Arithmetic Without Numbers (Draft)

What happens inside an LLM when it tries to calculate with nothing but matrices

ALVARO VIDELA

A language model has no fingers, no beads, no scratch paper, and no number line. It has matrices. This article asks what kind of arithmetic grows inside that body: phase-like geometry rather than written columns, a residual stream rather than paper, and failure modes shaped by left-to-right token emission and finite representational resolution. The strongest verified result is scoped to one frozen 8B Llama: an activation-derived route can read operation and operand arguments from hidden states under a strict no-parser runtime. The point is not that the model was made into an exact calculator. The point is how carefully one has to work before saying that the calculation was really present inside.

> This PDF is the static article. The animated companion, with the helix slider and browser-side simulations, is available at https://alvaro-videla.com/llm-arithmetic-internals/article_interactive/.

1 THE QUESTION

If you learned arithmetic the ordinary human way, you probably learned it with a body. You counted on fingers, grouped things into piles, lined digits into columns, and carried a one. Human arithmetic is full of objects: fingers, beads, marks, columns, strokes, and places.

A language model has none of that. It has matrices.

At each layer, a grid of numbers transforms another grid of numbers. Tokens enter, activations flow, logits come out. No fingers. No abacus. No column of digits written on a page. And yet, if you ask a modern language model for the greatest common divisor of two numbers, or a multiplication, or a division with remainder, something inside that matrix-only body responds.

Sometimes it answers correctly. Often it does not. The question that started Rune was simple to ask and hard to answer: how does a language model do arithmetic, if all it has are vectors?

The first temptation is to treat this as a product problem. If the model is bad at arithmetic, call a calculator. A parser can read "what is 84 times 37," translate it into `84 * 37`, send that expression to Python, and return the result. Useful, but not much of a mystery. That general family of external-computation routes is well established: PAL and Program of Thoughts ask models to produce executable programs, ReAct interleaves reasoning with actions, and Toolformer studies models learning when and how to call APIs [2, 4, 9, 11].

Rune was chasing a different question. Could we look inside the model and find the calculation it was trying to perform? Could the model's own internal states tell us the operation and operands? And, if so, could we use that information without cheating by reading the prompt text directly?

The promise of the article is this: a matrix-only body can invent ways to represent divisibility, magnitude, digit chunks, and numeric emission. The model did not conjure arithmetic from nothing; it distilled patterns from human text about human mathematics. The surprise is that gradient descent, given our symbols and a matrix body, settled on representations no human would learn with fingers.

2 A DIFFERENT KIND OF TOOL USE

The original dream was more ambitious: a mechanism-aware just-in-time compiler for model arithmetic. The ideal pipeline looked like this. The model reads an arithmetic prompt; we identify the internal mechanism it is using; we replace the unreliable part with exact computation; then the model continues naturally, as if it had done the math itself.

It is a seductive pipeline, and it is not what Rune proves.

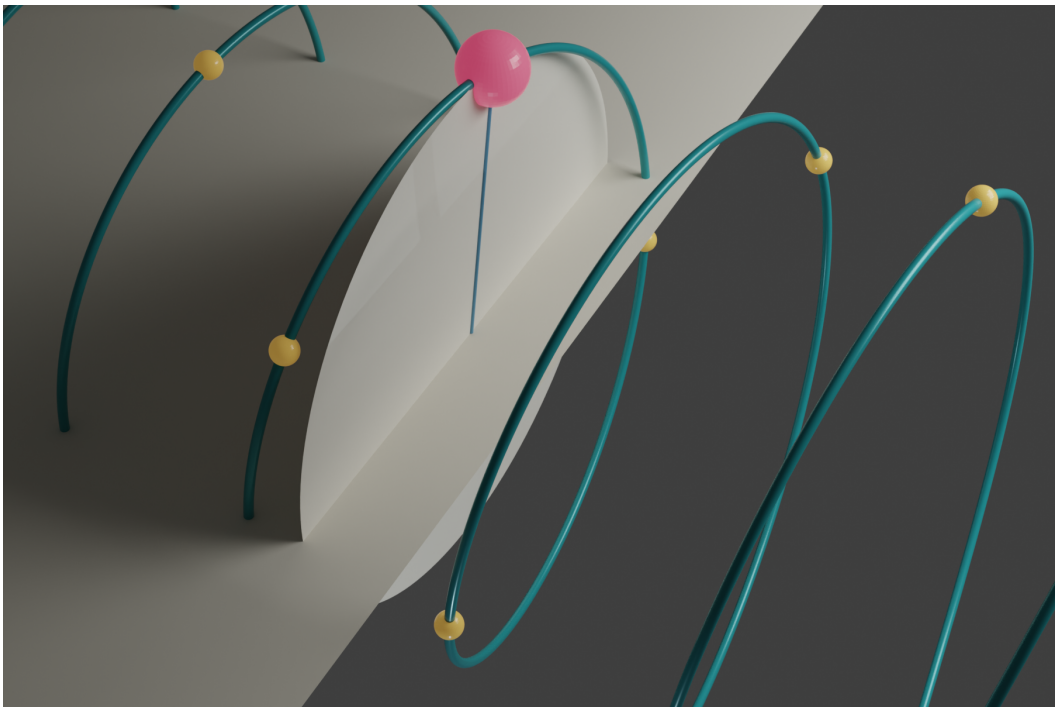

Fig. 1. A visual analogy for integer phase geometry in a matrix-only model: value can be represented by position on rotating directions rather than by fingers, beads, or written columns.

What survived the experiments was narrower, but more interesting than ordinary tool use. In a frozen Llama model, internal activations expose enough structure to recover the arithmetic operation and operands under a strict no-parser rule. The runtime is not allowed to use regexes, prompt text, hidden labels, command-line operands, or gold answers. It gets token IDs and activations. From those activations, it must infer: this is gcd, or lcm, or multiplication, or division with remainder; these are the two numbers. Only after that decoded tuple exists may Python compute.

The calculator is not the surprising part. The surprising part is that its arguments can come from inside the model.

A normal tool route can solve the problem without learning anything about model internals. Rune asks whether the model's residual stream already carries the calculation description, even when the model's own next-token answer would be wrong.

The no-parser rule is therefore epistemic, not merely procedural. It exists so that later, when the article says the hidden state contained the calculation, the reader does not have to wonder whether a regex, a hidden operand field, or a gold answer smuggled the result into the route. The supported claim is narrow: activation-derived op and operand readouts can supply calculator arguments for an opaque no-parser route on the current frozen Llama benchmark slices, without also proving native arithmetic repair, residual-state JIT replacement, or cross-model transfer.

The production operand route used a frozen layer-22 chunk path: selected operand chunks, with attention-based selection, were decoded by the project chunk probe. That matters because reading operands directly from their input-token positions in early layers would be too close to parsing in disguise. Layer-22 chunk selection is a stronger internal-state claim, though still a readout claim

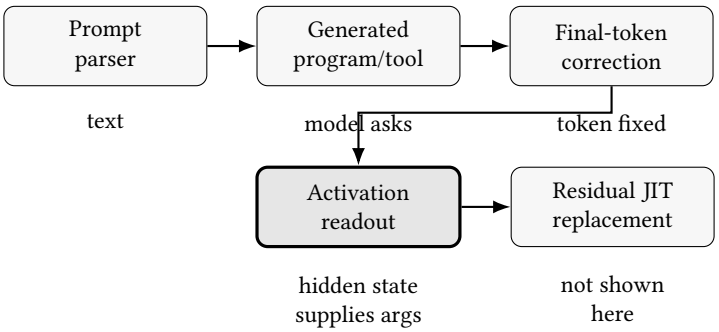

Fig. 2. Claim ladder for the same visible answer. Rune's supported claim is the highlighted activation-readout route: the calculator arguments come from hidden vectors, not from prompt parsing or an already-known answer.

rather than proof of writable replacement. The exact probe artifact is listed in the repository's experiment notes.

3 THE MATRIX BODY

George Lakoff and Rafael Núñez argued that human mathematics is built from embodied experience: grouping, moving, measuring, collecting, and balancing [6]. A transformer has a different body. Its body is matrices, residual streams, attention maps, and learned projections. If human arithmetic can grow out of fingers and spatial metaphors, what kind of arithmetic grows out of a matrix-only body?

Recent mechanistic work suggests a geometric answer. Kantamneni and Tegmark argued that language models can represent integers on generalized helices and use trigonometric structure for addition [5]. Stolfo, Belinkov, and Sachan used causal mediation analysis to study arithmetic mechanisms in transformers [10]. Other work, including Nikankin and colleagues' "bag of heuristics" account, warns that model arithmetic is often not a clean schoolbook algorithm but a mixture of learned features and prompt-sensitive heuristics [7].

Rune builds on that line rather than originating it. The contribution here is a scoped reproduction-and-extension story around activation-derived tool arguments, provenance controls, and resolution limits. We found Fourier-like readouts and value-like states, but also brittleness, prompt sensitivity, and failures that look less like a perfect algorithm than a finite-resolution internal geometry.

The body mapping is the useful way to read the rest of the story. Fingers, beads, and tally marks become directions in activation space. The number line and human motion-along-a-path metaphors become phase rotation on a helix, close to Kantamneni and Tegmark's clock-like picture. Scratch paper becomes the residual stream, a shared scratchpad with no named variables. Right-to-left carry algorithms are constrained by a body that must emit text left to right. Running out of paper or column width becomes a resolution budget: nearby chunk readout directions crowd together.

3.1 A Vector Changes as the Model Reads

Imagine reading `What is the gcd of 84 and 36?` one token at a time. The model does not create neat variables named `operand_a` and `operand_b`. Each token position carries a long vector. As the prompt passes through transformer layers, those vectors are updated repeatedly. Some updates move information across positions: the token for `36` can affect the state near the answer position.

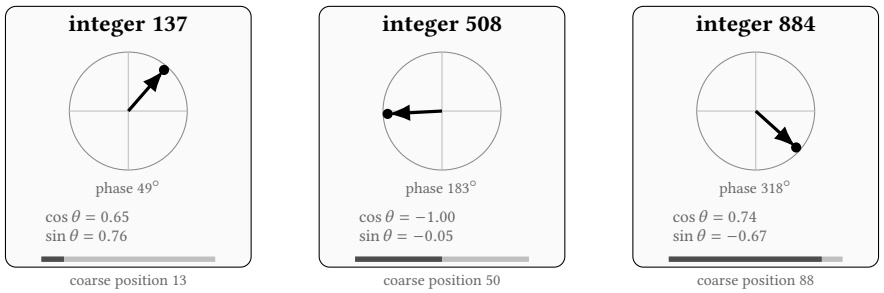

Fig. 3. Static counterpart to the animated helix control. The interactive page lets the reader drag the integer value; in print, three snapshots show the same idea. A value can be represented by phase around a cycle, by sine/cosine coordinates of that phase, and by a coarser scale coordinate.

Table 1. The same arithmetic task, two different bodies. The analogy is not literal anatomy; it is a guide to which representations and failures each body makes natural.

Human arithmetic body	Matrix arithmetic body
Fingers, beads, tally marks	Directions in activation space
Number line and motion along a path	Phase rotation on a helix
Scratch paper and written columns	Residual stream with no named variables
Right-to-left carries	Left-to-right token emission
Running out of column width	Crowded chunk readout subspaces

Other updates reshape the local state: a direction in the vector may become more gcd-like, more operand-like, or more answer-like.

The terms are less exotic in context. A token is one unit the model reads or prints; for numbers it may be a chunk of digits rather than a full integer. The activation is the temporary vector at a token position. The residual stream is the running vector passed from layer to layer, a shared scratchpad without named variables. Attention lets one position gather information from others; the MLP, or feed-forward block, then reshapes that position's vector locally. At the end, logits score possible next tokens.

This is why readouts and patches are possible at all. Operation and operands can leave traces in the residual stream, and a small readout may recover them. A patch asks a stronger question: whether moving a state changes behavior. A writable intervention is stronger still. The vector itself does not label which claim is true.

4 NEXT-TOKEN PRESSURE

Next-token prediction changes the arithmetic problem because it changes the body. A human doing subtraction on paper usually works from the least significant digit toward the left: units first, then tens, then hundreds, carrying or borrowing as needed. Paper permits that. A language model must emit text from left to right. For the answer `15696`, it must commit to the visible prefix before the suffix exists. In tokenized form, the answer may come out as chunks like `15` then `696`. The model is not filling in a scratch sheet from right to left; it is walking forward through the string.

That left-to-right constraint creates a rendering problem separate from the arithmetic problem. The model may have partial information about the answer but still lose precision as it emits longer

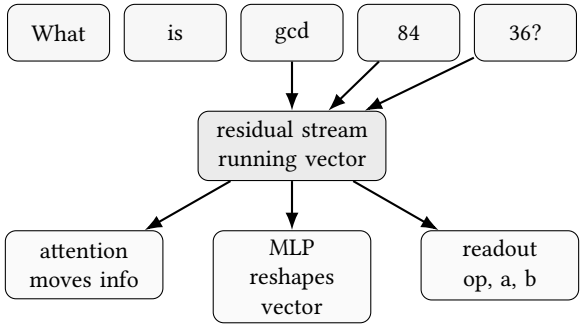

Fig. 4. The residual stream is a running vector, not a named variable table. Attention can move operand information across positions; the MLP reshapes the local vector; a readout asks whether facts such as `gcd, 84, 36` are recoverable from that state.

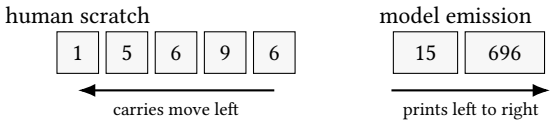

Fig. 5. Humans can compute from the rightmost digit and write the answer later. A next-token model must emit the visible prefix first. That makes numeric rendering a separate pressure from internal arithmetic.

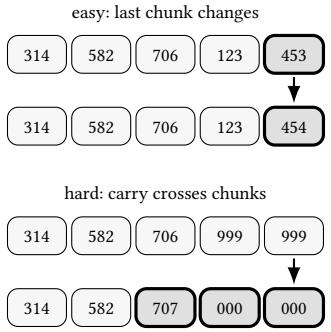

Fig. 6. The counting experiment exposed the same left-to-right pressure in token chunks. The easy case edits one final chunk; the hard case must update several chunks before the model has printed them.

strings. In Rune's subtraction scaling experiment, exact greedy generation stayed at 96.7 percent for 6-digit subtraction, fell to 63.3 percent at 10 digits, reached 53.3 percent at 13 digits, and crossed below 50 percent at 14 digits. At 24 digits it was down to 6.7 percent.

A separate counting experiment made the same pressure visible in a simpler setting. The model saw runs of large consecutive numbers and had to print the next one. An easy case simply advances the last chunk. A deep-carry case must change multiple chunks at once. Cases like this collapsed; the best deep-carry cell reached only 18.75 percent accuracy, and the dominant error was stuck-copy. In this body, arithmetic failure need not look like a wrong carried one in a paper column. It can look like a visible token stream copying old chunks because the carry has to cross boundaries before the model can print them.

Table 2. Counting continuation example. The model tokenizer splits the visible number into chunks; the hard case requires a carry across two chunk boundaries.

Case	Chunks
Easy next	`314 | 582 | 706 | 123 | 453 -> 314 | 582 | 706 | 123 | 454`
Deep carry	`314 | 582 | 706 | 999 | 999 -> 314 | 582 | 707 | 000 | 000`
Stuck-copy error	`314 | 582 | 706 | 999 | 999 -> 314 | 582 | 706 | 999 | 999`

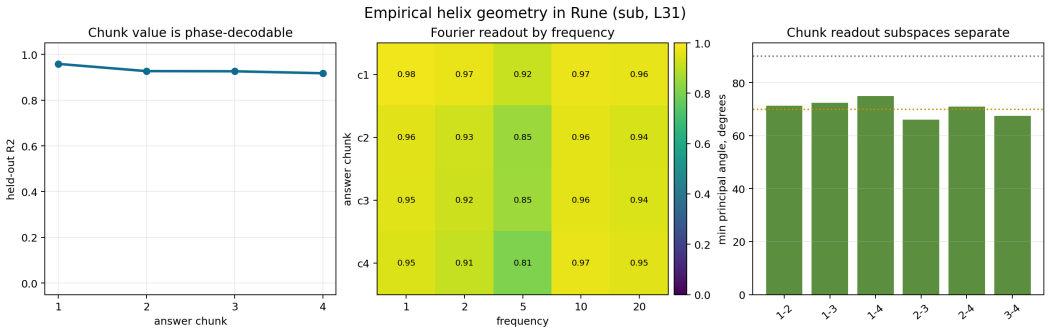

Fig. 7. Empirical helix-resolution evidence for multi-chunk subtraction. Longer answers remain partly readable, but adjacent chunk subspaces become more crowded and less forgiving.

Table 3. Subtraction exact-match rate by answer digit count. Source: `cd_e10_operand_scaling`, 30 greedy generations per digit band.

Digits	Exact	Prefix correct	Mean edit distance
6	96.67%	97.2%	0.10
10	63.33%	80.7%	0.60
13	53.33%	73.8%	1.07
14	43.33%	67.1%	1.07
16	33.33%	64.6%	1.40
24	6.67%	17.6%	8.77

The helix-resolution experiments gave a more mechanistic picture of that boundary. For 12-digit subtraction answers split into four 3-digit chunks, each chunk remained strongly phase-decodable at layer 31: R^2—roughly, how much of the chunk's value a simple readout recovered—was about 0.96 for chunk 1 and about 0.92 for chunk 4. So the signal did not simply vanish. The subtler finding was crowding. Adjacent chunk readout subspaces were not cleanly orthogonal; the minimum principal angle, a measure of how separable two nearby readout directions are, dropped as low as 66 degrees. Smaller angles mean less separation. In a 14-digit, five-chunk follow-up, chunks 2–4 lost R^2 and adjacent angles tightened further, for example c3–c4 went from 67.4 degrees to 58.8 degrees. Two of three preregistered crowding predictions passed.

The phrase "the helix gets saturated" is too crude. The cleaner claim is a resolution budget: longer answers still live in readable geometry, while nearby chunk directions crowd together. This is the matrix body's version of running out of column width.

Table 4. Adjacent chunk subspace angles tightened in the 14-digit follow-up. Smaller angles mean less separation between nearby readout directions.

Chunk pair	12-digit	14-digit
c1–c2	71.3°	65.3°
c2–c3	66.0°	60.2°
c3–c4	67.4°	58.8°

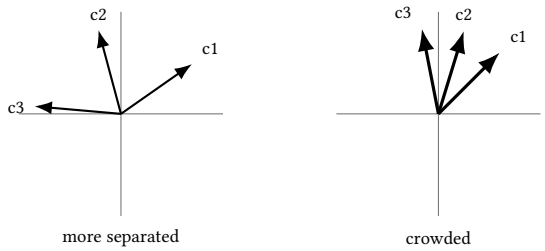

Fig. 8. Crowding is a geometric resolution problem. In the experiments this was measured with principal angles between chunk readout subspaces; the drawing is only a visual guide to what smaller angles mean.

5 WHAT FAILED USEFULLY

At first, this looked like a path toward the compiler dream. If we could write the right answer state into the model, perhaps the model would continue from there.

The evidence sharpened that hope into something more modest. Making a model emit a known answer is not the same as making the model compute. If the experiment already knows the answer, and then uses that answer to choose a steering vector, it has measured the model's ability to render a supplied value. Useful, yes; but it tells us little about arithmetic understanding, and it would not be a deployable no-parser route.

This was one of the first major lessons: token rendering can masquerade as computation.

The residual-write story also became clearer. We tried to write corrected answer information into the model's residual stream and let it continue. For the tested single-site writes, residual interventions had no accuracy advantage over simpler token or logit correction. Worse, they disturbed surrounding behavior more. A readable variable is not necessarily a writable register; in these runs, reading the body was easier than operating on it.

Mechanistic interpretability often celebrates reading: decode a concept, localize a feature, find a direction. Sparse autoencoders and dictionary learning are one important approach to naming such features in dense activation spaces [1, 3]. Activation patching is one common way to move from readout toward causal evidence, but patching methodology has its own pitfalls and metric choices [12]. Engineering asks for the harsher version: change the state, preserve the rest of the behavior, and let execution continue.

6 THE INSPECTION TOOLBOX

Most of Rune was not one big experiment. It was a toolbox applied repeatedly, with stricter controls each time. The tools are useful, but each answers a different question.

This is why the article keeps distinguishing readout, causality, and writable replacement. A probe reads a fact from an activation. An SAE tries to name reusable features. A patch copies part

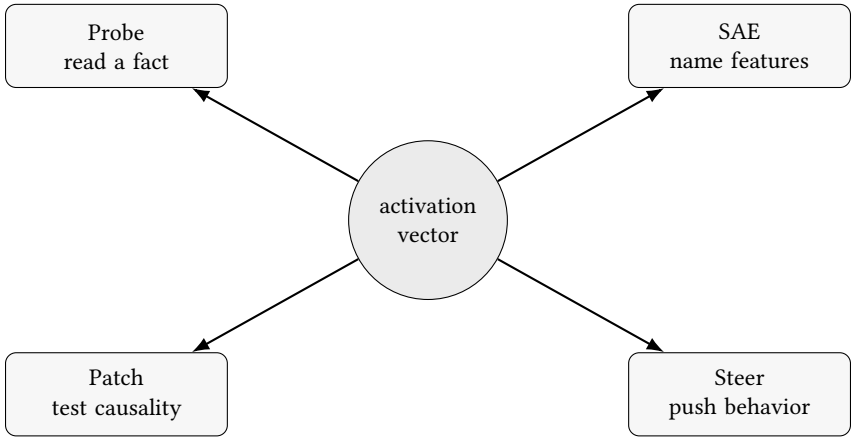

Fig. 9. The same activation can be inspected in different ways. The tools do not make the same claim: readout, feature naming, causal testing, and steering are different standards of evidence.

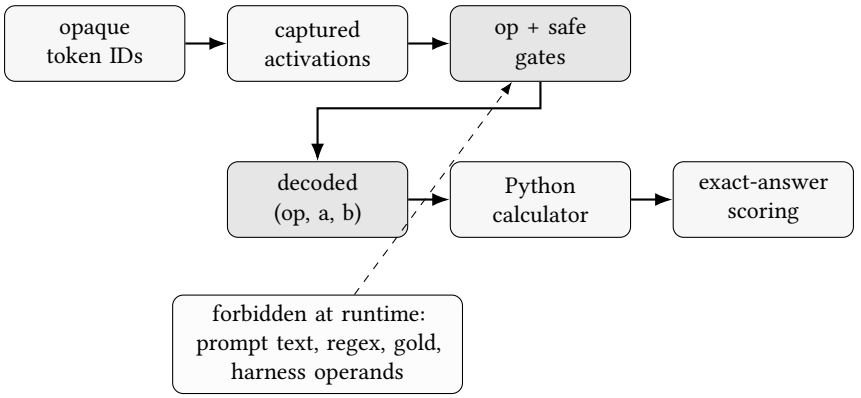

Fig. 10. The surviving route is activation-derived tool use. The calculator is ordinary; the provenance question is whether its arguments came from hidden state rather than forbidden prompt-derived fields.

of one run into another and watches whether behavior follows. Steering is the blunt instrument: push the state and see what moves, or what breaks. Treating those as the same claim is how interpretability experiments overclaim.

7 THE ROUTE THAT SURVIVED

The final route matters because it is the proof apparatus for the wonder. It does not make the model a better native calculator. It asks whether the matrix body really carries the calculation description strongly enough that a calculator can be called without reading the prompt text.

The route used activation-only gates and operand readouts. It had to decide when an arithmetic route was safe to fire, identify the operation, identify the operand pair, and then call Python only after the decoded tuple was formed. The final claim-bearing rerun was preregistered on June 2, 2026, with thresholds, operand bounds, and the benchmark-suite runner fixed in advance.

On a broad benchmark with the Llama weights kept frozen, the route passed across four operations: multiplication, division with remainder, gcd, and lcm. Here “passed” means two things at

Table 5. Broad frozen arithmetic/adversarial cross-seed aggregate on Llama-3.1-8B. Routed is exact-answer accuracy after activation-derived routing. Lift is routed exact accuracy minus the frozen model's native exact accuracy on target prompts, with non-fired targets counted as the route's produced answer. False-fire is the rate of unwanted calculator calls on prompts where the route should stay silent.

Operation	Routed	Lift	False-fire rate
multiplication	0.865	0.852	0.000
division with remainder	0.909	0.693	0.000
lcm	0.969	0.966	0.000
gcd	0.922	0.594	0.000

once. On real arithmetic prompts, the route had to fire: a gate had to decide that the calculator was allowed to run, and the operation and operands had to come from activations. On adversarial prompts, written to tempt the route into doing the wrong thing, it had to stay silent. Across 11,736 locked examples—with examples, thresholds, and scoring rules fixed before the final aggregate—and 1,536 targets, it produced large exact-answer lifts with zero fires on the constructed hard-negative suite used in this audit. A hard negative is a deliberately tricky no-fire prompt: it may contain quoted arithmetic, code, a table of numbers, or an instruction saying not to compute, but the correct behavior is no calculator call (Table 5).

The route's operand bounds were frozen at integers from 0 through 9999, with at most 12 generated answer tokens. In plain terms, the supported claim is two-integer arithmetic within that published route, not arbitrary long arithmetic.

On a recognized source slice from the DeepMind Mathematics Dataset of Saxton and colleagues [8], the result covered three operations: gcd, division with remainder, and lcm. Recognized means the audit could map a dataset prompt to one of the supported forms: two integer operands, a supported operation, and a checkable answer. It is a coverage filter; unsupported DeepMind prompts remain outside the claim. Multiplication was not claimed there because the source filtering did not produce enough accepted two-integer multiplication examples for a powered result. Across 3,822 locked examples and 1,233 targets, the activation-derived route again produced strong exact-answer lifts with zero recorded false fires.

In plain percentages, the accepted DeepMind slice is not a story about preserving answers the model already knew. The routed exact rates were 99.2 percent for division with remainder, 100 percent for gcd, and 98.0 percent for lcm, with exact-answer gains of 81.0, 50.2, and 96.8 percentage points over the frozen model's native answers.

The safety and provenance work mattered as much as the benchmark lifts. Provenance, concretely, means the audit trail for where the calculator arguments came from. We ran a full replay audit over 15,558 runtime bundles, excluding forbidden fields such as prompt text, regex outputs, decoded token spans, harness operands, operation labels, and gold answers. The route had to reproduce from allowed runtime artifacts. It passed with zero replay failures.

We also ran an independent hard-negative audit, using no-fire cases generated separately from the positive arithmetic benchmark. The categories included quoted arithmetic, do-not-compute prompts, wrong-operation prompts with the same numbers, tables, logs, code, invoices, distractor-heavy number text, decimals, signs, and out-of-domain cases. Across 10,200 non-trigger examples, the route did not fire. That zero is a strong scoped audit result, not a universal safety claim: the hard negatives were constructed within this project, and an independently generated adversarial suite would be the next stronger test.

Table 6. Examples of the fire/no-fire boundary. The route should fire on arithmetic requests, but not merely because arithmetic-looking text appears.

Should fire	Should not fire
`Calculate the highest common factor of 5924 and 1024.`	`She wrote 'gcd(48, 18) = 6' on the whiteboard and then changed the subject to budgets of 200 and 300.`
`What is the remainder when 7696 is divided by 5130?`	`A reporter typed '144 / 12' into her notes but the story was about a basketball game.`
`What is the smallest common multiple of 4740 and 1152?`	`The chart showed 6, 12, 18, 24 as factor labels but the article discussed musical notation.`

8 THE BOUNDARY

There is causal support, but it has to be said carefully. In causal interchange tests, selected internal operand chunks could be patched from donor examples into recipient examples, and the decoded operands and routed calculator answers followed the donor. This uses the same broad causal-intervention discipline as activation patching and causal mediation work, not a new causal methodology invented here [10, 12]. It supports the claim that those internal chunks are causally involved in the decoded tuple. In the final DeepMind causal summary, division with remainder cleared the powered pair-count gate. Gcd and lcm had perfect rates in the available pairs, but too few pairs to clear the frozen causal gate. So the causal evidence is supportive, not fully powered across every final operation.

The cross-model result was a stop sign. The real Qwen operand-localization diagnostic reports `QWEN_OPERAND_ROUTE_FAIL`: across the sampled answer-site and input-token positions, ordered and unordered pair recovery stayed at 0.000. The current Llama route did not transfer. Internal activation routes are not portable the way text parsers are. A parser sees the same string. A different model may have a different body, a different geometry, and therefore a different arithmetic.

The next steps are concrete: build model-specific operand localizers instead of assuming transfer; complete causal interchange tests where source coverage allows it; compare any residual-write attempt against boring logit and parser baselines; and keep the no-parser replay boundary as a non-negotiable test.

Here is the final claim without ceremony: on the current benchmark slices, a frozen Llama model's activations can supply operation and operand arguments for an exact calculator under an opaque, no-parser runtime. The result is a boundary claim. It leaves native arithmetic repair, general cross-model transfer, and behavior-preserving residual JIT compilation unsolved. Inside that boundary, the model's hidden state can contain the calculation even when the visible answer would be wrong, and the provenance rules are what keep that sentence honest.

9 WHAT TO TAKE AWAY

Models really do build arithmetic-shaped internal structure. It is not the arithmetic humans experience with fingers and written columns. It is a geometry of directions, rotations, periodic features, and value-like residual states: one way for a matrix-only body to represent divisibility, magnitude, chunks, and numeric emission.

Reading those structures is easier than turning them into a robust system. Probes can look good for the wrong reason; steering can render an answer without proving computation; residual writes can change a token while damaging the model's state. The useful science came from the controls that survived after the attractive first signals were made harder to fool.

There are three routes here, and they should not be graded as one achievement. Text-driven tool use parses the prompt and calls the tool. Activation-derived tool use proves that the tool arguments came from the model's internal state. Residual replacement writes the result back into the model and preserves behavior. Rune makes progress on the second route. The third remains open.

If you want to build systems from those traces, do not trust the first clean signal. Ask what else it could be measuring. Remove the prompt text. Remove the labels. Replay without the forbidden fields. Add the uncomfortable negatives. Compare against the plain baseline. Separate what the model represents from what you can safely write.

That is how the story gets smaller. It is also how it becomes true.

REFERENCES

[1] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E. Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. 2023. Towards Monosemanticity: Decomposing Language Models With Dictionary Learning. *Transformer Circuits Thread* (2023). https://transformer-circuits.pub/2023/monosemantic-features/

[2] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research* (2023). https://doi.org/10.48550/arXiv.2211.12588 arXiv:2211.12588.

[3] Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. 2023. Sparse Autoencoders Find Highly Interpretable Features in Language Models. *arXiv preprint arXiv:2309.08600* (2023). https://doi.org/10.48550/arXiv.2309.08600

[4] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. PAL: Program-aided Language Models. *arXiv preprint arXiv:2211.10435* (2022). https://doi.org/10.48550/arXiv.2211.10435

[5] Subhash Kantamneni and Max Tegmark. 2025. Language Models Use Trigonometry to Do Addition. *arXiv preprint arXiv:2502.00873* (2025). https://doi.org/10.48550/arXiv.2502.00873

[6] George Lakoff and Rafael E. Núñez. 2000. *Where Mathematics Comes From: How the Embodied Mind Brings Mathematics into Being*. Basic Books, New York, NY.

[7] Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. 2024. Arithmetic Without Algorithms: Language Models Solve Math With a Bag of Heuristics. *arXiv preprint arXiv:2410.21272* (2024). https://doi.org/10.48550/arXiv.2410.21272

[8] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing Mathematical Reasoning Abilities of Neural Models. *International Conference on Learning Representations* (2019). https://doi.org/10.48550/arXiv.1904.01557 arXiv:1904.01557.

[9] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. *arXiv preprint arXiv:2302.04761* (2023). https://doi.org/10.48550/arXiv.2302.04761

[10] Alessandro Stolfo, Yonatan Belinkov, and Mrinmaya Sachan. 2023. A Mechanistic Interpretation of Arithmetic Reasoning in Language Models using Causal Mediation Analysis. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (2023). https://doi.org/10.48550/arXiv.2305.15054 arXiv:2305.15054.

[11] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. *International Conference on Learning Representations* (2023). https://doi.org/10.48550/arXiv.2210.03629 arXiv:2210.03629.

[12] Fred Zhang and Neel Nanda. 2024. Towards Best Practices of Activation Patching in Language Models: Metrics and Methods. *International Conference on Learning Representations* (2024). https://doi.org/10.48550/arXiv.2309.16042 arXiv:2309.16042.