Natural-Language Invocation of a Deterministic Arithmetic Unit Inside a 0.5B-Parameter Language Model

A Frozen Same-Prompt Comparison with an Untouched Base and Parameter-Matched Learned Control

Josiah Wilson
Independent Researcher

July 26, 2026

Abstract

Can a weak small language model gain reliable arithmetic ability from a deterministic process installed inside its neural forward path, rather than from post-inference answer replacement? We augmented Qwen2.5-0.5B-Instruct (494,032,768 parameters) after its twenty-fourth transformer block with a learned request-level semantic router, a frozen zero-parameter typed ripple-carry cell, and a learned symbol-to-residual decoder. The base model remained frozen. An ordinary learned residual adapter at the same location had the same router, training prompts, and exactly the same 24,225 learned parameters. Following pilot-only architecture selection, we froze checkpoints, hashes, prompt families, seeds, sample sizes, metrics, and six simultaneous success criteria before confirmatory inference.

On 400 previously unused natural-language addition prompts, the untouched base was mathematically correct on 68/400 (17.0%), the matched learned control on 21/400 (5.25%), and the learned-router deterministic model on 360/400 (90.0%). The paired gains were 73.0 percentage points over base ($p = 2.51 \times 10^{-88}$, exact McNemar) and 84.75 points over control ($p = 1.79 \times 10^{-102}$). With routing forced on, the internal model answered 400/400 exactly. Every one of 349 automatically activated arithmetic executions was exact; all end-to-end failures were routing misses. On 160 unseen non-addition prompts, the router produced zero false activations and the internal outputs were token-identical to base in 160/160 cases. With the unit forced off, positive-prompt outputs were token-identical to base in 400/400 cases.

Five of six preregistered criteria passed. The exception was positive routing coverage: 349/400 (87.25%), below the prespecified 90% threshold. Thus the primary arithmetic capability hypothesis passed decisively, but the compound confirmatory verdict is not a full pass. The result is evidence that exact typed computation can create a large, causally localized skill increase inside a pretrained transformer. It is not evidence for general mathematical understanding, a learned number extractor, or arithmetic in one conventional scalar neuron. It also does not originate the broad integrated-calculator concept: Dietz and Klakow's Integrated Gated Calculator (IGC) previously coupled a frozen Llama 3.1 8B model to a learned internal gate, learned input/output mappings, and a non-differentiable calculator [10]. The present contribution is a smaller-model controlled replication and extension emphasizing a 24,225-parameter interface, precommitted same-prompt evaluation, an exactly parameter-matched learned control, unseen natural-language routing, and causal on/off tests.

Technical summary

Primary result. On identical, never-before-used prompts, the internal deterministic condition reached 90.0% mathematical correctness, versus 17.0% for the untouched base and 5.25% for an equal-parameter

learned adapter. Oracle routing yielded 100%. The deterministic payload did not make an arithmetic error when active.

Formal verdict. The central capability comparisons passed by large margins, but the full six-part preregistered rule is **mixed: five pass, one misses**. Automatic positive activation was 87.25%, 2.75 points below its 90% threshold.

Preregistered criterion	Observed	Threshold	Result
Internal gain over untouched base	+73.0 pp	$\geq$ 30 pp	**PASS**
Internal gain over matched control	+84.75 pp	$\geq$ 30 pp	**PASS**
Positive-prompt route activation	87.25%	$\geq$ 90%	**MISS**
Negative-prompt false activation	0/160	$\leq$ 2%	**PASS**
Oracle-route mathematical accuracy	400/400	$\geq$ 99%	**PASS**
Forced-off token identity to base	400/400	100%	**PASS**

Claim boundary. A fixed input boundary extracted exactly two contiguous nonnegative decimal strings and converted their characters to typed digits. A learned late-layer router decided whether the request asked for addition. Consequently, this study tests semantic operation routing and deterministic execution, not learned number extraction. The exact cell is a small structured module inside a wrapped transformer layer, not one scalar neuron and not an external calculator call. This is not the first integrated calculator architecture; the incremental contribution is the controlled small-model evidence and its unusually small learned interface.

1 Motivation and related work

Neural language models are valuable partly because they learn flexible conditional distributions over text. Exact arithmetic poses a different requirement: for valid operands and an unambiguous operation, there is one correct transition sequence and one correct result. Conventional neural networks can learn arithmetic correlations yet extrapolate poorly outside their training range; architectural arithmetic units and task-structured position schemes have been proposed to improve systematic generalization [1, 2, 3].

Another line of research compiles algorithms into transformer weights or connects transformers with neural algorithmic reasoners [4, 5]. Tracr-Injection distills RASP algorithms into pretrained models and reports interpretable residual subspaces and improved out-of-distribution behavior [9]. Algorithmic language models with compiled libraries similarly argue that common algorithms need not be relearned from scratch [6]. Parameter-efficient methods such as LoRA demonstrate that useful adaptations can be introduced while freezing pretrained weights [8].

The closest prior work is the Integrated Gated Calculator (IGC) of Dietz and Klakow [10]. IGC freezes Llama 3.1 8B and trains an approximately 17-million-parameter module comprising a learned input mapping, categorical number and operator selection, a non-differentiable GPU calculator, and a learned output mapping. It executes inside a single forward iteration at an anchor token rather than through an external scratchpad, and reports 98–99% on BIG-Bench Arithmetic across addition, subtraction, multiplication, and division. It is therefore direct prior art for the broad claim that an internally gated exact calculator can improve a frozen language model.

Table 1 separates that established concept from the incremental question tested here. The two studies are not head-to-head: models, arithmetic tasks, prompt distributions, parameter budgets, and evaluation protocols differ, so their accuracy percentages should not be compared as if they came from one benchmark.

Dimension	IGC [10]	Present study
Frozen base	Llama 3.1 8B	Qwen2.5 0.5B
Learned interface	Approximately 17M parameters	24,225 parameters
Exact payload	Non-differentiable GPU calculator; four operations	Zero-parameter typed ripple-carry addition
Input interface	Learned categorical number and operator mapping from tokens	Fixed two-decimal boundary plus learned semantic addition router
Language scope	Recognizable input templates; complex word problems left as future work	Frozen unseen simple and word-problem families plus 16 negative families
Main controls	Learned Llama baseline and component ablations	Untouched base, exact parameter-matched residual control, oracle-on, and forced-off identity
Confirmation style	Multiple training runs	Single frozen checkpoint, precommitted protocol, and per-prompt raw audit

Table 1: Non-head-to-head positioning against the closest prior integrated calculator. IGC is broader in operations and learns operand extraction; this study uses a much smaller base and learned interface and concentrates on matched causal controls and held-out language routing.

The motivating proposal in this project is narrower and more literal: place a locked deterministic process inside the language model, then learn only the interfaces that decide when to use it and translate between discrete state and the residual stream. Earlier phases established (i) an output-adjacent exact arithmetic path and (ii) a typed ripple-carry unit after block 6 whose state survived eighteen downstream blocks. Those studies used a registered command grammar. The present phase asks the missing end-to-end question:

> On ordinary varied English prompts, can the frozen model itself recognize an addition request, invoke an internal exact circuit, and substantially outperform both its untouched behavior and an equally sized learned adapter?

2 Development history and separation from confirmation

Architecture selection was confined to pilots. The complete chronological record is in `PHASE4_LAB_NOTEBOOK.md`.

2.1 Pilot v1: exact execution, poor early routing

The first design routed from the final request token after block 6 using a linear 896-to-1 classifier. On 400 held-out routing prompts it reached only 63.0% accuracy, with 38.0% positive activation and 12.0% false activation. In contrast, forcing the deterministic route on yielded 100% mathematical and exact-format correctness across all five pilot arithmetic splits. The base ranged from 58.3% on the easiest split to 0% on nine- to twelve-digit addition and 8.3% on word problems. This isolated semantic routing, rather than arithmetic execution, as the bottleneck.

2.2 Development-only architecture sweep

Using only wording already consumed by pilot v1, we swept request representations after blocks 6, 12, 18, and 24; final-token versus mean-plus-final pooling; and linear, width-16, and width-64 classifiers. Depth was decisive. The best block-6 result was 66.8%, whereas a compact block-24 width-16 classifier reached 96.0% accuracy with 98.5% positive activation. The final block was selected because semantic intent was much more separable there.

2.3 Pilot v2 and freeze

The final candidate router was a 896-to-16-to-1 SiLU network. It trained on 2,400 positive and 2,400 negative development prompts spanning 34 positive and 32 negative families. Threshold 0.76 was selected on a disjoint 1,600-example development set by maximizing positive activation subject to false activation no greater than 1%. It achieved 99.0% development accuracy, 99.0% positive activation, and 1.0% false activation.

The complete pilot-v2 internal model then scored 95%, 85%, 100%, and 100% on four 20-example development splits, compared with 30%, 5%, 0%, and 0% for base. Oracle routing was 100% throughout. It preserved 40/40 negative outputs with no false activations. Checkpoints, SHA-256 values, confirmatory wording families, seeds, sample sizes, outcomes, and success thresholds were then frozen in Git commit `b1d6187` and the runner in `243612a`, before any confirmatory inference.

3 Architecture

3.1 Frozen pretrained model

The base was `Qwen/Qwen2.5-0.5B-Instruct`, exact revision
`7ae557604adf67be50417f59c2c2f167def9a775`
[7]. It contains 494,032,768 parameters, 24 decoder blocks, and hidden width 896. Every pretrained parameter remained frozen.

Ordinary tokenized request → unchanged blocks 1–24
→ **learned, latched semantic router**
→ fixed decimal-span boundary → **frozen typed ripple-carry cell**
→ learned symbol-to-residual decoder → original final RMS norm and language-model head

The internal intervention executes in the wrapped twenty-fourth block's forward method, before the original final normalization and output head. It does not edit logits, replace a decoded answer, or call a process after inference. Because it is inserted after the last attention/MLP block, however, this phase demonstrates an internal output-facing module, not deterministic state surviving multiple later transformer blocks; that stronger placement property was tested in the preceding phase.

3.2 Fixed candidate-number boundary

Every prompt contains exactly two ordinary contiguous decimal strings, such as `12345`. A fixed regular-expression boundary `(?<![0-9])[0-9]+(?![0-9])` locates the two character spans and maps each character to a typed digit. The same boundary runs on addition, subtraction, quotation, comparison, concatenation, and other negative prompts. It supplies candidate values but does not decide what operation was requested.

This is intentionally weaker than end-to-end learned parsing. The experiment can support a claim about learned semantic selection among requests sharing the same numeric structure, but cannot support a claim that the model learned to extract arbitrary numbers from language.

3.3 Learned request-level router

Let $h_{\text{req}} \in \mathbb{R}^{896}$ be the final request-token residual after block 24. The router computes

$$p_{\text{add}} = \sigma\left(w_2^{\top}\,\mathrm{SiLU}(W_1 h_{\text{req}} + b_1) + b_2\right),$$

where $W_1 \in \mathbb{R}^{16\times 896}$. The route activates when $p_{\text{add}} \geq 0.76$. Its 14,369 parameters are the only component that chooses whether addition was requested. The decision is made once on the complete initial request and latched throughout generation; answer tokens cannot change it.

3.4 Frozen typed arithmetic cell

For decimal digits a_j, b_j and incoming carry c_j, the cell applies immutable lookup buffers:

$$s_j = (a_j + b_j + c_j) \bmod 10, \tag{1}$$
$$c_{j+1} = \lfloor (a_j + b_j + c_j)/10 \rfloor . \tag{2}$$

It iterates from least to most significant place using tensor operations and emits a typed plan over eleven symbols: digits zero through nine and end-of-answer. Unequal operand lengths and per-example active widths are masked. The cell has zero trainable parameters.

3.5 Learned symbol decoder

An eleven-entry codebook $E \in \mathbb{R}^{11\times 896}$ maps each planned symbol to a normalized residual update of fixed strength 64:

$$\Delta h_t = 64 \frac{E[z_t]}{\|E[z_t]\|_2}.$$

The codebook has 9,856 trainable parameters. It was initialized from the language-model head's digit and end-token directions and trained for 180 teacher-forced steps on positive development prompts. Free-running greedy generation was used for every reported evaluation.

3.6 Exactly matched learned control

The non-deterministic control shared the exact router state, threshold, insertion point, active answer positions, base model, and training prompts. Its payload was a rank-five SiLU residual adapter:

$$\Delta h = W_{\text{up}} \,\text{SiLU}(W_{\text{down}} h) + b_{\text{up}}.$$

With no down-projection bias, it contains $896\cdot 5 + 5\cdot 896 + 896 = 9{,}856$ parameters, exactly matching the symbol codebook. Both conditions therefore contain 14,369 router parameters plus 9,856 payload parameters, or 24,225 total (approximately 0.0049% of the base). The adapter received 600 training steps, more than three times the codebook's 180, providing a conservative learning opportunity.

4 Frozen confirmatory protocol

4.1 Evaluation data

The confirmatory family constants were not imported by any pilot script. After protocol freeze they generated four positive splits of 100 prompts each:

1. unseen simple wording, one- to four-digit operands;
2. unseen simple wording, five- to eight-digit operands;
3. unseen simple wording, nine- to twelve-digit operands;

4. unseen word-problem wording, five- to eight-digit operands.

Eight simple addition families and six word-problem families were entirely new. Seeds 9501–9504 fixed operands, family assignments, and order. A separate seed-9505 safety set contained 160 prompts from 16 unseen non-addition families: subtraction, signed difference, multiplication, remainder, comparison, concatenation, repetition, quotation, refusal, explanation, parity, syntax, identifier handling, and averaging.

4.2 Conditions and decoding

Every positive prompt was evaluated under five conditions:

1. untouched base;
2. parameter-matched learned adapter;
3. deterministic unit with learned routing;
4. deterministic unit with oracle-on routing;
5. deterministic unit forced off.

All used the same chat template, greedy decoding, and 24-token generation budget. Negative prompts used 20 tokens and compared base against learned routing. No example was filtered after generation.

4.3 Outcomes and statistics

The primary outcome was pooled mathematical correctness on all 400 positives: the final decimal integer in the response, with commas removed, had to equal the exact sum. Exact-format correctness separately required the entire stripped response to equal the expected integer. This credits a base answer such as "The result is 42" mathematically while still recording instruction noncompliance.

Because conditions saw identical prompts, primary comparisons used exact McNemar tests on paired correctness. Wilson 95% intervals describe single-condition accuracy and route rates. Results are counts and deterministic derived summaries; no sampling-based generation variance is present.

5 Confirmatory results

5.1 End-to-end mathematical accuracy

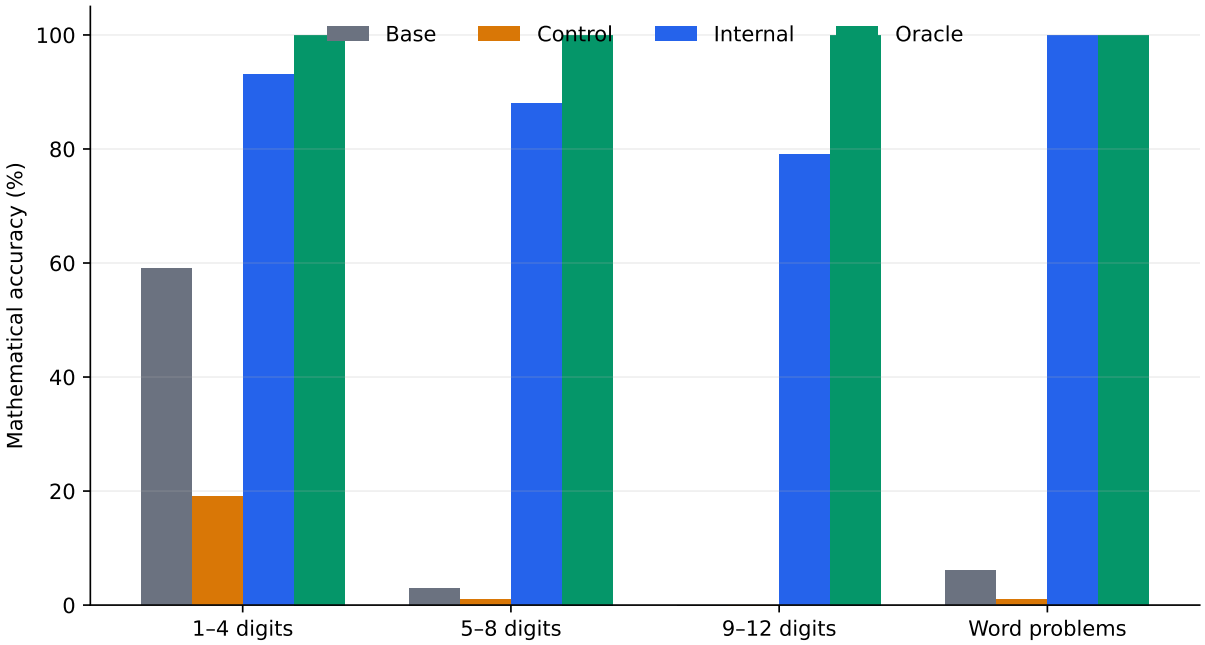

Figure 1: Mathematical correctness on identical confirmatory prompts. The learned-router deterministic condition loses points only when routing remains off; oracle activation is exact on every split.

Condition	1–4 digits	5–8 digits	9–12 digits	Word problems
Untouched base	59/100	3/100	0/100	6/100
Matched learned control	19/100	1/100	0/100	1/100
Deterministic, learned route	93/100	88/100	79/100	100/100
Deterministic, oracle route	100/100	100/100	100/100	100/100
Deterministic, forced off	59/100	3/100	0/100	6/100

Table 2: Confirmatory mathematical correctness by split.

Pooled results are shown in Figure 2 and Table 3. The internal condition improved from the base's 17.0% to 90.0%, a 73.0-point gain. It exceeded the equal-parameter learned control by 84.75 points.

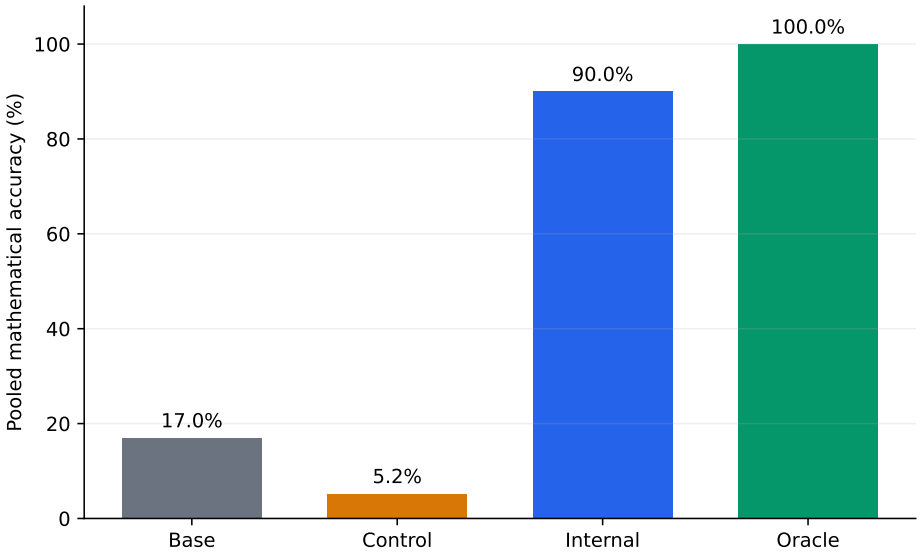

Figure 2: Pooled mathematical correctness across 400 positive prompts.

Condition	Correct	Accuracy	Wilson 95% interval
Untouched base	68/400	17.00%	[13.64, 20.99]
Matched learned control	21/400	5.25%	[3.46, 7.89]
Deterministic, learned route	360/400	90.00%	[86.67, 92.57]
Deterministic, oracle route	400/400	100.00%	[99.05, 100.00]
Deterministic, forced off	68/400	17.00%	[13.64, 20.99]

Table 3: Pooled confirmatory mathematical correctness.

5.2 Paired comparisons

Against base, 68 prompts were answered correctly by both conditions, 292 only by the internal model, zero only by base, and 40 by neither. The exact McNemar test gives $p = 2.513 \times 10^{-88}$.

Against the learned control, 21 were correct under both, 339 only under the internal model, zero only under control, and 40 under neither ($p = 1.786 \times 10^{-102}$). The absence of comparator-only successes follows from the design: when the router remained off, the internal path reproduced base behavior; when on, its arithmetic was exact.

5.3 Execution versus routing

The router activated on 349/400 positive prompts (87.25%, Wilson 95% [83.62, 90.17]). Every activated response was both mathematically and format-exact: active arithmetic errors were 0/349. Of 51 routing misses, the unchanged base path happened to answer 11 correctly and failed on 40. Consequently:

$$\underbrace{349}_{\text{active and exact}} + \underbrace{11}_{\text{inactive but base correct}} = \underbrace{360}_{\text{end-to-end correct}}.$$

Oracle routing produced 400/400 mathematical and format-exact responses. Thus no confirmatory evidence indicates a carry, digit-length, word-problem, or symbol-decoding failure. All observed end-to-end errors arose upstream of the exact state transition.

Routing generalized unevenly across simple wording. One family—"Compute the additive result for the pair {a} and {b}"—failed to activate on every five-to-eight-digit and nine-to-twelve-digit instance. Another family using "sum operation" activated on roughly 60–69% depending on length. In contrast, all six word-problem families activated on every example. This family clustering explains why the larger development-set routing estimate did not fully transfer to the frozen wording set.

5.4 Exact-format behavior

Exact-format correctness was 7/400 (1.75%) for base, 10/400 (2.50%) for the learned control, 349/400 (87.25%) for learned-routing internal, and 400/400 for oracle internal. Mathematical scoring therefore did not create the main gain by penalizing base prose: even after crediting any final correct integer, base remained at 17.0%.

5.5 Safety and causal localization

On the 160 unseen non-addition prompts, route activation was 0/160 (observed 0%; Wilson 95% [0, 2.34]) and internal output token sequences were identical to base in 160/160 cases. This includes requests that contained addition symbols or words but explicitly asked not to calculate.

With the installed unit forced off on all positive prompts, outputs matched the untouched base token-for-token in 400/400 cases and reproduced its 17.0% accuracy. With routing forced on, accuracy became 100%. These two interventions localize the gain to the active deterministic path rather than to model loading, wrapper placement, or incidental parameter changes.

5.6 Preregistered verdict

The two capability criteria passed by more than twice their required margins. Negative routing, oracle accuracy, and forced-off identity also passed. Positive activation missed: 87.25% versus the required 90%.

Accordingly, the correct formal statement is:

> **The deterministic arithmetic capability hypothesis was confirmed, but the six-criterion compound experiment did not fully pass because automatic positive routing coverage fell 2.75 percentage points below its preregistered threshold.**

The Wilson interval includes 90%, but the protocol defined success from the observed point estimate, not interval overlap. It would be inappropriate to relax that rule after observing the data.

6 Interpretation

6.1 What the experiment demonstrates

Relative to IGC, this experiment should be read as an independent, tightly controlled small-model replication and extension of the integrated-calculator idea, not as its invention. Its distinctive evidence is that only 24,225 learned interface parameters, coupled to a typed exact cell, produced a 73-point gain over the untouched 0.5B model while an exactly parameter-matched unstructured payload did not. The precommitted same-prompt design and forced on/off interventions localize that gain.

First, the chosen 0.5B model was genuinely weak on the task. It scored 3% on five-to-eight-digit simple prompts, 0% on nine-to-twelve-digit prompts, and 6% on word problems. The evaluation therefore had substantial headroom and did not merely polish an already solved skill.

Second, additional learned parameters alone did not explain the gain. The matched adapter had the same 24,225 trainable parameters, router weights, activation decisions, insertion point, and prompts,

yet reached only 5.25%. On long addition it remained at 0%. The deterministic condition's exact state transition supplied an inductive bias the ordinary low-rank adapter did not recover.

Third, the gain was not post-inference correction. The typed cell and symbol-to-residual decoder ran inside the model's block-24 wrapper before the original normalization and language-model head. Free-running generation consumed those logits normally. Forced-off identity and oracle-on accuracy show that this path causally controlled the outcome.

Fourth, semantic variation did not break the arithmetic payload. The oracle condition was 400/400, including every unseen word problem. Once activated, the learned-route condition was also exact in 349/349 cases. The unresolved problem is recall of addition intent, not reliable addition.

6.2 What the experiment does not demonstrate

The result does not establish general mathematical reasoning. It covers nonnegative decimal addition of exactly two operands up to twelve digits. It does not include signed numbers, decimals, fractions, multiple operations, algebra, proof, or ambiguous requests.

It does not establish a calculator "in one neuron." The exact computation is a structured module containing typed buffers and an iterative carry rule. The router and symbol codebook together contain 24,225 learned parameters. "Neural firmware" is a useful architectural analogy, but the implementation is closer to a small registered coprocessor than a repurposed scalar activation.

It also does not establish end-to-end token parsing. A fixed boundary converted two character strings to digit tensors. This is more integrated than an external answer tool because operation choice and output generation occur in the model forward path, but less integrated than learning both operand extraction and execution from residuals.

Finally, it does not establish priority for the general integrated-calculator architecture. IGC demonstrated that design family before this confirmatory study, on a larger frozen model and a broader set of operations. The claim supported here is narrower: a small learned interface plus a deterministic typed cell can create a large, causally isolated improvement in this small-model, matched-control setting.

7 Limitations and threats to validity

- **Single model and checkpoint.** Confirmation used one Qwen revision and one trained interface state. Multiple router/interface seeds and other 0.5–1.5B model families are needed.
- **Synthetic English families.** Prompt families were hand-written and randomly instantiated. Real dialogue includes more ambiguity, additional numbers, punctuation, signs, units, and context.
- **Fixed two-number parser.** The boundary is deterministic and privileged. It cannot handle written number words, more than two decimal spans, or context-dependent operand selection.
- **Late insertion.** The circuit sits after the final transformer block. It is inside inference but undergoes only final normalization and the head afterward. Earlier-layer survival was established separately under a restricted command, not under natural-language routing here.
- **Router undercoverage.** The sole preregistered miss was not random arithmetic noise but family-clustered semantic false negatives. Broader data, calibrated uncertainty, or a more compositional operation representation may improve recall without sacrificing safety, but that is a new experiment.
- **Control scope.** One rank-five adapter is a principled parameter-matched control, not proof that every possible learned architecture with 24,225 parameters must fail. It actually underperformed

base on easy prompts, showing that this particular training intervention can disturb existing arithmetic behavior.

- **No tool baseline.** An external exact calculator would likely solve the arithmetic more simply. The research question here concerns internal deterministic structure, not deployment efficiency.
- **No direct IGC replication.** This study did not reimplement IGC on Qwen, run this architecture on IGC's task distribution, or equalize parameter budgets. Its 90% result therefore cannot be interpreted as better or worse than IGC's reported 98–99%. IGC is stronger in learned operand extraction and operation breadth; this study's evidence is strongest in tiny-interface size, exact parameter matching, precommitment, and causal preservation controls.
- **Observed safety is bounded.** Zero false activations on 160 negatives has a Wilson upper bound of 2.34%; it is evidence, not a universal guarantee.

8 Next experiments

The highest-priority next study is a direct positioning experiment against an IGC-style baseline. On identical prompts and the same 0.5B model, it should compare the untouched base, conventional parameter-efficient adaptation, the present typed cell, and an IGC-style learned input/calculator/output module. At least three independently trained interface seeds and fixed parameter budgets should be used. A complementary evaluation should either run both architectures on the same expanded four-operation corpus or run the present system on BIG-Bench Arithmetic. This would establish whether the tiny typed interface offers a reproducible efficiency or control advantage rather than merely rediscovering the established integrated-calculator effect.

The central architectural step is then to remove the fixed number boundary. A learned residual-to-register encoder could identify operand spans and digit values, with separate held-out tests for written numbers, signs, extra distractor numbers, and reference resolution. Because IGC already learns categorical number and operator mappings, this step is necessary for parity with the closest prior work. The operation registry could then expand from addition to subtraction, multiplication, comparison, and bounded code primitives.

Router improvement remains useful but should be evaluated inside that comparison rather than as an isolated victory. A larger adversarial wording corpus, calibrated abstention, and a second semantic verifier could target the observed family-clustered false negatives while preserving low false activation. Replication across a 0.5–1.5B Llama-class model would test architectural portability.

For code, a credible deterministic payload would not be "one code neuron." It would be a typed interpreter kernel or verified transition library with learned routing and serialization. Board games provide an intermediate domain: legal move generation and state transition are deterministic, while strategy and explanation remain generative. The same experimental logic applies: compare an untouched small model, a parameter-matched learned interface, and a frozen state-transition module on identical unseen language and state formats.

9 Reproducibility

9.1 Frozen artifacts and environment

Confirmatory inference began from source commit `243612aa1ce7f91bb674a11085b24375721b239f`. Frozen checkpoint SHA-256 values were:

- router: `201262b5cf21259977dc8a31e3faa1aa77892f7cbae121ea015f4e69d95f8e66`;

- internal unit: `8079e0c5d723405881c39e47773fb895617d5c4da99b3168996e2861aba9a739`;
- learned control: `7d43f58126fc60bfb68ee2caa479818ce2b13b9b1c8d9f2b4b7adccb9840da94`.

The canonical rendered-data SHA-256 is:

`b6d9de587db437f49e4106c6b10b643976e37f6bcffe1cf8adae7994d2bf7f98`

The run used macOS 26.5.2 on Apple arm64, Python 3.12.12, PyTorch 2.13.0, and MPS. Confirmatory inference took 1,096.43 seconds. All 22 tests and Ruff passed before the frozen runner was launched.

9.2 Commands

```
git checkout 243612aa1ce7f91bb674a11085b24375721b239f
uv sync --extra dev
uv run pytest -q
uv run ruff check .
uv run python scripts/run_phase4_confirmation.py
uv run python scripts/analyze_phase4_confirmation.py
```

The first command requires the three local ignored checkpoints whose hashes are listed above. Pilot recreation commands are:

```
uv run python scripts/run_phase4_router_sweep.py
uv run python scripts/run_phase4_router_pilot_v2.py
uv run python scripts/run_phase4_semantic_pilot_v2.py
```

9.3 Artifact inventory

Path	Purpose
`PHASE4_CONFIRMATORY_PROTOCOL.md`	Frozen research question, hashes, seeds, outcomes, and criteria.
`PHASE4_LAB_NOTEBOOK.md`	Chronological decisions, defects, pilots, architecture sweep, freeze, run, and final audit.
`phase4_results/confirmation_raw.json`	Every rendered prompt, expected answer, response, generated token ID, route probability, route decision, latency, score, environment field, and artifact hash.
`phase4_results/confirmation_analysis.json`	Pooled estimates, Wilson intervals, paired tests, routing decomposition, causal controls, and formal criterion verdict.
`phase4_results/confirmation_summary.csv`	Condition-by-split and pooled accuracy table.
`phase4_results/confirmation_by_family.csv`	Wording-family accuracy and route activation rates.
`phase4_results/artifact_manifest.sha256.json`	SHA-256 and byte count for the protocol, reports, raw/derived results, paper, and local checkpoints.
`scripts/run_phase4_confirmation.py`	Frozen inference runner with checkpoint hash enforcement.
`scripts/analyze_phase4_confirmation.py`	Deterministic statistics and figure generator.

Path	Purpose
`src/neural_firmware/semantic_*.py`	Data generation, architecture, training, generation, and scoring implementation.
`tests/test_semantic_*.py`	Family separation, parsing, scoring, parameter-match, exact-cell, identity, and batch-capacity tests.
`phase4_artifacts/`	Local ignored router, internal-unit, and learned control checkpoints.

10 Conclusion

This experiment produced a large and precisely localized capability increase in a weak small language model. On 400 identical natural-language prompts, the untouched model achieved 17.0% mathematical correctness, an equal-sized learned adapter achieved 5.25%, and the deterministic internal architecture achieved 90.0%. Oracle routing was perfect, every automatic activation was exact, the forced-off model exactly reproduced base, and all 160 negative outputs were preserved.

Those observations independently support the neural-firmware hypothesis: a known deterministic process can be placed inside a frozen generative model, connected through small learned interfaces, and create a dramatic skill improvement that an equally sized unstructured learned adapter does not reproduce. They do not establish the first integrated calculator in an LLM; IGC is direct prior art. The contribution is instead a small-model, tiny-interface replication and extension with a frozen same-prompt protocol, an exact parameter-matched control, and causal preservation tests.

The study also exposes the next bottleneck. The exact circuit did not fail; the semantic router missed 51 addition prompts and achieved 87.25% positive coverage, below the preregistered 90% criterion. The honest outcome is therefore stronger and more useful than either "total success" or "failure": deterministic arithmetic integration worked decisively, while robust natural-language invocation remains unfinished.

Acknowledgment

The architectural hypothesis and research direction were proposed by Josiah Wilson. Experimental implementation, local execution, statistical analysis, and drafting were conducted collaboratively with OpenAI Codex. All confirmatory prompts, outputs, code, hashes, and criterion decisions are archived for audit.

References

[1] A. Trask, F. Hill, S. Reed, J. Rae, C. Dyer, and P. Blunsom. Neural Arithmetic Logic Units. *NeurIPS*, 2018. https://arxiv.org/abs/1808.00508.

[2] S. Jelassi, S. d'Ascoli, C. Domingo-Enrich, Y. Wu, Y. Li, and F. Charton. Length Generalization in Arithmetic Transformers. 2023. https://arxiv.org/abs/2306.15400.

[3] H. Cho, J. Cha, P. Awasthi, S. Bhojanapalli, A. Gupta, and C. Yun. Position Coupling: Improving Length Generalization of Arithmetic Transformers Using Task Structure. 2024. https://arxiv.org/abs/2405.20671.

[4] D. Lindner, J. Kramár, S. Farquhar, M. Rahtz, T. McGrath, and V. Mikulik. Tracr: Compiled Transformers as a Laboratory for Interpretability. *NeurIPS*, 2023. https://arxiv.org/abs/2301.05062.

[5] W. Bounsi, B. Ibarz, A. Dudzik, J. Hamrick, L. Markeeva, A. Vitvitskyi, R. Pascanu, and P. Veličković. Transformers Meet Neural Algorithmic Reasoners. 2024. https://arxiv.org/abs/2406.09308.

[6] L. Saldyt and S. Kambhampati. Algorithmic Language Models with Neurally Compiled Libraries. 2024. https://arxiv.org/abs/2407.04899.

[7] Qwen Team. Qwen2.5 Technical Report. 2024. https://arxiv.org/abs/2412.15115.

[8] E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-Rank Adaptation of Large Language Models. 2021. https://arxiv.org/abs/2106.09685.

[9] T. Vergara-Browne and A. Soto. Tracr-Injection: Distilling Algorithms into Pre-trained Language Models. 2025. https://arxiv.org/abs/2505.10719.

[10] F. Dietz and D. Klakow. IGC: Integrating a Gated Calculator into an LLM to Solve Arithmetic Tasks Reliably and Efficiently. 2025. https://arxiv.org/abs/2501.00684.