errors were incorrect answers in completed replies. Qwen3.5-27B exhausted the budget in 93 of its 99 failures, and Qwen3.5-0.8B with chain of thought in all of its 217.

| System | +10 | +30 | +100 | +300 | +1,000 | ×2 | ×4 | ×6 | ×10 | ×15 | ×20 | All |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sketchpad | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 220 |
| GPT-6.1 Sol | 20 (0) | 20 (0) | 20 (0) | 18 (0) | 19 (0) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 217 |
| Claude Opus 5.5 | 20 (0) | 20 (0) | 20 (0) | 19 (0) | 17 (0) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 216 |
| Gemini 4 Argon | 20 (0) | 20 (0) | 20 (0) | 18 (1) | 9 (8) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 207 |
| DeepSeek-V4-Pro | 20 (0) | 20 (0) | 19 (0) | 18 (0) | 2 (17) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 199 |
| Mistral-Large-4 | 20 (0) | 20 (0) | 14 (5) | 7 (10) | 0 (20) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 7 (13) | 168 |
| Gemma 4 31B | 20 (0) | 20 (0) | 17 (0) | 6 (2) | 0 (15) | 20 (0) | 20 (0) | 20 (0) | 19 (1) | 9 (0) | 5 (0) | 156 |
| GLM-5.3 | 20 (0) | 19 (1) | 11 (6) | 4 (15) | 0 (20) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 16 (3) | 2 (18) | 152 |
| gpt-oss-120b | 20 (0) | 20 (0) | 10 (9) | 0 (18) | 0 (20) | 20 (0) | 20 (0) | 20 (0) | 20 (0) | 16 (4) | 3 (17) | 149 |
| Qwen3.5-27B, chain of thought | 18 (2) | 12 (8) | 8 (10) | 1 (19) | 0 (20) | 20 (0) | 20 (0) | 15 (4) | 14 (6) | 6 (11) | 7 (13) | 121 |
| Qwen3.5-0.8B, chain of thought | 0 (20) | 0 (20) | 0 (20) | 0 (20) | 0 (20) | 3 (17) | 0 (20) | 0 (20) | 0 (20) | 0 (20) | 0 (20) | 3 |
| Qwen3.5-0.8B | 0 (8) | 0 (20) | 0 (20) | 0 (20) | 0 (20) | 15 (4) | 8 (9) | 0 (14) | 0 (20) | 0 (19) | 0 (20) | 23 |

**Extended Data Table 6 | Lean statements.** The package is in `paper/lean/` of the Sketchpad repository (Lean 4.35.0, Mathlib); the files other than `Sheet.lean` and `Paper.lean` are in its `Polyformer/` folder. `Paper.lean` provides aliases by paper number, such as `theorem2_counterexample` for `theorem8a_counterexample`.

| Statement | Lean declarations | File |
|---|---|---|
| Theorem 1 | `Sketchpad.sheet_exact_iff`, `Sketchpad.sheet_certified`; token form `regeneration_trace_iff`, `pinned_regeneration_iff` | `Sheet.lean`, `Pinned.lean` |
| Theorem 2 | `no_finite_test_suite`, `theorem8a_counterexample`, `no_finite_check` | `Unpinned.lean` |
| Theorem 3 | `Sketchpad.sheet_exact_of_close`; token form `regeneration_of_close`; unpinned decoding `unpinned_exact_of_close` | `Sheet.lean`, `Pinned.lean`, `Unpinned.lean` |
| Theorem 4 | (a) `error_decomposition`; (b) `Sketchpad.sheet_compose_exact`, `Sketchpad.sheet_error_bound`; token form `compose_exact`, `error_bound` | `Compose.lean`, `Sheet.lean` |
| Lemma 1 | `sel_eq_of_close`, `sel_eq_of_strict` | `Pinned.lean` |
| Lemma 2 | `Polyformer.Interp.regeneration`, `scanl_eq_iff_foldl_eq`, `regeneration_trace_iff` | `Interp.lean`, `Pinned.lean` |
| Lemma 3 | `histRun_eq_of_closed`, `reach_closed` | `Pinned.lean` |
| Lemma 4 | `resid_congr`, `cache_sound`, `cachedForward_eq`, `policy_tick_eq` | `Pinned.lean`, `Policy.lean` |

**Lemma 1** (Margins decide the argmax). *Let $O$ be a finite set, $z, \tilde{z} \in \mathbb{R}^O$, $y \in O$ and $\tau \in \mathbb{R}$. If $z_a + 2\tau < z_y$ for every $a \in O \setminus \{y\}$ and $|\tilde{z}_a - z_a| \le \tau$ for every $a \in O$, then every maximiser of $\tilde{z}$ over $O$ is $y$. In particular, if $m(v) > 0$, then $f(v) = T(v)$ under every tie rule.*

*Proof.* Let $s$ maximise $\tilde{z}$ over $O$ and suppose $s \neq y$. Then $\tilde{z}_s \le z_s + \tau < z_y - \tau \le \tilde{z}_y$, which contradicts $\tilde{z}_y \le \tilde{z}_s$. For the second statement take, at each output position $k$, $O = \Sigma$, $y = T_k(v)$, $\tilde{z} = z = z_k(v)$ and $\tau = 0$; the hypothesis is that the $k$-th term in (2) is positive.

**Lemma 2** (Regeneration). *Let $F, \delta \colon S \times E \to S$, $s_0 \in S$ and $B \subseteq E$. Then $F^w(s_0) = \delta^w(s_0)$ for every $w \in B^*$ if and only if $F(s,e) = \delta(s,e)$ for every $s \in \mathrm{Reach}_\delta^B(s_0)$ and $e \in B$. Both conditions are equivalent to equality of the traces of $F$ and $\delta$ on every $w \in B^*$.*

*Proof.* ($\Leftarrow$) Induction on the length of $w$. For $w = \varepsilon$ both sides are $s_0$. For $w = ue$ with $u \in B^*$ and $e \in B$,

$$F^{ue}(s_0) = F\big(F^u(s_0), e\big) = F\big(\delta^u(s_0), e\big) = \delta\big(\delta^u(s_0), e\big) = \delta^{ue}(s_0),$$

by the induction hypothesis and then by the assumption at $s = \delta^u(s_0) \in \mathrm{Reach}_\delta^B(s_0)$.

($\Rightarrow$) Let $s = \delta^u(s_0)$ with $u \in B^*$, and let $e \in B$. Applying the assumption to $u$ and to $ue$,

$$F(s,e) = F\big(F^u(s_0), e\big) = F^{ue}(s_0) = \delta^{ue}(s_0) = \delta(s,e).$$

Traces: the $i$-th entry of the trace on $w$ is the fold over the prefix $e_1 \cdots e_i \in B^*$, so equal folds on all words give equal traces; conversely, the last entries of equal traces are the folds over $w$.

**Lemma 3** (Steps that read the history). *Let $G\colon E^* \times S \times E \to S$, and for $w = e_1 \cdots e_n$ let $s'_0 = s_0$ and $s'_i = G(e_1 \cdots e_{i-1}, s'_{i-1}, e_i)$. If $G(u,s,e) = \delta(s,e)$ for all $u \in B^*$, $s \in \mathrm{Reach}_\delta^B(s_0)$ and $e \in B$, then $s'_n = \delta^w(s_0)$ for every $w \in B^*$.*

*Proof.* By induction on $i$, $s'_i = \delta^{e_1 \cdots e_i}(s_0)$. This holds for $i = 0$. If it holds for $i-1$, then $s'_{i-1} \in \mathrm{Reach}_\delta^B(s_0)$, and the assumption with $u = e_1 \cdots e_{i-1}$ gives $s'_i = \delta(s'_{i-1}, e_i) = \delta^{e_1 \cdots e_i}(s_0)$. Lean proves the lemma with any set $Q \ni s_0$ closed under $\delta$ on $B$ in place of $\mathrm{Reach}_\delta^B(s_0)$, which is the smallest such set.

**Definition 3** (Causal decoder). *A causal decoder of depth $L$ maps tokens $x_0 x_1 \cdots$ to states $h_p^0 = \mathrm{emb}(x_p, p)$ and $h_p^{\ell+1} = \Lambda_\ell(p, h^\ell)$, where $h^\ell = (h_0^\ell, h_1^\ell, \dots)$ is the sequence of layer-$\ell$ states and $\Lambda_\ell(p,h)$ depends only on $h_0, \dots, h_p$; the logits of a sequence of length $N$ are $\mathrm{head}(h_{N-1}^L)$. A cached forward of a block $b$ after a prefix $\Pi$ reads $h_p^\ell$ for $p < |\Pi|$ from a cache computed on $\Pi$ alone and computes the positions $p \geq |\Pi|$ by the same recursion.*

**Lemma 4** (Causality and caching). *(i) $h_p^\ell$ depends only on $x_0, \dots, x_p$. (ii) The cached forward of $b$ after $\Pi$ computes $h_p^\ell(\Pi b)$ at every layer and position, and hence the logits of $\Pi b$. (iii) Decoding a two-token block $b_1 b_2$ one token at a time on the cache of $\Pi$ gives the logits of $\Pi b_1 b_2$.*

*Proof.* (i) Induction on $\ell$: $h_p^0$ reads only $x_p$; if $x$ and $x'$ agree up to $p$, the induction hypothesis gives $h_i^\ell(x) = h_i^\ell(x')$ for $i \leq p$, so $h_p^{\ell+1}(x) = h_p^{\ell+1}(x')$ by causality of $\Lambda_\ell$. (ii) Induction on $\ell$. For $p < |\Pi|$ the cache holds $h_p^\ell(\Pi)$, which equals $h_p^\ell(\Pi b)$ by (i). For $p \geq |\Pi|$, layer 0 embeds $b_{p-|\Pi|} = (\Pi b)_p$, and layer $\ell+1$ applies $\Lambda_\ell(p,\cdot)$ to states equal to $h^\ell(\Pi b)$ by the induction hypothesis. (iii) Decoding $b_1$ on the cache of $\Pi$ leaves the cache of $\Pi b_1$ by (ii), and decoding $b_2$ on that cache gives the logits of $\Pi b_1 b_2$, again by (ii).

## Theorem 1

*Proof of Theorem 1.* Fix $c_0 \in I$ and apply Lemma 2 with $S = C$, a single event $E = B = \{*\}$, $F(c,*) = \sigma_f(c)$ and $\delta(c,*) = \sigma_T(c)$. The words over $B$ are $*^n$, $F^{*^n}(c_0) = \sigma_f^n(c_0)$, the trace on $*^n$ is the run of length $n$, and $\mathrm{Reach}_\delta^B(c_0) = \{\sigma_T^n(c_0) : n \geq 0\} = \mathrm{Reach}_T(\{c_0\})$ (Definition 1). By the lemma, the model's run from $c_0$ equals the reference run at every length if and only if

$$\sigma_f(c) = \sigma_T(c) \quad \text{for every } c \in \mathrm{Reach}_T(\{c_0\}). \tag{13}$$

*Sufficiency.* Let $f = T$ on $R_T(I)$ and $c \in \mathrm{Reach}_T(\{c_0\})$. If $\mathrm{view}(c) = \bot$, then $\sigma_f(c) = c = \sigma_T(c)$. Otherwise $v = \mathrm{view}(c) \in R_T(I)$ and $\sigma_f(c) = \mathrm{paint}(c, f(v)) = \mathrm{paint}(c, T(v)) = \sigma_T(c)$. So (13) holds for every $c_0 \in I$.

*Necessity.* Let $v \in R_T(I)$, so that $v = \mathrm{view}(c)$ for some $c \in \mathrm{Reach}_T(\{c_0\})$ and $c_0 \in I$; then $\mathrm{view}(c) \neq \bot$, because $R_T(I)$ excludes $\bot$. By (13), $\mathrm{paint}(c, f(v)) = \mathrm{paint}(c, T(v))$, and injectivity of $\mathrm{paint}(c,\cdot)$ gives $f(v) = T(v)$.

*The certificate.* If $R_T(I) \subseteq V$ and $m(v) > 0$ for all $v \in V$, Lemma 1 gives $f = T$ on $V \supseteq R_T(I)$, and sufficiency, which did not use injectivity, applies.

## Theorem 2

Let $\Sigma$ be a set of tokens, $\Pi$ a fixed prompt and $\rho\colon \Sigma^* \to \Sigma$ a step map that returns a maximiser of the logits at the last position of a context. On events $e_1 e_2 \cdots$ from a state $s_0$, pinned decoding computes $s_i = \rho(\Pi\, s_{i-1}\, e_i)$; unpinned decoding keeps the whole context, $x_0 = \Pi\, s_0$, $s_i = \rho(x_{i-1}\, e_i)$ and $x_i = x_{i-1}\, e_i\, s_i$. A row $(s,e)$ of a table $\delta$ passes if the logits at $\Pi\, s\, e$ have their strict maximum at $\delta(s,e)$.

**Definition 4** (The counterexample). *The tokens are $\Sigma = \{\omega, \mathsf{u}, \mathsf{v}, \mathsf{a}\}$, where $\omega$ is a sink token, and the prompt is $\Pi = \omega\,\mathsf{u}\,\mathsf{a}\,\mathsf{u}\,\mathsf{v}\,\mathsf{a}\,\mathsf{v}$. The table has states $\{\mathsf{u}, \mathsf{v}\}$, the event $\mathsf{a}$, $\delta(\mathsf{u},\mathsf{a}) = \mathsf{u}$, $\delta(\mathsf{v},\mathsf{a}) = \mathsf{v}$ and start $\mathsf{u}$, so its state is $\mathsf{u}$ after every input. For $R > 0$ and $c \in \mathbb{R}$, the logits at the last position of a context $x = x_0 x_1 \cdots x_{N-1}$ are*

$$\ell(x)_y = R\,[\,y = x_{N-2}\,] + [\,y = \mathsf{v}\,]\,\eta(x), \qquad \eta(x) = \frac{\sum_{j=0}^{N-1} e^{\kappa_j}\,\beta_j}{\sum_{j=0}^{N-1} e^{\kappa_j}}, \tag{14}$$

*where $[P]$ is 1 if $P$ holds and 0 otherwise, with scores $\kappa_0 = c$ and $\kappa_j = 0$ for $j \geq 1$, and values $\beta_j = 4R$ if $x_j = \mathsf{a}$ and $\beta_j = 0$ otherwise. The first term is a lookup that returns the state token of the last row, which is the correct output for $\delta$; the second is one attention head at the last position, with score $c$ on the sink, which writes its value to the logit of $\mathsf{v}$. If $x_0 = \omega$ and $k$ of the $n = N - 1$ tokens after it are $\mathsf{a}$,*

$$\eta(x) = \frac{4Rk}{e^c + n}. \tag{15}$$

**Theorem 2** (Necessity of the pinned-sheet policy, precise form). *Let $\mathcal{T}$ be a finite set of contexts, $M$ the largest length of a context in $\mathcal{T}$ ($M = 0$ if $\mathcal{T}$ is empty), and $e^c = 3M + 5$. For every $R > 0$ and every tie rule, the decoder (14): (a) passes both rows of $\delta$ with positive margins; (b) at every context in $\mathcal{T}$ of the form $\Pi\, w\, s\, \mathsf{a}$ with $s \in \{\mathsf{u}, \mathsf{v}\}$, has its strict maximum at $\delta(s, \mathsf{a})$; (c) under unpinned decoding of $\mathsf{a}^{2M+4}$ ends in the state $\mathsf{v}$, whereas $\delta$ ends in $\mathsf{u}$; and (d) under pinned decoding equals $\delta$ on every input.*

*Proof. Two facts.* At a context ending in $\mathsf{v}\,\mathsf{a}$, $\ell_{\mathsf{v}} = R + \eta \geq R > 0$ while every other logit is 0, since $\eta \geq 0$; so $\mathsf{v}$ is the strict maximiser. At a context ending in $\mathsf{u}\,\mathsf{a}$, $\ell_{\mathsf{u}} = R$, $\ell_{\mathsf{v}} = \eta$ and $\ell_\omega = \ell_{\mathsf{a}} = 0$; so $\mathsf{u}$ is the strict maximiser if $\eta < R$, and $\mathsf{v}$ is if $\eta > R$.

(a) A row $\Pi\, s\, \mathsf{a}$ has $n = 8$ tokens after the sink, three of them $\mathsf{a}$, so $\eta = 12R/(e^c + 8)$ by (15), which is less than $R$ because $e^c = 3M + 5 > 4$. Both facts then give the rows.

(b) A test context $x = \Pi\, w\, \mathsf{u}\, \mathsf{a}$ of length $N \leq M$ has $n = N - 1$ tokens after the sink and at most $N - 1$ of them $\mathsf{a}$, so

$$\eta(x) \leq \frac{4R(N-1)}{3M + 4 + N},$$

and the right-hand side is less than $R$ if and only if $3N < 3M + 8$, which holds because $N \leq M$. Test contexts ending in $\mathsf{v}\,\mathsf{a}$ are covered by the first fact.

(c) A state $\mathsf{v}$ is followed only by $\mathsf{v}$, by the first fact, so if $s_{i-1} = \mathsf{u}$, then $s_0 = \cdots = s_{i-1} = \mathsf{u}$ and the context that determines $s_i$ is $\Pi\, \mathsf{u}\, (\mathsf{a}\,\mathsf{u})^{i-1}\, \mathsf{a}$, with $2i + 6$ tokens after the sink, $i + 2$ of them $\mathsf{a}$. By (15),

$$\eta = \frac{4R(i+2)}{e^c + 2i + 6} > R \iff 2i > e^c - 2,$$

and $\eta < R$ if $2i < e^c - 2$. Induction on $i$ gives $s_i = \mathsf{u}$ while $2i < e^c - 2$. If $M$ is odd, the step $i = (3M + 3)/2$ has $2i = e^c - 2$, so $\eta = R$ and $\ell_{\mathsf{u}} = \ell_{\mathsf{v}}$; the tie rule may return either state. In every case, $s_i = \mathsf{v}$ for the first $i$ with $2i > e^c - 2$, if not earlier, and $\mathsf{v}$ persists by the first fact. For $i = 2M + 4$, $2i = 4M + 8 > 3M + 3 = e^c - 2$, so $s_{2M+4} = \mathsf{v}$.

(d) Under pinned decoding every step reads a row $\Pi\, s\, \mathsf{a}$ with $s \in \{\mathsf{u}, \mathsf{v}\}$, which returns $\delta(s, \mathsf{a}) \in \{\mathsf{u}, \mathsf{v}\}$ by (a). Lemma 2 with $S = \{\mathsf{u}, \mathsf{v}\}$, $B = \{\mathsf{a}\}$ and $F(s, \mathsf{a}) = \rho(\Pi\, s\, \mathsf{a})$ gives equality with $\delta$ on every input.

With $e^c = 2N + 5$ instead, the same computation gives a decoder that passes both rows and is exact under unpinned decoding on every input of length at most $N$, yet wrong on $\mathsf{a}^{N+2}$ (`no_finite_check`); testing every input up to a given length does not suffice either. Conversely, if the context moves every logit by at most $\tau$ from the logit at the pinned row and every certified margin exceeds $2\tau$, unpinned decoding is exact on every input (`unpinned_exact_of_close`). In the construction, the head's write is such a perturbation, and it grows without bound.

## Theorem 3

*Proof of Theorem 3.* The first statement is Lemma 1. For the second, fix $c_0 \in I$ and let the execution path's step $t$ be $G_t(c) = \mathrm{paint}(c, \tilde{f}_t(\mathrm{view}(c)))$ if $\mathrm{view}(c) \neq \bot$ and $G_t(c) = c$ otherwise, where $\tilde{f}_t(v)$ is the argmax of the execution-path logits $\tilde{z}_k^{(t)}(v)$ at each position; these may depend on $t$, and through it on the rest of the batch. Let $c \in \mathrm{Reach}_T(\{c_0\})$ with $v = \mathrm{view}(c) \neq \bot$. Then $v \in R_T(I) \subseteq V$, so $\|\tilde{z}_k^{(t)}(v) - z_k(v)\|_\infty \leq \tau$ and $m(v) > 2\tau$, and Lemma 1 at each position gives $\tilde{f}_t(v) = T(v)$, hence $G_t(c) = \sigma_T(c)$; this also holds when $\mathrm{view}(c) = \bot$. Lemma 3 with a single event, so that the history is the step count, gives $G_t \circ \cdots \circ G_1(c_0) = \sigma_T^t(c_0)$ for every $t$.

In Lean, the first statement is `sel_eq_of_close` and Lemma 3 is `histRun_eq_of_closed`; their combination is `Sketchpad.sheet_exact_of_close` for the sheet machine, with the margin condition stated per output position, `regeneration_of_close` for the token form and `unpinned_exact_of_close` for unpinned decoding. By Lemma 4, a pinned step computed on a cache of a fixed prefix, or decoded token by token on it, yields exactly the logits of the full prompt (`cachedForward_eq`, `policy_tick_eq`), so in exact arithmetic it is an execution path with $\tau = 0$.

### Theorem 4

**Definition 5** (Transcript). *A transcript makes $K$ calls. Call $i$ receives the input $x_i = q_i(r_{<i})$ and returns $r_i = E_i(x_i)$, where $E_i$ is the procedure that executes it and $r_{<i} = (r_1, \dots, r_{i-1})$; the answer $g(r)$ reads only $r = (r_1, \dots, r_K)$. Call $i$ has a table $T_i$, the function its procedure is certified to compute. The intended computation has inputs $q_i^*$, functions $T_i^*$ and combining program $g^*$, with results $r_i^* = T_i^*(q_i^*(r_{<i}^*))$ and answer $g^*(r^*)$.*

**Theorem 4** (Composition, precise form). *(a) If $g(r) \neq g^*(r^*)$, then for some $i \leq K$, $E_i(x_i) \neq T_i(x_i)$, or $T_i(q_i^*(r_{<i}^*)) \neq T_i^*(q_i^*(r_{<i}^*))$, or $q_i(r_{<i}^*) \neq q_i^*(r_{<i}^*)$; or else $g(r^*) \neq g^*(r^*)$. (b) Let every quantity depend on a transcript $\omega$ in a set $\Omega$ with a measure $\mu$. If $E_i(x_i) = T_i(x_i)$ for every call of every transcript, then*

$$\begin{aligned}\mu\{g(r) \neq g^*(r^*)\} \leq\ & \mu\{\exists i\colon T_i(q_i^*(r_{<i}^*)) \neq T_i^*(q_i^*(r_{<i}^*))\} \\ & + \mu\{\exists i\colon q_i(r_{<i}^*) \neq q_i^*(r_{<i}^*)\} + \mu\{g(r^*) \neq g^*(r^*)\}.\end{aligned}$$

*Proof.* (a) Suppose that none of the four events occurs. We show $r_i = r_i^*$ for all $i \leq K$ by strong induction on $i$. If $r_j = r_j^*$ for all $j < i$, then $r_{<i} = r_{<i}^*$, and

$$r_i = E_i(q_i(r_{<i})) = T_i(q_i(r_{<i})) = T_i(q_i(r_{<i}^*)) = T_i(q_i^*(r_{<i}^*)) = T_i^*(q_i^*(r_{<i}^*)) = r_i^*,$$

using in turn that the procedure, the transcription and the table are correct. Then $g(r) = g(r^*) = g^*(r^*)$, because $g$ reads only the results and the combining program is correct, a contradiction.

(b) Under the hypothesis, (a) places the event $\{g(r) \neq g^*(r^*)\}$ inside the union of the other three, and the bound follows from monotonicity and subadditivity of $\mu$, which hold for arbitrary sets, so no measurability is needed; a probability distribution gives (5). For a procedure certified on the views it reaches, the hypothesis $E_i(x_i) = T_i(x_i)$ is Theorem 1 applied to its run on $x_i$, whatever the length of $x_i$ and the order of the calls. In Lean, (a) is proved for arbitrary $E_i$, $T_i$ and $T_i^*$, and (b) for calls executed by sheet procedures (`Sketchpad.sheet_error_bound`) and by the token-form procedure (`error_bound`), with the hypothesis derived from Theorem 1 in each form (`Sketchpad.sheet_compose_exact`, `compose_exact`).

# B Certificate format

A certificate is a JSON file distributed with the model graph. It specifies the prompt forms under which it was measured, the numerical precision of each graph, the library version, hashes of the glyph set and of every model file, the device and the time, and, for each table, the number of rows, the number passed and every row's margin in a fixed order, to $10^{-3}$ logits. A client accepts a certificate only if every hash matches the loaded files, and recomputes the lowest-margin tenth of each table before use.

# C Reproducibility

Every number in this paper is listed, with its source record, in `paper/numbers.md` of the Sketchpad repository. Training, server certification, the gate identity test and dialogue fine-tuning are scripted for Modal; the device certificate and browser test run in the client; the algebra tests and cost tables run with `pnpm vitest run src/lib/sketchpad/program`; the comparison is in `paper/baselines/`; the Lean proofs build with `lake build` in `paper/lean/`, and `lake env lean Paper.lean`, run in the same folder, prints the axioms of every statement.

# References


1. Wei, J. et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In Advances in Neural Information Processing Systems, 24824–24837 (2022). `https://arxiv.org/abs/2201.11903`
2. Nye, M. et al. Show Your Work: Scratchpads for Intermediate Computation with Language Models. Preprint at arXiv:2112.00114 (2021). `https://arxiv.org/abs/2112.00114`
3. Nogueira, R., Jiang, Z. & Lin, J. Investigating the Limitations of Transformers with Simple Arithmetic Tasks. Preprint at arXiv:2102.13019 (2021). `https://arxiv.org/abs/2102.13019`

4. Anil, C. et al. Exploring Length Generalization in Large Language Models. In Advances in Neural Information Processing Systems, 38546–38556 (2022). https://arxiv.org/abs/2207.04901
5. Lee, N., Sreenivasan, K., Lee, J. D., Lee, K. & Papailiopoulos, D. Teaching Arithmetic to Small Transformers. In International Conference on Learning Representations (ICLR 2024) (2024). https://arxiv.org/abs/2307.03381
6. Zhou, H. et al. What Algorithms can Transformers Learn? A Study in Length Generalization. In International Conference on Learning Representations (ICLR 2024) (2024). https://arxiv.org/abs/2310.16028
7. Zhou, Y. et al. Transformers Can Achieve Length Generalization But Not Robustly. Preprint at arXiv:2402.09371 (2024). https://arxiv.org/abs/2402.09371
8. McLeish, S. et al. Transformers Can Do Arithmetic with the Right Embeddings. In Advances in Neural Information Processing Systems, 108012–108041 (2024). https://doi.org/10.52202/079017-3430
9. Cho, H. et al. Position Coupling: Improving Length Generalization of Arithmetic Transformers Using Task Structure. In Advances in Neural Information Processing Systems, 22233–22315 (2024). https://doi.org/10.52202/079017-0701
10. Schick, T. et al. Toolformer: Language Models Can Teach Themselves to Use Tools. In Advances in Neural Information Processing Systems, 68539–68551 (2023). https://doi.org/10.52202/075280-2997
11. Gao, L. et al. PAL: Program-aided Language Models. In Proceedings of the 40th International Conference on Machine Learning, 10764–10799 (2023). https://arxiv.org/abs/2211.10435
12. Chen, W., Ma, X., Wang, X. & Cohen, W. W. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. Transactions on Machine Learning Research (2023). https://arxiv.org/abs/2211.12588
13. Gou, Z. et al. ToRA: A Tool-Integrated Reasoning Agent for Mathematical Problem Solving. In International Conference on Learning Representations (ICLR 2024) (2024). https://arxiv.org/abs/2309.17452
14. Feng, J. et al. ReTool: Reinforcement Learning for Strategic Tool Use in LLMs. In International Conference on Learning Representations (ICLR 2026) (2026). https://arxiv.org/abs/2504.11536
15. Moore, E. F. Gedanken-Experiments on Sequential Machines. In Automata Studies, 129–153 (1956). https://doi.org/10.1515/9781400882618-006
16. Cai, J., Shin, R. & Song, D. Making Neural Programming Architectures Generalize via Recursion. In International Conference on Learning Representations (ICLR 2017) (2017). https://arxiv.org/abs/1704.06611
17. Lewandowski, A., Machado, M. C. & Schuurmans, D. Universal computation is intrinsic to language model decoding. Preprint at arXiv:2601.08061 (2026). https://arxiv.org/abs/2601.08061
18. Hu, E. J. et al. LoRA: Low-Rank Adaptation of Large Language Models. In International Conference on Learning Representations (ICLR 2022) (2022). https://arxiv.org/abs/2106.09685
19. Qwen Team. Qwen3.5: Towards Native Multimodal Agents. Qwen blog (2026). https://qwen.ai/blog?id=qwen3.5
20. Yang, S., Kautz, J. & Hatamizadeh, A. Gated Delta Networks: Improving Mamba2 with Delta Rule. In International Conference on Learning Representations (ICLR 2025) (2025). https://arxiv.org/abs/2412.06464
21. Li, Z. & Hoiem, D. Learning without Forgetting. IEEE Transactions on Pattern Analysis and Machine Intelligence **40**, 2935–2947 (2018). https://doi.org/10.1109/TPAMI.2017.2773081
22. ONNX Runtime developers. ONNX Runtime. Software. Source https://github.com/microsoft/onnxruntime, MIT license; citation per the repository CITATION.cff (date-released 29 November 2018); the demo uses onnxruntime-web 1.26.0-dev.20260416 through Transformers.js 4.2.0 (2018). https://onnxruntime.ai
23. Hugging Face. Transformers.js. GitHub repository. Software, version 4.2.0 (released 23 April 2026), Apache-2.0 (2026). https://github.com/huggingface/transformers.js
24. Ninomiya, K., Jones, B. & Blandy, J. WebGPU. W3C Candidate Recommendation Draft, 15 September 2026. Latest version: https://www.w3.org/TR/webgpu/ (2026). https://www.w3.org/TR/2026/CRD-webgpu-20260915/
25. Knuth, D. E. The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Third edn, Addison-Wesley (1997).
26. Kaltofen, E. & Trager, B. M. Computing with Polynomials Given by Black Boxes for Their Evaluations: Greatest Common Divisors, Factorization, Separation of Numerators and Denominators. Journal of Symbolic Computation **9**, 301–320 (1990). https://doi.org/10.1016/S0747-7171(08)80015-6
27. Kronecker, L. Grundzüge einer arithmetischen Theorie der algebraischen Grössen. Journal für die reine und angewandte Mathematik **92**, 1–122 (1882). https://doi.org/10.1515/crll.1882.92.1
28. Zhou, J. et al. Instruction-Following Evaluation for Large Language Models. Preprint at arXiv:2311.07911 (2023). https://arxiv.org/abs/2311.07911
29. McCloskey, M. & Cohen, N. J. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In Psychology of Learning and Motivation, 109–165 (1989). https://doi.org/10.1016/S0079-7421(08)60536-8
30. Kirkpatrick, J. et al. Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of Sciences **114**, 3521–3526 (2017). https://doi.org/10.1073/pnas.1611835114
31. Agarwal, R. et al. On-Policy Distillation of Language Models: Learning from Self-Generated Mistakes. In International Conference on Learning Representations (ICLR 2024) (2024). https://arxiv.org/abs/2306.13649

32. Lu, K. & Thinking Machines Lab. On-Policy Distillation. Thinking Machines Lab: Connectionism. Blog post, 27 October 2025 (2025). https://doi.org/10.64434/tml.20251026
33. Shenfeld, I., Pari, J. & Agrawal, P. RL's Razor: Why Online Reinforcement Learning Forgets Less. In International Conference on Learning Representations (ICLR 2026) (2026). https://arxiv.org/abs/2509.04259
34. Shao, Z. et al. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. Preprint at arXiv:2402.03300 (2024). https://arxiv.org/abs/2402.03300
35. Cobbe, K. et al. Training Verifiers to Solve Math Word Problems. Preprint at arXiv:2110.14168 (2021). https://arxiv.org/abs/2110.14168
36. Hendrycks, D. et al. Measuring Massive Multitask Language Understanding. In International Conference on Learning Representations (ICLR 2021) (2021). https://arxiv.org/abs/2009.03300
37. nostalgebraist. interpreting GPT: the logit lens. LessWrong. Blog post, 31 August 2020 (2020). https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens
38. Belrose, N. et al. Eliciting Latent Predictions from Transformers with the Tuned Lens. Preprint at arXiv:2303.08112 (2023). https://arxiv.org/abs/2303.08112
39. Turner, A. M. et al. Steering Language Models With Activation Engineering. Preprint at arXiv:2308.10248 (2023). https://arxiv.org/abs/2308.10248
40. Rimsky, N. et al. Steering Llama 2 via Contrastive Activation Addition. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 15504–15522 (2024). https://doi.org/10.18653/v1/2024.acl-long.828
41. Abbe, E., Bengio, S., Lotfi, A., Sandon, C. & Saremi, O. How Far Can Transformers Reason? The Globality Barrier and Inductive Scratchpad. In Advances in Neural Information Processing Systems, 27850–27895 (2024). https://doi.org/10.52202/079017-0874
42. Yang, C., Srebro, N., McAllester, D. & Li, Z. PENCIL: Long Thoughts with Short Memory. In Proceedings of the 42nd International Conference on Machine Learning, 71092–71156 (2025). https://arxiv.org/abs/2503.14337
43. Rajani, N. et al. No Robots. Hugging Face dataset. Dataset HuggingFaceH4/no_robots, CC BY-NC 4.0 (2023). https://huggingface.co/datasets/HuggingFaceH4/no_robots
44. Kaiser, Ł. & Sutskever, I. Neural GPUs Learn Algorithms. In International Conference on Learning Representations (ICLR 2016) (2016). https://arxiv.org/abs/1511.08228
45. Price, E., Zaremba, W. & Sutskever, I. Extensions and Limitations of the Neural GPU. Preprint at arXiv:1611.00736 (2016). https://arxiv.org/abs/1611.00736
46. Trask, A. et al. Neural Arithmetic Logic Units. In Advances in Neural Information Processing Systems (2018). https://arxiv.org/abs/1808.00508
47. Madsen, A. & Johansen, A. R. Neural Arithmetic Units. In International Conference on Learning Representations (ICLR 2020) (2020). https://arxiv.org/abs/2001.05016
48. Liu, T. & Low, B. K. H. Goat: Fine-tuned LLaMA Outperforms GPT-4 on Arithmetic Tasks. Preprint at arXiv:2305.14201 (2023). https://arxiv.org/abs/2305.14201
49. Yang, Z. et al. GPT Can Solve Mathematical Problems Without a Calculator. Preprint at arXiv:2309.03241 (2023). https://arxiv.org/abs/2309.03241
50. Shen, S., Shen, P. & Zhu, D. RevOrder: A Novel Method for Enhanced Arithmetic in Language Models. Preprint at arXiv:2402.03822 (2024). https://arxiv.org/abs/2402.03822
51. Charton, F. Learning the greatest common divisor: explaining transformer predictions. In International Conference on Learning Representations (ICLR 2024) (2024). https://arxiv.org/abs/2308.15594
52. Kasliwal, S. Algorithmic Scratchpads and Curriculum Staging for Arithmetic Reasoning in Tiny Transformers. Preprint at arXiv:2610.09003 (2026). https://arxiv.org/abs/2610.09003
53. Wei, Z. On the Mirage of Long-Range Dependency, with an Application to Integer Multiplication. Preprint at arXiv:2603.29069 (2026). https://arxiv.org/abs/2603.29069
54. Wei, Z. On the Spatiotemporal Dynamics of Generalization in Neural Networks. Preprint at arXiv:2602.01651 (2026). https://arxiv.org/abs/2602.01651
55. Papazov, H., D'Angelo, F. & Flammarion, N. Exact Learning of Arithmetic with Differentiable Agents. Preprint at arXiv:2511.22751 (2025). https://arxiv.org/abs/2511.22751
56. Quirke, P., Neo, C. & Barez, F. Understanding Addition and Subtraction in Transformers. Preprint at arXiv:2402.02619 (2024). https://arxiv.org/abs/2402.02619
57. Back de Luca, A., Giapitzakis, G. & Fountoulakis, K. Learning to Add, Multiply, and Execute Algorithmic Instructions Exactly with Neural Networks. In Advances in Neural Information Processing Systems, 87469–87513 (2025). https://doi.org/10.52202/085713-2636
58. Lan, N., Geyer, M., Chemla, E. & Katzir, R. Minimum Description Length Recurrent Neural Networks. Transactions of the Association for Computational Linguistics **10**, 785–799 (2022). https://doi.org/10.1162/tacl_a_00489
59. Ito, T., Puri, R., Campbell, M. & Ram, P. Universal Transformers for Circuit Computations: Perfect Length Generalization in Tiny Transformers. Preprint at arXiv:2608.31067 (2026). https://arxiv.org/abs/2608.31067
60. Reed, S. & de Freitas, N. Neural Programmer-Interpreters. In International Conference on Learning Representations (ICLR 2016) (2016). https://arxiv.org/abs/1511.06279

61. Zaremba, W., Mikolov, T., Joulin, A. & Fergus, R. Learning Simple Algorithms from Examples. In Proceedings of the 33rd International Conference on Machine Learning, 421–429 (2016). https://arxiv.org/abs/1511.07275
62. Lai, J. et al. Executing Arithmetic: Fine-Tuning Large Language Models as Turing Machines. Preprint at arXiv:2410.07896 (2024). https://arxiv.org/abs/2410.07896
63. Dugan, O. et al. OccamLLM: Fast and Exact Language Model Arithmetic in a Single Step. In Advances in Neural Information Processing Systems, 35665–35699 (2024). https://doi.org/10.52202/079017-1125
64. Dietz, F. & Klakow, D. IGC: Integrating a Gated Calculator into an LLM to Solve Arithmetic Tasks Reliably and Efficiently. Preprint at arXiv:2501.00684 (2025). https://arxiv.org/abs/2501.00684
65. Giannou, A. et al. Looped Transformers as Programmable Computers. In Proceedings of the 40th International Conference on Machine Learning, 11398–11442 (2023). https://arxiv.org/abs/2301.13196
66. Turkcan, M. K. Loom: A Scalable Analytical Neural Computer Architecture. Preprint at arXiv:2604.08816 (2026). https://arxiv.org/abs/2604.08816
67. Percepta. transformer-vm. GitHub repository. Software (Apache-2.0): compile programs directly into transformer weights; repository created 25 March 2026; accessed 31 October 2026 (2026). https://github.com/Percepta-Core/transformer-vm
68. Weng, Y. et al. Mastering Symbolic Operations: Augmenting Language Models with Compiled Neural Networks. In International Conference on Learning Representations (ICLR 2024) (2024). https://arxiv.org/abs/2304.01665
69. Xu, R., Yang, C., Li, Y. & McAllester, D. Towards Training Transformers as a Universal Computer. In Findings of the Association for Computational Linguistics: EMNLP 2026, 48933–48954 (2026). https://doi.org/10.18653/2026.findings-emnlp.2284
70. Li, C. et al. Chain of Code: Reasoning with a Language Model-Augmented Code Emulator. In Proceedings of the 41st International Conference on Machine Learning, 28259–28277 (2024). https://arxiv.org/abs/2312.04474
71. Qian, C. et al. ToolRL: Reward is All Tool Learning Needs. In Advances in Neural Information Processing Systems, 116896–116926 (2025). https://doi.org/10.52202/085713-3524
72. Zhang, A. L., Kraska, T. & Khattab, O. Recursive Language Models. Preprint at arXiv:2512.24601 (2025). https://arxiv.org/abs/2512.24601
73. Katz, G., Barrett, C., Dill, D. L., Julian, K. & Kochenderfer, M. J. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In Computer Aided Verification (CAV 2017), 97–117 (2017). https://doi.org/10.1007/978-3-319-63387-9_5
74. Gross, J. et al. Compact Proofs of Model Performance via Mechanistic Interpretability. In Advances in Neural Information Processing Systems, 79453–79515 (2024). https://doi.org/10.52202/079017-2523
75. Chow, T. S. Testing Software Design Modeled by Finite-State Machines. IEEE Transactions on Software Engineering **SE-4**, 178–187 (1978). https://doi.org/10.1109/TSE.1978.231496
76. Schuurmans, D., Dai, H. & Zanini, F. Autoregressive Large Language Models are Computationally Universal. Preprint at arXiv:2410.03170 (2024). https://arxiv.org/abs/2410.03170
77. Rodionov, G. & Prokhorenkova, L. Discrete Neural Algorithmic Reasoning. In Proceedings of the 42nd International Conference on Machine Learning (ICML 2025), 51851–51865 (2025). https://arxiv.org/abs/2402.11628
78. Back de Luca, A. & Fountoulakis, K. Certification from Examples is Hard for Circuits and Transformers under Minimal Overparametrization. Preprint at arXiv:2605.22964 (2026). https://arxiv.org/abs/2605.22964
79. Yang, A., Bergsträßer, P., Zetzsche, G., Chiang, D. & Lin, A. W. Length Generalization Bounds for Transformers. In Proceedings of the 43rd International Conference on Machine Learning, 144227–144248 (2026). https://arxiv.org/abs/2603.02238
80. Rust, P. et al. Language Modelling with Pixels. In International Conference on Learning Representations (ICLR 2023) (2023). https://arxiv.org/abs/2207.06991
81. Tschannen, M., Mustafa, B. & Houlsby, N. CLIPPO: Image-and-Language Understanding from Pixels Only. In 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 11006–11017 (2023). https://doi.org/10.1109/CVPR52729.2023.01059
82. Lee, K. et al. Pix2Struct: Screenshot Parsing as Pretraining for Visual Language Understanding. In Proceedings of the 40th International Conference on Machine Learning, 18893–18912 (2023). https://arxiv.org/abs/2210.03347
83. Wei, H., Sun, Y. & Li, Y. DeepSeek-OCR: Contexts Optical Compression. Preprint at arXiv:2510.18234 (2025). https://arxiv.org/abs/2510.18234
84. Cheng, J. et al. Glyph: Scaling Context Windows via Visual-Text Compression. In Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 37145–37158 (2026). https://doi.org/10.18653/v1/2026.acl-long.1722
85. Hoshen, Y. & Peleg, S. Visual Learning of Arithmetic Operations. In Proceedings of the AAAI Conference on Artificial Intelligence, 3733–3739 (2016). https://doi.org/10.1609/aaai.v30i1.9882
86. Balter, S. G., Jerzak, E. & Jerzak, C. T. Multiplication in Multimodal LLMs: Computation with Text, Image, and Audio Inputs. In Findings of the Association for Computational Linguistics: ACL 2026, 40766–40780 (2026). https://doi.org/10.18653/v1/2026.findings-acl.2025
87. Hu, Y. et al. Visual Sketchpad: Sketching as a Visual Chain of Thought for Multimodal Language Models. In Advances in Neural Information Processing Systems, 139348–139379 (2024). https://doi.org/10.52202/

079017-4423

88. Menon, S., Zemel, R. & Vondrick, C. Whiteboard-of-Thought: Thinking Step-by-Step Across Modalities. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, 20016–20031 (2024). https://doi.org/10.18653/v1/2024.emnlp-main.1117
89. Li, C. et al. Imagine While Reasoning in Space: Multimodal Visualization-of-Thought. In Proceedings of the 42nd International Conference on Machine Learning, 36340–36364 (2025). https://arxiv.org/abs/2501.07542
90. Akter, S. N., Madaan, A., Lee, S., Yang, Y. & Nyberg, E. Self-Imagine: Effective Unimodal Reasoning with Multimodal Models using Self-Imagination. Preprint at arXiv:2401.08025 (2024). https://arxiv.org/abs/2401.08025
91. Shi, W. et al. MathCanvas: Intrinsic Visual Chain-of-Thought for Multimodal Mathematical Reasoning. In Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 27933–27954 (2026). https://doi.org/10.18653/v1/2026.acl-long.1287
92. Sutherland, I. E. Sketchpad: A Man-Machine Graphical Communication System. PhD thesis, Massachusetts Institute of Technology (1963). http://hdl.handle.net/1721.1/14979
93. Lample, G. & Charton, F. Deep Learning for Symbolic Mathematics. In International Conference on Learning Representations (ICLR 2020) (2020). https://arxiv.org/abs/1912.01412
94. Agarwal, V., Aditya, S. & Goyal, N. Analyzing the Nuances of Transformers' Polynomial Simplification Abilities. Preprint at arXiv:2104.14095 (2021). https://arxiv.org/abs/2104.14095
95. Kera, H., Ishihara, Y., Kambe, Y., Vaccon, T. & Yokoyama, K. Learning to compute Gröbner bases. In Advances in Neural Information Processing Systems, 33141–33187 (2024). https://doi.org/10.52202/079017-1044
96. Lewkowycz, A. et al. Solving Quantitative Reasoning Problems with Language Models. In Advances in Neural Information Processing Systems, 3843–3857 (2022). https://doi.org/10.52202/068431-0278
97. Feng, G. et al. Towards Revealing the Mystery behind Chain of Thought: A Theoretical Perspective. In Advances in Neural Information Processing Systems, 70757–70798 (2023). https://doi.org/10.52202/075280-3100
98. Rusu, A. A. et al. Progressive Neural Networks. Preprint at arXiv:1606.04671 (2016). https://arxiv.org/abs/1606.04671
99. Buehler, E. L. & Buehler, M. J. X-LoRA: Mixture of low-rank adapter experts, a flexible framework for large language models with applications in protein mechanics and molecular design. APL Machine Learning **2** (2024). https://doi.org/10.1063/5.0203126
100. Greenewald, K. et al. Activated LoRA: Fine-tuned LLMs for Intrinsics. In Advances in Neural Information Processing Systems, 59025–59052 (2025). https://doi.org/10.52202/085713-1769
101. Tonmoy, A. B., Hoque, K. F., Arham, M. S. I. & Luthra, A. Modality-Gated Deep Adapters: Adding a Modality to a Frozen Embedding Model with Exact Preservation. Preprint at arXiv:2609.26182 (2026). https://arxiv.org/abs/2609.26182
102. de Moura, L. & Ullrich, S. The Lean 4 Theorem Prover and Programming Language. In Automated Deduction – CADE 28, 625–635 (2021). https://doi.org/10.1007/978-3-030-79876-5_37
103. The mathlib Community. The Lean Mathematical Library. In Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020), 367–381 (2020). https://doi.org/10.1145/3372885.3373824