Don't Learn What You Can Compute: Arithmetic Residual Blocks for Exact Arithmetic in Transformers

Eric Gilerson
Independent Researcher
gilerson@umd.edu

Abstract

Language models typically approximate arithmetic through learned weights, even when an exact computation is available. We introduce Arithmetic Residual Blocks (ARBs), which recognize a supported terminal equation, compute its bounded integer result with a frozen residue-number core, and expose that result inside a transformer's forward pass through a small learned residual/output interface. On a frozen SmolLM2-360M model, ARB reaches 99.95% exact match on 36,000 supplied 1–3-digit expressions, versus 4.56% for the unmodified model. The interface remains near ceiling at rank one (99.89% with 305,602 learned parameters, versus 1.86M at rank 32). We next train two equally sized 20M decoders from random initialization on a mixture of ordinary text and equations: the ARB model reaches 97.7% supplied-equation accuracy versus 19.0% for the learned-arithmetic baseline, with essentially the same held-out ordinary-text NLL. Finally, a five-seed staged experiment shows that keeping arithmetic exact lets a model retain it while acquiring 40,000 unrelated key–value associations. ARB therefore provides a concrete in-forward computation primitive, evidence that it integrates with language-model training, and a controlled first measurement of reduced task interference.

Code and artifacts: https://github.com/EricGilerson/MathLLM

1 Introduction

Arithmetic is a useful stress test for language models: answers are objectively checkable, yet next-token prediction must represent carrying, borrowing, and multi-digit dependencies [Nogueira et al., 2021, Lee et al., 2024]. People learn place value, carrying, and long division in school because those concepts tell them how a calculation is structured. When accuracy matters, they write an expression down, use a calculator, or enter it in a spreadsheet. Reasoning then directs an exact, cheap computation: recognize when it applies and formulate the problem correctly.

Language models currently face two broad options: approximate arithmetic in learned weights, or emit a call to an external program [Schick et al., 2023, Parisi et al., 2022]. We study a third architectural path for bounded computations: embed the exact operation directly in the transformer's forward pass as a frozen residual block. An explicit terminal equation such as `139876*3=` is parsed, evaluated with standard tensor operations, and made available to the model before it emits the answer. The calculation therefore runs as part of a single transformer forward pass.

An *Arithmetic Residual Block* (ARB) recognizes a supported equation at the end of the token stream, computes its integer result deterministically, and injects a representation of that result at answer positions. The language model provides the surrounding context and consumes the result; the frozen module provides the bounded arithmetic value. This instantiates **Don't Learn What You Can**

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Compute: when a function has a cheap exact implementation and a safe interface, trainable capacity can be devoted to expressing the computation, incorporating its result, and generating the surrounding language. Our evaluated prompts supply the terminal equation. They measure the in-forward interface and its role during training; formulation and selection are natural next questions beyond this protocol.

The paper follows that idea from retrofit to training. We first attach ARB to a frozen 360M model and measure whether a small interface can turn its exact result into answer tokens. We then train 20M models from random initialization on ordinary text interleaved with direct and instruction-prefixed equations, asking whether the same computation primitive can participate in ordinary next-token learning. Finally, we isolate a capacity question with two small models of the same learned size: when arithmetic is provided exactly, what changes while the model learns a separate key–value memory task?

Our contributions are:

1. a guarded ARB design with a frozen residue-number arithmetic core, a learned residual/output interface, and explicit fallback outside its supported domain;
2. a 36,000-case frozen SmolLM2-360M evaluation, domain regression tests, component dependencies, and a low-rank interface sweep;
3. joint-from-scratch and five-seed staged experiments that separate supplied-equation compatibility from a narrower measured form of reduced task interference; and
4. containment and runtime measurements that report both no-trigger behavior and the eager prototype's active-path cost.

2 Related work

Arithmetic has been used to probe learned sequence models [Nogueira et al., 2021, Lee et al., 2024]. Classical circuit results place log-precision transformers in uniform TC^0 [Merrill and Sabharwal, 2023], the same broad regime containing standard arithmetic circuits. ARB takes that observation as a construction opportunity: residue arithmetic decomposes a bounded calculation into small, parallel operations with a concrete tensor representation. The contribution is therefore architectural rather than an impossibility claim—placing a verified arithmetic circuit beside learned representations and measuring how a model uses its output.

Tool-augmented models learn to call external programs [Schick et al., 2023, Parisi et al., 2022]. ARB instead has a fixed, bounded arithmetic interface inside the forward pass. It is related to neuro-symbolic and differentiable-programming approaches [Garcez and Lamb, 2019, Manhaeve et al., 2018, Evans and Grefenstette, 2018]; its learned interface uses parameter-efficient components related to adapters and LoRA [Houlsby et al., 2019, Hu et al., 2022].

3 Arithmetic Residual Blocks

3.1 Guarded contract

Given a prompt suffix $a\ \mathsf{op}\ b =$, a token-level extractor requires contiguous digit tokens, one supported operator, and an equals sign at the prompt boundary. A shared validator permits addition, non-negative subtraction ($a \geq b$), multiplication, and exact nonzero division ($b \mid a$), provided operands and result lie in the configured CRT range. Malformed strings, inexact or zero division, negative subtraction, and overflow take the ordinary base-model path; ARB injection and arithmetic LoRA remain off.

Completed equations earlier in a context do not activate answer injection. A model may become armed if it generates a valid terminal equation itself, but every arithmetic evaluation here supplies an equation; we do not evaluate equation-writing or tool selection.

3.2 Frozen residue-number computation

For pairwise coprime moduli $p_1, \ldots, p_m$, an integer n is represented by $(n \bmod p_i)_{i=1}^{m}$. The Chinese Remainder Theorem makes this representation unique on $[0, P)$, where $P = \prod_i p_i$. We use

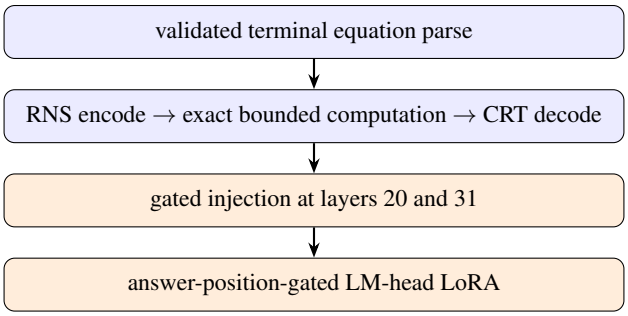

Figure 1: ARB components. Blue is frozen and orange is learned. Invalid inputs bypass all ARB-specific learned paths.

$\{7, 11, 13, 17, 19, 23\}$, for which $P = 7{,}436{,}429$, covering $999 \cdot 999 = 998{,}001$. Each residue maps to

$$\phi_p(r) = (\cos(2\pi r/p), \sin(2\pi r/p)) \,. \tag{1}$$

Viewing these coordinates as a complex number gives

$$\phi_p(a)\phi_p(b) = \phi_p((a+b) \bmod p), \qquad \overline{\phi_p(b)}\phi_p(a) = \phi_p((a-b) \bmod p). \tag{2}$$

Multiplication and modular division use frozen per-prime lookup tables after residue decoding. CRT reconstructs the bounded ordinary integer, which fixed arithmetic decomposes into decimal digits. The arithmetic operations, lookup tables, and reconstruction weights are fixed rather than fitted from arithmetic examples.

For the largest prime, adjacent residues are separated by $2\pi/23 \approx 15.7°$, far exceeding float32 perturbations. This is a practical decision margin for the finite supported domain, not a proof about every accelerator or dtype. Exact division needs special treatment because a divisor can be zero modulo individual RNS primes. We reconstruct from nonsingular lanes and re-encode missing lanes. A targeted 800-case test of prime-multiple divisors, including $84/7$ and multiples of $7 \cdot 11$ and $11 \cdot 13$, is 800/800 correct. This repair does not make inexact division supported.

3.3 Learned interface

The result encoding is injected at layer 20 and after final RMSNorm (layer 31):

$$h'_t = h_t + \sigma(g) f_\theta(r, j_t), \tag{3}$$

where r is the fixed result representation, j_t is answer-digit position, and f_θ is a small MLP. A LoRA update on the frozen language-model head is enabled by the same validated predicate. The gate starts open ($\sigma(g) \approx 0.95$): the frozen-backbone study asks whether an interface can use available information, not whether a closed gate discovers an untrained service.

The core is evaluated once for a valid suffix; the learned interface is applied only at answer positions. For ordinary text or an invalid suffix, the core, injectors, and arithmetic LoRA are bypassed.

The interface has two deliberately separate roles. The frozen core makes the bounded computation available, while the injectors and LM-head LoRA communicate its value to the residual stream and digit logits. The position signal allows the interface to distinguish, for example, hundreds from units positions in a multi-digit answer. This separation is why the frozen-model result is a retrofit-interface result, rather than evidence that the frozen transformer learned arithmetic or invocation.

4 Experimental setup

4.1 Frozen 360M interface experiment

We use the raw `HuggingFaceTB/SmolLM2-360M` foundation checkpoint (30 decoder layers, hidden size 1024). The backbone remains frozen. Two injectors and a rank-32 LM-head LoRA contain 1,859,074 trainable parameters (0.52% of the nominal model count), including 1,603,584 in head

LoRA. We train on 250,000 generated direct expressions with loss only on answer digits. The deterministic generator covers addition, non-negative subtraction, multiplication, and exact integer division, oversampling harder operand-length combinations.

For example, a record is `347*291=100977`; the prefix is supplied and loss applies only to `100977`. At evaluation, the model receives `347*291=` and greedily emits its answer. The primary test has 1,000 examples for each of nine ordered operand-length pairs and four operations (36,000 total). The raw foundation baseline receives the identical prompt without ARB or arithmetic LoRA. This is a supplied-expression protocol, not a test of natural-language problem solving or tool selection.

The architecture ablations are fresh training runs, not models obtained by masking parts of the submitted checkpoint. They retain the generated corpus, answer-only objective, frozen backbone, and evaluation suite. In contrast, the short inference-time component diagnostic below asks only whether the already-trained two-site interface depends on each component.

The rank sweep changes only output-head LoRA rank while retaining both injection sites. The placement ablation is a fresh rank-32 run with only the final post-RMSNorm injection; it should not be confused with inference-time toggling.

4.2 Mixed-corpus training from random initialization

To test ARB during language-model training itself, we train two approximately 20M-parameter decoders from random initialization: a 20,098,818-parameter ARB model and a 20,097,602-parameter baseline whose learned layers are widened to match the total size. Both optimize full next-token loss for 25,000 updates, rather than answer-only loss. Their training stream mixes 75% FineWeb-Edu ordinary-text passages, 15% canonical direct equations, and 10% equations after one of eight short instruction templates. Held-out ordinary-text documents are disjoint from training; reported results average three independent initializations.

Representative records are `35+23=58` and `Compute this arithmetic expression. Return only the integer answer: 35+23=58`. Training therefore teaches the model both ordinary text modeling and how to present a supplied computation in two surface forms. The held-out contextual evaluation changes the wording of the instruction while retaining the explicit equation, measuring whether the jointly trained interface survives a new template rather than merely memorizing one prompt.

4.3 Staged associative-memory protocol

Both variants are two-layer, width-96, four-head decoders with a 2,048-token vocabulary and context length 16. ARB has 436,882 learned parameters; the widened baseline has 437,166, or 284 more. The second task is a 40,000-entry associative memory. Each record combines two opaque key tokens from a shared 512-token bank, a separator, and one value token from another shared 512-token bank: for example, `<factkeyba><factkeyde><factmap><factvalueco>`. Given the two-key prefix and separator, the model must rank `<factvalueco>` first. Reusing the key and value vocabularies prevents memorization through one embedding row per whole fact; success requires learning the binding between a key pair and its value. The alphabetic token spellings cannot activate ARB.

Both variants first receive 8,000 arithmetic-only setup updates, reset AdamW, then receive 30,000 matched mixed updates: 384 fact instances and 128 arithmetic rehearsals per batch. Mappings, examples, dropout RNG, optimizer settings, and evaluations are fixed within each of five initialization pairs. This controlled setting measures competition between an exact computation and an unrelated learned memory.

The target-masked arithmetic pool contains 25,000 expressions with up to two-digit operands; evaluation uses 4,000 disjoint expressions. In phase B, normalized target losses are weighted 0.75 for facts and 0.25 for arithmetic. The schedule supplies 11.52M fact presentations (288 per binding) and 3.84M arithmetic rehearsals per condition. The fact-only reference receives the same fact batches and fact-loss weight, so the comparison asks about interference rather than a larger update budget.

Table 1: Frozen SmolLM2-360M exact match (%). Entries are ARB/base; every cell has 1,000 examples.

Digits	Add	Subtract	Multiply	Exact division
1×1	100.0/17.3	100.0/0.9	100.0/24.9	100.0/9.5
1×2	100.0/1.5	100.0/0.8	100.0/5.4	100.0/1.9
1×3	100.0/0.9	99.5/0.7	100.0/1.7	100.0/1.0
2×1	100.0/18.9	100.0/0.4	100.0/25.5	100.0/1.1
2×2	100.0/7.9	100.0/1.0	100.0/3.5	100.0/0.0
2×3	99.9/0.8	100.0/0.0	99.9/0.9	99.9/0.0
3×1	99.9/8.3	99.6/1.3	99.9/17.2	100.0/1.8
3×2	99.9/3.9	100.0/0.0	99.9/0.8	100.0/0.0
3×3	99.9/4.2	100.0/0.1	99.7/0.0	100.0/0.1
All	99.99/7.08	99.90/0.58	99.93/8.88	99.99/1.71

Table 2: Wrong-digit positions across the 17 failed exact-match examples. Position 1 is the most significant answer digit.

Operation	Errors	Pos. 1	Pos. 2	Pos. 3	Pos. 4	Pos. 5+
Addition	1	0	0	1	0	0
Subtraction	9	0	0	9	–	–
Multiplication	6	1	0	3	1	1
Division	1	0	0	1	–	–
Total	17	1	0	14	1	1

5 Results

5.1 Exact supplied arithmetic

ARB reaches 35,983/36,000 (99.9528%) versus 1,642/36,000 (4.5611%) for the raw foundation model (Table 1). The gap appears in every operation and operand-length pairing, including 3×3 multiplication (99.7% versus 0.0%). This establishes that the learned residual/output interface can surface a deterministic arithmetic value under the direct supplied interface.

The 17 wrong examples comprise one addition, nine subtraction, six multiplication, and one division error. Across their 18 wrong digit positions, 15 occur at the third emitted digit or later, consistent with an autoregressive interface limitation rather than an RNS arithmetic failure. A component diagnostic on 36 supplied expressions obtains 36/36 correct with all components, 0/36 with injection disabled, 1/36 with head LoRA disabled, and 3/36 with both disabled. These are dependencies of the trained output interface, not evidence of learned invocation.

Table 2 makes the error pattern explicit. It does not identify a causal mechanism, but it rules out a concentration in one operation that would be expected from an arithmetic-core failure. The raw foundation model is evaluated with the same terminal prompt $a\mathsf{op}b =$ and greedy decoding; its result is therefore a direct-interface baseline, not a comparison to an instruction-tuned chat model or a tool-using system.

The supported-domain regression contains 1,350/1,350 correct cases; the prime-multiple division slice is 800/800. These validate the advertised contract but do not extend it to inexact division, negative subtraction, or out-of-range results.

5.2 Interface dependence and injection placement

The direct result depends on the learned interface as designed. Table 3 first toggles components of the original two-site checkpoint *only at inference* on a balanced 36-case diagnostic. Removing either residual injection or head LoRA sharply reduces accuracy. These rows establish dependency of the trained two-site interface, not whether a one-site architecture could learn the task from initialization.

The final row is the trained placement ablation: a separately initialized rank-32 model with only the final post-RMSNorm injection. It reaches 35,928/36,000 (99.8000%). Its 72 errors are concentrated:

Table 3: Inference-time dependency versus trained placement ablation. The first four rows toggle one existing two-site checkpoint; the final row is a separately trained final-only model.

Configuration	Injection	Head LoRA	Exact match
Existing two-site checkpoint	on	on	36/36 (100.0%)
Both injections off	off	on	0/36 (0.0%)
Head LoRA off	on	off	1/36 (2.8%)
Both paths off	off	off	3/36 (8.3%)
Fresh final-only rank-32 model	layer 31	on	35,928/36,000 (99.8000%)

Table 4: Five paired staged-diagnostic outcomes at 30k mixed updates. The baseline and ARB within a row use the same mapping, schedule, and RNG stream.

Seed	Baseline fact top-1	ARB fact top-1	ARB minus baseline
20260830	58.4700%	61.6875%	+3.2175 pp
20260831	55.7350%	63.5725%	+7.8375 pp
20260832	56.3650%	53.0275%	-3.3375 pp
20260833	59.6250%	61.0950%	+1.4700 pp
20260834	61.2750%	61.6950%	+0.4200 pp

60 are in 3×3 multiplication, while the remaining 35,000 examples are 99.9657% correct. Final-layer injection is therefore sufficient for near-ceiling direct decoding, while the two-site configuration improves the hardest multiplication cases.

5.3 Compactness and placement

Having established that the frozen core can be read out through the interface, we next ask how much learned machinery that readout requires and where the result should enter the model. Table 5 varies only the output-head LoRA rank, holding both injection sites fixed. Rank one retains 99.8944% accuracy with 96.875% fewer output-head parameters than rank 32. This indicates that translating the fixed result representation into digit logits can be very compact in this setting; the fixed-seed sweep does not identify a minimum sufficient rank or a convergence-rate effect.

A fresh final-only rank-32 model reaches 35,928/36,000 (99.8000%); 60 of its 72 errors are 3×3 multiplications. This supports a practical role for middle-layer injection on difficult cases, but rank and placement were not crossed factorially.

5.4 Mixed-corpus training and held-out instruction templates

The preceding results retrofit a frozen 360M model: only the ARB interface is trained, and the rank and placement studies ask how compact that retrofit can be. We now change the experimental regime. Two 20M models are trained end-to-end from random initialization on the same mixture of ordinary text and supplied equations; the baseline learns arithmetic entirely in its weights, while ARB can use the frozen arithmetic core throughout training. This experiment asks whether exact in-forward computation can coexist with ordinary language modeling and generalize across new instruction wording.

After this mixed training, ARB reaches 97.7% direct, 98.0% seen-contextual, and 99.0% held-out-contextual arithmetic accuracy, compared with 19.0%, 39.0%, and 37.0% for the equally sized learned-arithmetic baseline (Table 6). The held-out ordinary-text NLL is 3.13381 for ARB and 3.13463 for baseline (paired difference -0.00082; sample SD 0.00407). Here NLL is the standard next-token prediction loss on previously unseen FineWeb-Edu text; lower values indicate better prediction of that ordinary text. The nearly identical values show that the arithmetic benefit does not come with a measured change in this language-modeling metric.

Table 7 gives the three individual runs behind the mean. The arithmetic advantage appears in every initialization. Text NLL changes direction across seeds and remains small, indicating preserved performance on the measured ordinary-text prediction task; these runs do not establish an improvement in text quality.

Table 5: Both-layer output-head LoRA rank sweep. Full interface includes 255,490 non-head learned parameters.

Rank	Head parameters	Full interface	Exact match
32	1,603,584	1,859,074	35,983/36,000 (99.9528%)
16	801,792	1,057,282	35,978/36,000 (99.9389%)
8	400,896	656,386	35,981/36,000 (99.9472%)
4	200,448	455,938	35,976/36,000 (99.9333%)
2	100,224	355,714	35,964/36,000 (99.9000%)
1	50,112	305,602	35,962/36,000 (99.8944%)

Table 6: End-to-end 20M training from random initialization: mean across three 25k-step seeds. "Text NLL" is next-token loss on held-out FineWeb-Edu passages (lower is better). Contextual evaluation supplies an equation after a textual prefix.

Model	Text NLL	Direct	Seen context	Held-out context
Baseline	3.13463	19.0%	39.0%	37.0%
ARB	3.13381	97.7%	98.0%	99.0%

5.5 Staged associative-learning diagnostic

This second training study asks a different question: when a model must retain arithmetic while learning a large unrelated memory, does an exact arithmetic resource reduce that competition? The staged protocol starts both models with arithmetic training, then adds 40,000 opaque key–value bindings while continuing arithmetic rehearsal. The synthetic bindings isolate competition between retaining arithmetic and acquiring new associations under closely controlled conditions.

At 30k mixed updates, ARB is ahead on opaque-fact recall in four of five seeds: 60.2155% versus 58.2940% mean top-1, and 1.92409 versus 2.01904 fact NLL. The final paired top-1 difference is $+1.9215$ points with sample SD 4.0864, so this endpoint estimate includes zero.

The acquisition curves are more consistent. On a shared 4,096-fact slice, ARB leads in every seed at 5k, 10k, 15k, and 20k mixed updates, with mean paired advantages of +3.44, +7.46, +5.90, and +4.18 points. Baseline arithmetic drops from 66.615% after the arithmetic setup phase to 63.290% after mixed training despite rehearsal; ARB moves from 99.945% to 100.000%. Exact arithmetic stays available while learned capacity acquires unrelated associations, producing the reduced interference measured in this diagnostic.

The endpoint variability is informative rather than hidden. The five ARB-minus-baseline top-1 differences are +3.2175, +7.8375, -3.3375, +1.4700, and +0.4200 points. The consistent signal is faster early acquisition together with a large arithmetic-retention difference; the final fact-recall estimate remains imprecise. The diagnostic identifies an optimization pattern at the model level, while parameter-level attribution and reasoning-task transfer are separate questions.

5.6 Containment and runtime boundaries

The shared syntax-and-domain gate is inactive on 256 WikiText-103 documents (21,239 scored tokens): base and ARB perplexity are both 21.96115. In 100 SWE-bench Lite patches, 115 equals candidates yield zero valid triggers and no NLL change. These measurements include the detector and routing branch; equality follows because no injection or arithmetic LoRA activates. This is bounded evidence, not an unconditional no-forgetting guarantee: quoted valid in-domain equations such as `2+2=5` activate and alter likelihood. Tokenizers without usable digit/operator tokens need a separately validated extraction mapping.

Runtime is also a boundary, not a claimed advantage. These figures are what the current eager implementation reaches after inference-only optimization, with no retraining. On an A100, 96 paired compact equations take 228.3 ms with ARB versus 203.2 ms for the raw base (+25.2 ms, +12.4%), with 96/96 answers. The residue step is 4.0 ms and detection is 0.73 ms; the rest is the custom generation loop. No-trigger prose is +4.2 ms (+2.1%). A token-matched in-process calculator relay

Table 7: Individual end-to-end 25k-step training runs. Entries are baseline / ARB; "Text NLL" is held-out FineWeb-Edu next-token loss, and contextual evaluation supplies an explicit equation after text.

Seed	Text NLL	Direct	Seen context	Held-out context
20260723	3.14353 / 3.14739	21 / 99	41 / 99	41 / 100
20260725	3.12368 / 3.12009	5 / 97	25 / 98	24 / 97
20260726	3.13667 / 3.13394	31 / 97	51 / 97	46 / 100
Mean	3.13463 / 3.13381	19.0 / 97.7	39.0 / 98.0	37.0 / 99.0

Table 8: Five-seed staged diagnostic at 30k mixed updates. Fact top-1 is recall of 40,000 trained opaque mappings; lower NLL is better.

Metric	Baseline mixed	ARB mixed	ARB minus baseline
Fact top-1	58.2940%	60.2155%	+1.9215 pp (SD 4.0864)
Fact NLL	2.01904	1.92409	-0.09495
Arithmetic exact	63.290%	100.000%	+36.710 pp

measures 206.0 ms and remains faster on the triggered path. It receives an already available equation and omits call generation, parsing, process or network boundaries, and result interpretation.

6 Discussion and limitations

ARB demonstrates a useful architectural division of labor. The residue-number core handles a verified, bounded calculation; learned layers decide how that value should affect the residual stream and output distribution. Arithmetic is especially favorable because its residue decomposition yields small parallel subproblems. Extending this design to another function requires an equally explicit representation, contract, and validation procedure; broad programs, retrieval, and external state remain natural work for general interpreters and tools.

The experiments establish three complementary facts. The frozen study shows that a small learned interface can reliably expose a deterministic result through an existing language model. The mixed-from-scratch study shows that the same mechanism fits naturally into training on ordinary text and equations. The staged study shows a concrete optimization consequence: when arithmetic stays exact during memory acquisition, the model retains it while repeatedly learning opaque bindings faster early in training. Together these results motivate deterministic computation as a native model resource, rather than only an external service.

The current system targets supplied, single-step integer equations through a tokenizer-dependent contract. Compound-expression parsing, natural-language equation formation, long-horizon use of results, retrieval, external state, and open-ended programs remain outside its scope. The staged diagnostic is mechanistic rather than a model of world knowledge, and its final endpoint is variable across seeds. Scaling this result requires predeclared larger-model experiments with matched compute and separate tests of equation formation and downstream use.

There are two practical consequences of the contract. First, validity is syntactic and domain-based rather than semantic: a quoted but valid equation can activate even when a user intends to discuss an incorrect calculation. Second, unsupported expressions do not receive a verified answer; they follow the frozen base model. These are reasons to retain explicit validation and to evaluate a deployment interface on its intended tokenizer and data distribution.

7 Future work

The immediate next step is to train larger models on distributions where forming an equation, using its result later in a response, and deciding when the contract applies are explicit objectives. That setting would test whether a model can learn the interface policy as well as the interface decoder. A second direction is richer arithmetic: compound expressions, signed and rational representations, and longer-horizon composition require a parser and contract that remain as inspectable as the one

used here. A fused kernel implementation of the residue step and its injection is the natural way to remove the eager-PyTorch routing measured above. The present figures come from optimizing that path at inference time; building the same guarded computation into the architecture from training is the setting in which this overhead can approach the cost of the residue step itself.

8 Conclusion

Arithmetic Residual Blocks turn one instance of "Don't Learn What You Can Compute" into a tested transformer component. The frozen RNS core calculates supported expressions exactly; the learned interface places that value where a language model can use it. In a frozen 360M model, the interface produces 99.95% exact match on a 36,000-case direct suite and stays near that level with a rank-one output adapter. More importantly, training ARB-equipped models from random initialization on a mixture of ordinary text and equations raises supplied-equation accuracy from 19.0% to 97.7% without changing held-out ordinary-text loss, and the staged experiment links exact arithmetic to reduced interference during unrelated associative learning. These results make a practical case for treating verified deterministic computations as first-class architectural resources, with the interface contract defining both their usefulness and their limits.

Reproducibility and ethics statement

The public repository includes generators, configurations, evaluation scripts, and experiment documentation. Experiments use public text and synthetic arithmetic/association data, with no human subjects or personal data. Exact computation can improve reliability for supported inputs, but unsupported inputs follow the base path and valid quoted equations may still activate; ARB is not a substitute for verification in high-stakes settings.

References

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

Artur d'Avila Garcez and Luis C Lamb. Neurosymbolic ai: The 3rd wave. *Artificial Intelligence Review*, 2019.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, et al. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, 2019.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Namhoon Lee, Kartik Sreenivasan, Jason D Lee, Kwangjun Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers. In *International Conference on Learning Representations*, 2024.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, 2018.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks. *arXiv preprint arXiv:2102.13019*, 2021.

Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, et al. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

A Arithmetic core and interface details

The extractor operates on token identities, not decoded text. Supported forms include `35+23=`, `999+1=`, `123*45=`, and `84/7=`; whitespace is accepted only when represented by separate valid tokens. The final equals sign must end the prompt. `x = y`, `1+2==3`, `= hello`, and an equation completed earlier in context are not active answer sites. A valid terminal equation arms a bounded multi-digit answer representation, but model-generated equation formation is not evaluated.

For digits $d = (d_0, \ldots, d_{K-1})$ in least-significant-first order, frozen coefficients $c_i = (10^0 \bmod p_i, \ldots, 10^{K-1} \bmod p_i)$ give $r_i = (c_i \cdot d) \bmod p_i$. Addition and subtraction use the circle identities in Section 3. Multiplication and division decode residues against frozen codebooks, apply fixed tables, and re-encode. If a divisor is zero in a residue lane, reconstruction from nonsingular lanes followed by re-encoding repairs the lane only for otherwise exact in-domain division.

B Rank-sweep cell summary

No rank exhibits a sharp collapse. Across the nine operand-length cells, exact-match ranges are 99.9–100.0, 99.5–100.0, 99.7–100.0, and 99.9–100.0 for rank 32 addition, subtraction, multiplication, and division; rank one ranges are 99.2–100.0, 99.3–100.0, 99.8–100.0, and 100.0–100.0. These fixed-seed results use different historical early-stopping settings and should not be interpreted as rank-versus-convergence comparisons.

C Containment and timing protocol

Containment compares frozen-base and ARB scoring of identical token sequences. WikiText documents are capped at 128 tokens and aggregate NLL is token-weighted; SWE-bench patches are similarly capped, with each equals candidate audited against the contract. Timing uses paired prompts, warmup iterations, and a common generation budget. The calculator relay receives an already available equation and omits call generation, parsing, network/process boundaries, reconstruction, and interpretation.