THE ILLUSION OF COMPUTATION: A CIRCUIT-COMPLEXITY ACCOUNT OF STATE-TRACKING FAILURE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) reliably fail at core algorithmic tasks — multi-digit arithmetic, sorting, recursive procedures — commonly attributed to insufficient scale or context length. We argue these failures are consistent with a proven computational ceiling: bounded-precision transformers are contained in TC^0, and parallel-scan state-space models (SSMs) inherit essentially the same ceiling (one plausible explanation among several, not an automatic theory-to-practice inference, Section 3.2). We derive a falsifiable *budget* prediction — a task is out of reach unless realized scratchpad depth D_{realized} meets task-inherent depth D_{task} — and test it with a benchmark that independently manipulates D_{task} and D_{realized}: three depth-parameterized task forms under four prompting conditions, $K \in \{2, \ldots, 128\}$, paired with linear probing, across three open-weight transformers (Qwen2.5-0.5B/3B/14B) and an SSM control (Mamba-2.8B). Every CoT-based condition collapses to near-floor accuracy by moderate K, but the *mechanism* differs by task form and size: modular's failure is genuine arithmetic error surviving full format compliance at 3B/14B, not the compliance collapse driving 0.5B's failure; register shows the smooth per-step decay our auxiliary noise model predicts; and permutation is clearest at 14B, where tool-augmented accuracy reaches near-ceiling (1.00 for $K \leq 16$) while all CoT-based conditions stay at floor — direct evidence of a genuine propagation failure, not a miscalibrated task. Linear probing shows hidden states retain decodable state well past behavioral collapse, most durably for permutation. Mamba-2.8B replicates the same form-dependent split, evidence the ceiling is architecture-general rather than attention-specific.

1 INTRODUCTION

LLMs routinely produce fluent, plausible-sounding solutions to problems that require exact, multi-step computation, long-form arithmetic, sorting, simulating a short program and just as routinely get the answer wrong once the problem exceeds a modest size, even when each individual step is trivial in isolation. The standard explanations are that the model is too small, undertrained on the relevant data, or out of context. We argue, on diagnostic empirical grounds, that LLMs instead fail because they cannot *maintain and update deterministic state across reasoning steps*, and that this is not a defect scaling will fix.

That diagnostic case, on its own, is empirical and informal. This paper connects it to an existing literature on the circuit complexity of transformers (Merrill & Sabharwal, 2023b; Merrill et al., 2022; Merrill & Sabharwal, 2023a; Strobl, 2023; Merrill & Sabharwal, 2024) and state-space models (Merrill et al., 2024), making four contributions: (1) **a formal grounding** of the claim in the TC^0 ceiling on bounded-precision transformers, extended to chain-of-thought (Merrill & Sabharwal, 2024) as a falsifiable *budget* prediction — kept explicitly separate from an auxiliary, non-theory-derived per-step-noise envelope layered on top purely for testability (Sections 3 and 4.2); (2) **an architecture-general formulation**, showing via Merrill et al. (2024) that parallel-scan SSMs face essentially the same ceiling, reframing the claim as being about parallelizable, bounded-precision sequence architectures generally (Section 3); (3) **a depth-parameterized benchmark** that, unlike prior diagnostics including the concurrent WMF-AM (Hou et al., 2026), independently manipulates

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

task-inherent depth (D_{task}) and realized scratchpad depth (D_{realized}) via four prompting conditions (Section 4); and (4) **a representational/behavioral dissociation analysis** using linear probes paired with behavioral scores, extending Anonymous (2024) across the full depth sweep rather than one fixed difficulty level.

We report a completed pilot (Section 5): three transformer sizes, one SSM architecture control, and the probing analysis, with only a frontier closed-model arm still to be run and marked explicitly as pending rather than filled with placeholder numbers.

2 Related Work

Empirical failure diagnostics. A growing body of work documents that LLMs fail at algorithmic tasks in ways that do not simply improve with scale (Shojaee et al., 2025). This tradition is diagnostic and architecture-agnostic; without a mechanistic account it is vulnerable to the objection that the next model or prompting trick will dissolve the gap. Supplying that account is this paper's goal.

Circuit complexity of transformers. Building on hard/soft attention results (Hahn, 2020; Merrill et al., 2022), Merrill & Sabharwal (2023b) show log-precision, no-CoT transformers are contained in TC^0 (logical characterization: Merrill & Sabharwal, 2023a; tightened uniformity: Strobl, 2023; Chiang, 2025). Merrill & Sabharwal (2024) extend this to CoT, showing reachable complexity grows with decoding steps ($\mathsf{TC}^0 \to \mathsf{L}$ at $O(\log n)$ steps, full automaton simulation at $O(n)$) — the literature Section 3 draws on directly, not previously connected, to our knowledge, to a benchmark manipulating realized CoT length as an independent variable. A complementary line asks what transformers *can* compute given external memory or unbounded steps (Schuurmans, 2023; Giannou et al., 2023; Weiss et al., 2021) — existence proofs about constructions, not claims about conventionally trained models, discussed relative to our tool-augmented condition in Section 6.

State-space models. SSMs are often motivated as a remedy for transformers' state-tracking weakness via explicit recurrent state. Merrill et al. (2024) show this does not survive contact with the same analysis: the associative-scan formulation needed for parallel training places SSMs in TC^0 too, central to our architecture-generality claim and motivating the SSM control condition in Section 4.

Probing for latent state. Anonymous (2024) find linear probes decode intermediate state from hidden representations even when the final output is wrong — not a counterexample to Section 3.1's ceiling but a needed refinement (bounding what a circuit can *losslessly propagate*, not what it can represent noisily at one layer). We adopt this framing (Section 3.5) and extend the probing analysis across the full depth sweep and all three task forms.

Concurrent depth-parameterized benchmarks. Independently and concurrently, Hou et al. (2026) introduce WMF-AM, which also parameterizes cumulative state-tracking depth across 28 models. Our design differs (Section 5.6) in independently manipulating D_{task} vs. D_{realized}, pairing behavioral results with representational probing, and an explicit architecture-generality axis; we treat the two as complementary, not competing claims.

3 Theoretical Foundations: Why Stateful Computation Is Out of Reach

The central claim of this paper is empirical: LLMs fail on arithmetic, sorting, and recursive tasks not because of scale or context length, but because they cannot maintain and update deterministic state across reasoning steps. This section grounds that claim in known bounds on fixed-depth neural sequence models, converting "LLMs are not universal Turing machines" from a near-tautology into a falsifiable, architecture-general claim about *which* problems are reachable, and giving Section 4 a principled axis — circuit depth relative to sequential dependency length — to vary.

3.1 TRANSFORMERS AS CONSTANT-DEPTH THRESHOLD CIRCUITS

A single forward pass of a decoder-only transformer with bounded numerical precision and no intermediate decoding steps is simulable by a DLOGTIME-uniform family of constant-depth threshold circuits (Merrill & Sabharwal, 2023b;a; Strobl, 2023; Chiang, 2025): log-precision, no-CoT transformers are contained in uniform TC^0, a strict subclass of NC^1, L, NL, P under standard assumptions.

Proposition 1 (Task-class ceiling, no scratchpad)**.** *Under standard complexity assumptions (*$\mathsf{TC}^0 \subsetneq \mathsf{NC}^1 \subsetneq \mathsf{L} \subsetneq \mathsf{NL} \subsetneq \mathsf{P}$*), a log-precision, single-pass transformer cannot exactly solve any problem hard for a class strictly larger than* TC^0 *— e.g. graph connectivity (*NL*-complete), linear equality systems (*P*-complete), or finite-state automaton simulation on unbounded input (*NC^1*-complete) (Merrill & Sabharwal, 2023b; 2024). This is an asymptotic statement about a circuit* family*, one per input length, not about any single finite input or trained model's weights; Section 3.2 discusses what it does and does not license about a specific model failing a specific finite benchmark.*

This subsumes two of our three task families directly: iterated accumulation and automaton simulation are both *sequential state update*, where dependency-chain depth grows with input length. Sorting is NC^1 via sorting networks, already ruled out by Proposition 1 for exact single-pass in-context sorting. Failure rates rising with depth are *consistent with* this proven ceiling, not proof of it — the same pattern is consistent with several non-complexity-theoretic explanations, discussed next.

3.2 ASYMPTOTIC CEILINGS DO NOT, BY THEMSELVES, EXPLAIN ONE MODEL'S FINITE-SIZE FAILURES

Bridging an asymptotic circuit-family statement and one trained model's finite-K behavior is an inference, not an identity. A model can fail for reasons unrelated to TC^0: non-convergence to the capacity-saturating circuit, an under-representative training distribution, instruction-following breakdown that prevents even attempting the procedure (Section 5.2 documents exactly this), representation bottlenecks unrelated to circuit depth, or decoding-time instability — observationally similar to the complexity account at the level of "accuracy drops with K," which is why Section 4's quantitative prediction ($(1-\epsilon)^K$ decay under forced full-trace CoT, Proposition 2) is a sharper test: most alternatives do not predict that specific functional form. Where our data fail to isolate the mechanism cleanly, Sections 5.2 and 6 say so rather than treat the ceiling as confirmed by default; we defend that the observed failures are *consistent with, and predicted in advance by*, the ceiling — not that no other mechanism could explain them, nor that finite behavioral evidence could ever prove an asymptotic claim on its own.

3.3 WHAT EXTRA DECODING STEPS BUY — AND WHAT THEY DO NOT

Merrill & Sabharwal (2024) show decoding steps genuinely increase expressive power: with $\Theta(\log n)$ steps the reachable class expands from TC^0 to L; with $\Theta(n)$ steps, transformers can simulate arbitrary automata.

Proposition 2 (Chain-of-thought closes the gap only up to its own length)**.** *A transformer decoder using $T(n)$ intermediate generation steps is no more powerful than a computation running in $O(T(n))$ time/space beyond the model's constant-depth per-step computation (Merrill & Sabharwal, 2024). Any task whose minimal sequential dependency length exceeds the CoT tokens* actually produced *remains out of reach, regardless of parameter count.*

This reframes "scaling alone cannot bridge this divide" as testable: failure should be predictable from $r = D_{\text{task}}/D_{\text{realized}}$, the ratio the benchmark sweeps, not model size in isolation. This is a *necessary-condition budget claim only* — it does not, by itself, predict any particular functional form for how accuracy degrades short of that budget being met; Section 4.2 makes explicit the separate, auxiliary $(1-\epsilon)^K$ model layered on top purely for testability, and Section 5.3 keeps the two apart when reporting results.

3.4 THE CLAIM IS ARCHITECTURE-GENERAL, NOT ATTENTION-SPECIFIC

A natural rebuttal is that state-space architectures (S4, Mamba), explicitly designed with persistent hidden state, escape this. Merrill et al. (2024) show otherwise: the associative-scan formulation needed for parallel training places SSMs in TC^0 too, upgrading the claim to one about *parallelizable sequence architectures under bounded precision* generally, not self-attention specifically.

Proposition 3 (Architecture-general ceiling)**.** *Any sequence model whose per-step computation is a* DLOGTIME*-uniform constant-depth threshold circuit — including bounded-precision transformers (Merrill & Sabharwal, 2023b) and parallel-scan SSMs (Merrill et al., 2024) — inherits the ceiling in Proposition 1 without scratchpad, and the bound in Proposition 2 with autoregressive steps.*

This predicts a Mamba-family model should degrade at a rate comparable to a similar-size transformer despite its explicit recurrent state; the benchmark's architecture-control condition (Section 4.4) tests this directly.

3.5 REPRESENTATIONAL CAPACITY VS. BEHAVIORAL CAPACITY

Can encode and *does use* are different claims. Anonymous (2024) find linear probes recover intermediate arithmetic state from hidden representations even when the final output is wrong — not in tension with Propositions 1 to 3 but a refinement: a TC^0 circuit can compute partial, noisy correlates of a higher-complexity quantity without propagating it losslessly across steps. We state the paper's thesis at this more precise level:

> LLMs' failure at algorithmic tasks is not a failure to represent state, but a failure to *deterministically read, update, and re-propagate* state across a chain of steps whose length exceeds what the model's effective circuit depth (base depth plus realized scratchpad length) can support.

This directly motivates the paired probing experiment (Section 4.3): measure probe-decodable state fidelity alongside final-answer accuracy as depth increases, testing whether representation degrades gracefully while behavior fails sharply.

4 A DEPTH-PARAMETERIZED BENCHMARK FOR STATE-TRACKING FAILURE

Section 3 makes a specific prediction: performance should degrade as a function of task-inherent sequential depth relative to realized scratchpad length ($r = D_{\text{task}}/D_{\text{realized}}$), not as a function of task identity, model size, or token budget in isolation. Existing diagnostics — including the informal version of this paper's own starting claim (Section 1) and the independently-released WMF-AM benchmark (Hou et al., 2026) — test depth-dependent degradation but do not (a) hold task *semantics* fixed while sweeping depth, (b) separately vary D_{task} and D_{realized}, or (c) pair behavioral accuracy with representational (probed) state fidelity. This section specifies a benchmark that does all three. Full implementation is released alongside this paper (task generators, prompting templates, scoring, and probing code); we describe the design here and report results in Section 5.

4.1 DESIGN PRINCIPLE: ONE TASK FAMILY, ONE DEPTH KNOB

Rather than three unrelated task types, we use a single abstract task family — **sequential state accumulation** — instantiated in three surface forms so that any observed architecture- or model-level effect can be attributed to depth, not to surface-form idiosyncrasies (tokenizer quirks in multi-digit arithmetic, familiarity effects from pretraining exposure to sorted lists, etc.).

Arithmetic form Iterated modular accumulation: $s_0 = 0$, $s_i = (s_{i-1} \cdot a_i + b_i) \bmod m$ for $i = 1, \ldots, K$, given as a flat sequence of (a_i, b_i) pairs; report s_K. K is the depth knob. Modular arithmetic with a small, fixed m (e.g. $m = 97$) prevents shortcut memorization of common products and keeps the answer space small and easy to score exactly.

Symbolic form Register-machine simulation: a sequence of K instructions (`SET`, `ADD`, `SWAP`, `IFZERO`) operating on a small set of named registers, given in program order; report the final register values. This is the automaton-simulation instantiation of Proposition 1 (NC^1-complete for unbounded K) and is close to what "recursion" meant in the original draft, but made precisely controllable in depth.

Structural form Iterated permutation composition: apply K given permutations of a fixed small set (six elements) in sequence to an initial ordering; report the final ordering. This generalizes the sorting task — sorting networks are a special case of bounded-depth permutation composition — while removing the comparison-based framing that lets models pattern-match against pretraining-seen sorting explanations.

Each form is scored exactly (exact match on final state). K is swept over $\{2, 4, 8, 16, 32, 64, 128\}$ per form, with 100 sampled instances per (form, K) cell. One nuance worth disclosing here: random instruction sequences can be *effectively* shorter than their nominal K (e.g. an `IFZERO...SKIP_NEXT` chain that never triggers a real branch), so K is an upper bound on task-inherent depth per instance, not a guarantee every step is consequential; Section 5.3 reports how often this affects the falsifiability analysis.

4.2 Separating D_{TASK} from D_{REALIZED}

For each (form, K) cell, we run the same instance under four prompting conditions that independently vary D_{realized}:

1. **No scratchpad.** Direct answer, no intermediate tokens permitted (approximates the single-pass TC^0 regime of Proposition 1).
2. **Free CoT.** "Think step by step," no constraint on scratchpad length; we measure D_{realized} post hoc as the number of distinguishable intermediate state updates the model actually writes down.
3. **Forced full-trace CoT.** The model is instructed and few-shot demonstrated to emit exactly one state-update line per step. We distinguish three notions of realized depth rather than treat them as one: $D_{\text{prompted}} = K$ (what the instruction asks for); D_{emitted}, the number of state-update lines actually written regardless of correctness (Section 5's "compliance"); and D_{valid}, the longest *correct-from-step-1* prefix before the trace departs from the true trajectory (`scoring.py`'s `step_correct`). These coincide only when the model both complies and tracks state correctly throughout; "$D_{\text{realized}} = D_{\text{task}}$ by construction" is a claim about D_{prompted} only, and Section 5 reports the other two separately. This is the condition Proposition 2 predicts should approach ceiling if the failure is purely a budget effect rather than a deeper inability to update state given the steps.
4. **Tool-augmented.** The model may emit code that is executed externally and the result returned. This is the "escape hatch" condition — success here alongside failure in (1)–(3) would show the limitation is about in-context state propagation specifically, not about the model's understanding of the task, and connects to the memory-augmented and looped-transformer construction results of Schuurmans (2023) and Giannou et al. (2023), which establish that *some* augmentation suffices in principle.

Falsifiable prediction, and what the theory does and does not predict. Proposition 2 is a budget claim only: out of reach unless $D_{\text{realized}} \geq D_{\text{task}}$ is realized, with no particular functional form for degradation short of that. To make the budget claim testable, we layer a separate, explicit null model on top — constant per-step error ϵ, giving accuracy $\approx (1-\epsilon)^K$ once $D_{\text{realized}} \geq K$ is actually achieved — and test it only among instances that achieve that depth (by either D_{emitted} or D_{valid}), not among all K-prompted instances. A faster-than-predicted drop among those instances is evidence against the auxiliary envelope, not against Proposition 2's weaker budget claim, which a compliance or validity shortfall alone already explains; Section 5 keeps the two outcomes distinct.

4.3 Representational probing, paired with behavior

For open-weight models, at each step i we train a linear probe on the residual stream (at the token position for step i in the forced full-trace completion) to predict the true intermediate state s_i, using held-out instances. This gives a probe-accuracy curve over K to place alongside the behavioral curve from Section 4.2: Section 3.5 predicts probe accuracy should decay more slowly than no-scratchpad behavioral accuracy, extending Anonymous (2024)'s dissociation across all three forms and the full depth sweep rather than one fixed difficulty level.

4.4 Model and architecture coverage

To test Proposition 3, the model list includes a non-attention control (a Mamba-family model at comparable parameter count, same four conditions, probed where architecture permits) alongside a transformer size sweep and one or more frontier closed models (behavioral only) still to be added.

5 Results

Four models (Qwen2.5-0.5B/3B/14B-Instruct, plus the Mamba-2.8B architecture control) have a complete sweep across all three task forms, four conditions, and the full K range; Table 1's remaining row (a frontier closed model) is not yet started.

5.1 Setup

Task instances are generated with a fixed seed and 100 instances per (form, K) cell (Section 4); decoding is greedy throughout; models run locally via Hugging Face `transformers`. Two implementation bugs, both disclosed as corrections rather than footnotes since they changed reported numbers, were found and fixed during the pilot: a tool-augmented generation budget initially fixed regardless of K silently truncated otherwise-correct unrolled code at higher K (fixed by scaling with K; affected cells regenerated), and a tokenizer input-truncation bug affected `permutation` $K{=}128$ across every condition and model, discarding trailing prompt content and producing completions that read as capability collapse but were a harness artifact (fixed by raising the cap and repeating the tool-augmented instruction after the data; 12 affected cells regenerated). Section A.5 gives the full technical detail.

Table 1: Models evaluated and pilot status. "Complete" cells cover all 2,100 (form, K, condition, instance) combinations.

Model	Params	Architecture	Access	Probed?	Status
Qwen2.5-0.5B-Instruct	0.5B	Transformer	Open	Y	Complete
Qwen2.5-3B-Instruct	3B	Transformer	Open	Y	Complete
Qwen2.5-14B-Instruct	14B	Transformer	Open	Y	Complete
Mamba-2.8B	2.8B	SSM	Open (base)	N	Complete
Frontier closed model	—	Transformer	API	N	Not started

5.2 Headline result: accuracy vs. depth by condition

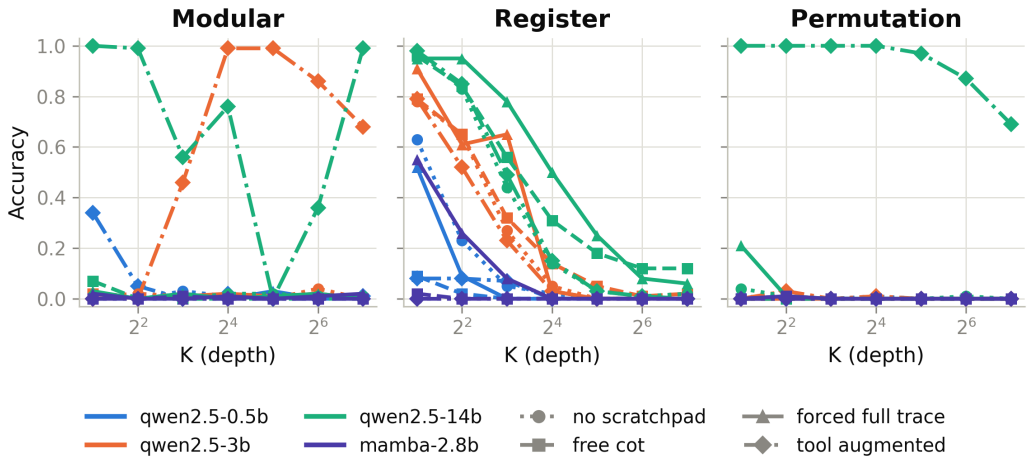

Figure 1: Behavioral accuracy vs. depth K, by task form and prompting condition, for all four models (color = model, line style/marker = condition). Every CoT-based condition in every form collapses to near-floor by some K for every model; the one exception, 14B tool-augmented on `permutation`, stays near ceiling through moderate K before decaying (Section 5.6). Mamba-2.8B's tool-augmented lines sit at floor for a disclosed, unrelated reason (base checkpoint, never emits code, Section 5.4), not an architecture effect. See caveats in text before reading this as full support for size-independence or architecture-generality.

Prediction being tested: no-scratchpad accuracy should degrade with K largely independent of model size (Proposition 1); forced full-trace accuracy should be substantially higher, closer to the $(1-\epsilon)^K$ envelope (Proposition 2). The no-scratchpad prediction holds unremarkably at every size. The forced-full-trace prediction does *not* hold uniformly, and the reason is form-dependent: **modular** is essentially zero at every K and size, but for different reasons — compliance itself collapses at 0.5B, while 3B/14B stay compliant and simply fail the arithmetic (Section 5.3); **register** shows the smoothly decaying failure the envelope predicts, improving with scale; and **permutation** is at floor under every CoT-based condition down to $K = 2$ — also what a poorly calibrated task would look like, a reading only a demonstrably capable model can rule out. At 14B, tool-augmented accuracy on permutation resolves this directly: 1.00 for $K \leq 16$, decaying only gracefully to 0.69 by $K{=}128$ (Section 5.6), while all three CoT-based conditions remain at floor by $K{=}8$ at the same size — a model that can solve the task's representation still cannot propagate it in context, the strongest evidence here that permutation's floor is a genuine state-propagation failure. Section A.1 gives the full per-size, per-form breakdown. This form-dependence — whether the model even attempts the procedure, and which mechanism breaks first — is itself the headline finding: neither question has one answer across task forms.

5.3 Per-step error rate fits

Table 2: Fitted per-step error rate ϵ (log-linear least-squares fit of accuracy$(K) \approx (1-\epsilon)^K$) for forced full-trace, all-completions vs. restricted to the naturally-compliant subset ($\geq K$ state-update lines emitted, $n \geq 10$ per K; register omitted from the compliant columns since its all-completions fit is already clean, see text).

		All completions		Compliant-only	
Model	Form	$\hat{\epsilon}$	Resid. SS	$\hat{\epsilon}$	Resid. SS (K used)
Qwen2.5-0.5B	modular	0.132	0.966	—	— ($K{\leq}4$ only)
Qwen2.5-0.5B	register	0.146	0.328	—	—
Qwen2.5-0.5B	permutation	0.148	0.885	—	— (none compliant)
Qwen2.5-3B	modular	0.101	1.247	0.101	1.250 (2–128)
Qwen2.5-3B	register	0.135	0.146	—	—
Qwen2.5-3B	permutation	0.148	0.885	0.148	0.885 (2–128)
Qwen2.5-14B	modular	0.107	1.165	0.107	1.165 (2–128)
Qwen2.5-14B	register	0.026	0.070	—	—
Qwen2.5-14B	permutation	0.146	0.629	0.146	0.628 (2–128)
Mamba-2.8B	modular	0.066	1.765	0.154	0.820 (2–64)
Mamba-2.8B	register	0.142	0.168	—	—
Mamba-2.8B	permutation	0.147	0.884	0.248	0.428 (2,4,8,16,64)

The fitted $\hat{\epsilon}$ values are similar across forms and sizes (0.10–0.15, register at 14B excepted), but the residual sums of squares tell the real story: **modular**'s stays large at every size (0.97–1.25) and **permutation**'s stays large through 3B (0.885), while **register** improves sharply with scale (0.33 at 0.5B to 0.07 at 14B) — the closest any (model, form) pair comes to Proposition 2's per-step-noise envelope. Section A.2 gives the full per-size numbers.

Is the modular/permutation residual actually a compliance artifact? A large residual could mean the model fails arithmetic step-by-step, or has simply stopped attempting the forced trace. We refit $\hat{\epsilon}$ on only the naturally-compliant subset ($\geq K$ state-update lines emitted, K with ≥ 10 compliant completions) — Table 2's right-hand columns. The split is clean: for **Qwen2.5-3B, -14B, and Mamba-2.8B**, natural compliance on modular/permutation is already 68–100% at nearly every K, and restricting to compliant completions barely moves the fit (e.g. 3B modular: $0.101/1.250$ vs. 1.247) — a genuine, compliance-independent arithmetic/composition failure. Only **Qwen2.5-0.5B** shows the compliance-collapse story we originally reported for all sizes. **Register**'s fit *improves* under the same conditioning at every size — the one form where the theory's auxiliary envelope is tested cleanly. This is strictly stronger evidence against the theory's simplest prediction than a compliance artifact would have been, since for modular/permutation at 3B+ it cannot be dismissed as declining to try. Section A.2 gives the full compliance-conditioned numbers plus three further checks: a D_{valid} (correct-from-start prefix) analysis, a falsifiability check against all 8,400 completions collected (holding in 98.0%), and a targeted 0.5B retry pilot showing its compliance gap survives resampling.

5.4 Architecture generality: transformer vs. SSM

Run. The architecture control is `state-spaces/mamba-2.8b-hf` (2.8B, size-matched to Qwen2.5-3B), run through the full sweep. As a *base*, non-instruct checkpoint it makes tool-augmented (0% of completions contain a fenced code block) and no-scratchpad uninformative; we lean on forced-full-trace, a completion-style task base models handle reasonably (Section A.3 details this caveat).

Figure 1 and Table 2 show the result is form-dependent exactly as Proposition 3 needs for a meaningful test: **modular** collapses near floor with a large, non-decaying residual ($\hat{\epsilon} = 0.066$, residual 1.77) matching Qwen2.5-3B/-14B's genuine-failure pattern rather than 0.5B's compliance collapse; **permutation**'s residual (0.884) matches Qwen2.5-0.5B/3B's 0.885 to three decimal places; and **register**'s envelope ($\hat{\epsilon} = 0.142$, residual 0.168) lands almost exactly on Qwen2.5-3B's (0.135, 0.146), though its D_{valid} growth diverges from the Qwen models' (Section A.2). This is evidence *for* Proposition 3, held at the confidence the design supports: one SSM checkpoint at one size with a base-vs-instruct confound, not architecture-generality demonstrated across an SSM sweep or an instruction-tuned SSM; Section A.3 gives the full breakdown.

5.5 Representational vs. behavioral state capacity

Run for all three models. Figure 2 overlays per-layer linear-probe ceiling accuracy against no-scratchpad behavioral accuracy, one panel per task form. The prediction, following Anonymous (2024), is that probe accuracy decays more slowly than behavioral accuracy; the answer is form-dependent. **Register** confirms it once K moves past the trivial end: probe and behavioral accuracy start close (14B: 0.96/0.98 at $K = 2$) but the gap is enormous by $K = 16$ (14B: 0.65/0.14), narrowing again only as both approach floor. **Permutation** (probing viable only for 3B/14B) instead shows the gap at *every* K, strikingly constant rather than narrowing (probe ≈0.23–0.53 vs. behavioral floor 0.00–0.04) — the clearest representational/behavioral dissociation in the pilot, precisely because it never closes. **Modular** shows no sustained gap: both curves are already at floor by $K = 2$–16 for all three models, consistent with the early, size-dependent floor Section 5.3 documents. Section A.4 gives the full per-K numbers and a data-quality caveat on 0.5B's noisy high-K register probe curve.

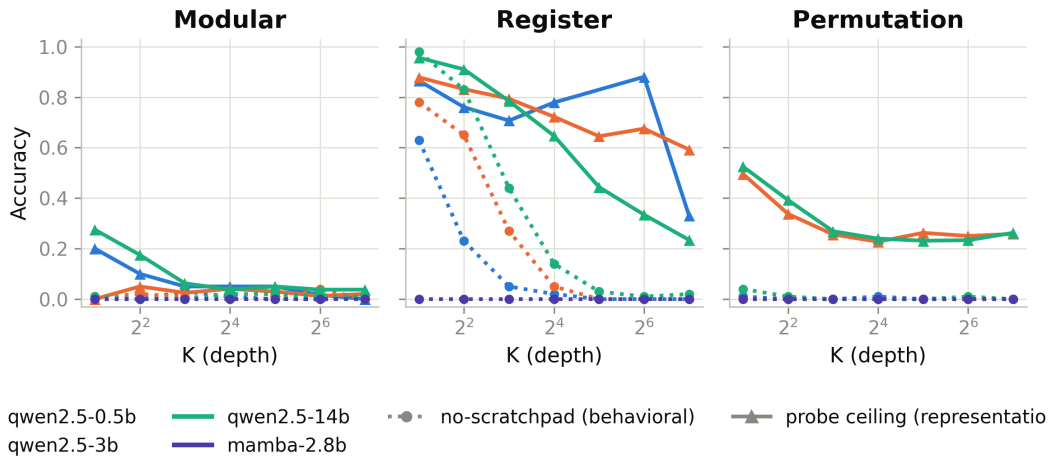

Figure 2: Probe ceiling accuracy (solid, triangle) vs. no-scratchpad behavioral accuracy (dotted, circle), by form, for all three models (color). Register's gap opens with K, narrowing only as both curves approach floor; permutation's is roughly constant across the whole range (0.5B omitted, see text); modular shows no sustained gap.

5.6 Tool-augmented condition

Tool augmentation should be the "escape hatch": if the model correctly delegates the state update to executed code, in-context propagation is no longer the bottleneck and accuracy should approach ceiling regardless of K (Section 4.2). It does not, and the breakdown by form shows why this is a secondary finding about code generation, not a counterexample to the state-tracking claim. Table 3

reports, per form and averaged over all K: the rate at which a fenced code block is present at all, the rate at which that code executes and yields a parseable result (i.e. defines the instructed `result` variable without erroring), and final accuracy.

Table 3: Tool-augmented breakdown, averaged over $K \in \{2, \ldots, 128\}$ (700 completions per form per model).

Model	Form	Code block present	Executes to a result	Final accuracy
Qwen2.5-0.5B	modular	1.00	0.86	0.06
Qwen2.5-0.5B	register	0.51	0.22	0.03
Qwen2.5-0.5B	permutation	0.33	0.13	0.00
Qwen2.5-3B	modular	0.99	0.81	0.57
Qwen2.5-3B	register	1.00	1.00	0.22
Qwen2.5-3B	permutation	1.00	0.99	0.01
Qwen2.5-14B	modular	1.00	1.00	0.67
Qwen2.5-14B	register	0.98	0.97	0.36
Qwen2.5-14B	permutation	1.00	1.00	0.93

Mechanics (has-code/executes) improve monotonically with scale on every form (0.5B: 20–100%; 3B: 81–100%; 14B: 97–100%), but accuracy tracks that improvement unevenly and the dominant failure mechanism shifts with K and size (wrong algorithm at 0.5B, give-up/transcription error at 3B, a sharp form split at 14B where permutation's algorithm-formulation gap closes outright but modular/register do not gain as cleanly). Why code-generation gains convert so much more completely into accuracy for permutation at this scale is not yet explained and is worth a targeted follow-up. Section A.6 gives the full per-model, per-mechanism breakdown.

Comparison with WMF-AM, not yet done. A meaningful comparison to Hou et al. (2026) needs overlapping models or task semantics at comparable depths; we have neither yet, and defer rather than draw one without them.

6 Discussion, Limitations, and Conclusion

The central prediction held across three transformer sizes: every CoT-based condition collapses well before K reaches the tens, but the mechanism differs by size and form — a compliance collapse only at 0.5B, and for 3B/14B on modular and permutation a genuine, persistent arithmetic/composition failure that survives full compliance and does not resolve with scale. At 14B, tool-augmented execution succeeds at permutation where all CoT-based conditions fail at the same size, evidence of a genuine state-propagation failure rather than a badly-posed task; probing finds latent state surviving well past behavioral collapse, most durably for permutation; and Mamba-2.8B replicates the same form-dependent split, evidence the ceiling is architecture-general (tempered by one checkpoint, one size, and a base-vs-instruct confound). None of the three outcomes that would have counted against these claims (Section 4) occurred; Section A.2 gives the caveats. An open question we flag rather than paper over: register's D_{valid} grows with K while modular/permutation's does not, possibly an operation-familiarity confound (register resembles ordinary variable mutation in code) the pilot cannot separate from depth. If the ceiling is architecture-general, the theory-implied fix is not scale but architectures making step count a function of the input (e.g. Universal Transformer, Adaptive Computation Time) — untested here.

Limitations. Our theoretical results concern exact computation and assume bounded-precision arithmetic (all models run in `bfloat16`); the pilot omits a looped/universal-transformer positive control; free-CoT realized-depth estimation is an unvalidated heuristic; results use greedy decoding only (a targeted sampling check on the two highest-weight cells in Section 5.3 matched closely, but broader robustness is untested); and coverage is three transformer sizes plus one SSM checkpoint at one size — partial, not definitive, support for the scale- and architecture-generality claims.

AI use statement

In this work, we used a generative-AI coding agent (Claude, Anthropic) to implement and debug the pilot harness, including a GPU out-of-memory issue, a silent input-truncation bug affecting `permutation` K=128 (Section 5.1), model-loading network hangs, several probing-pipeline bugs, and a `mamba_ssm` dependency conflict resolved via an alternate checkpoint. In response to external review, it also audited existing completions for forced-full-trace compliance (revealing the

compliance-collapse story held only for the smallest model), ran the resulting retry- and sampling-based follow-up pilots, computed the D_{valid} analysis, and drafted the empirical prose in Section 5 from the harness's output. We did not use generative AI to produce the paper's theoretical claims or propositions (Section 3), which are the authors' own restatement and extension of the cited results, and we reviewed all AI-assisted code and prose against the underlying data before including it. We take responsibility for the final content of this work.

Ethics Statement

This work evaluates open-weight language models on synthetic, procedurally generated algorithmic tasks; it involves no human subjects, no personal or sensitive data, and no dataset release beyond synthetic task instances from a public seed. We do not identify ethical concerns beyond the general considerations for LLM capability evaluation.

Reproducibility Statement

Task generators, prompting templates, scoring code, the tool-execution sandbox, and the probing pipeline are released at **[TODO: repo URL]**, with a fixed generation seed (`config.py`) so all reported task instances are exactly reproducible. Section 5.1 and Table 1 give the model list and decoding settings used. The theoretical results in Section 3 are restatements of results proved in full in the cited papers (Merrill & Sabharwal, 2023b; 2024; Merrill et al., 2024).

References

Anonymous. States hidden in hidden states: Implicit discrete state representations emerge in LLMs' hidden states. 2024. arXiv:2407.11421.

David Chiang. Transformers with soft and average-hard attention: AHAT equals SMAT under DLOGTIME-uniformity. 2025.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. 2023. arXiv preprint.

Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020.

Dengzhe Hou, Lingyu Jiang, Deng Li, Zirui Li, Fangzhou Lin, and Kazunori D. Yamada. WMF-AM: Probing LLM working memory via depth-parameterized cumulative state tracking. 2026. arXiv:2603.27343.

William Merrill and Ashish Sabharwal. A logic for expressing log-precision transformers. In *NeurIPS*, 2023a.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023b. arXiv:2207.00729.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *ICLR*, 2024. arXiv:2310.07923.

William Merrill, Ashish Sabharwal, and Noah A. Smith. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, 10:843–856, 2022.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *ICML*, 2024. arXiv:2404.08819.

Dale Schuurmans. Memory augmented large language models are computationally universal. 2023. arXiv preprint.

Parshin Shojaee et al. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. 2025. arXiv preprint.

Lena Strobl. Average-hard attention transformers are constant-depth uniform threshold circuits. 2023. arXiv:2308.03212.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. *ICML*, 2021.

A Appendix

The main text keeps only the headline claim and numbers for each result; this appendix gives the full derivations, per-model breakdowns, and secondary analyses referenced from the main text by subsection.

A.1 Headline result: full per-size breakdown

At 0.5B, the no-scratchpad prediction holds unremarkably: accuracy is at or near floor for all three forms by $K = 8$ (register is the outlier, starting at 63% at $K = 2$ before collapsing by $K = 32$). The forced-full-trace prediction does *not* hold, but not for the reason Proposition 2 anticipates: rather than a smoothly compounding per-step error, the model's compliance with the forced-trace instruction itself collapses with K. For **modular**, forced-full-trace accuracy is essentially zero at every K, including $K = 2$ (0.02): inspecting completions shows that from $K \geq 16$ onward the model frequently skips the instructed step-by-step trace entirely and emits a single ungrounded guess (e.g. `"After step 16: state = 93\nAnswer: 93"` with no preceding steps), so $D_{\text{realized}} \ll K$ in practice despite the prompt demanding $D_{\text{realized}} = K$. **Register** behaves differently and more informatively: forced-full-trace accuracy starts much higher (0.52 at $K = 2$, vs. modular's 0.02) and decays roughly monotonically through $K = 8$ before hitting floor, i.e. the model does attempt the full trace on this form at low-to-moderate K and fails on the arithmetic itself, not on instruction-following — a genuinely different failure mode from modular's at the same nominal condition. **Permutation** is at floor under every condition at every K, including $K = 2$; Section 5.6 shows this form's failure starts even before the reasoning step, at the level of task representation. We should be explicit about a competing reading of this fact, rather than only the one we favor: floor performance at the *minimal* depth is also exactly what a poorly calibrated task would look like — if the surface form of the permutation-composition prompt is simply hard for this model to parse into the right representation regardless of K, then increasing K is not isolating state-tracking demands at all, because the task never leaves the floor for any manipulation to be visible against. A task-calibration problem and a genuine, depth-independent representation failure predict the same $K = 2$ floor at one model size, and what would distinguish them is whether a substantially larger model clears the floor at low K — calibration problems are generally scale-sensitive in a way a genuine circuit-depth ceiling on this specific, minimal instance should not be.

At 3B, the picture both confirms and complicates the 0.5B story. On **register**, scale helps substantially in the regime where the model attempts the task at all: no-scratchpad rises to 78% at $K = 2$ (vs. 63%) and forced-full-trace to 91% (vs. 52%), and forced-full-trace now holds up through $K = 8$ (65%) before a sharp collapse at $K = 16$ (3%) — a later and sharper collapse point than 0.5B's, but a collapse nonetheless; scale moved *where* the wall is, not whether there is one. On **modular**, direct-answer and CoT conditions barely move at all (still at or near floor at every K, including $K = 2$), but tool-augmented accuracy does something neither model does anywhere else in this pilot: it is near zero at $K = 2$–4, jumps to 99% at $K = 16$–32, and then *declines again* to 68% by $K = 128$ (Section 5.6 traces this to a change in what code the model writes at each regime, not a change in whether it writes code at all). Scale does not monotonically rescue this condition; it opens a specific escape hatch at intermediate depth and then partially closes it again at high depth. On **permutation**, every CoT-based condition remains at floor across the entire K range at 3B too, including $K = 2$ — *despite* the model now reliably writing code that parses and executes for this form (Section 5.6: 100% has-code, 99% executes, up from 0.5B's 33%/13%), it still computes the wrong permutation essentially every time (1% correct). This is the comparison the main text calls for: a 6× larger model that demonstrably *can* engage with the task representation (it writes and runs code targeting it) still cannot solve it, which weakens the pure task-miscalibration reading relative to a genuine, scale-resistant difficulty with permutation composition specifically — though at 3B alone it does not eliminate that reading outright, since 3B could still be below whatever size threshold calibration would require.

At 14B, that threshold is crossed decisively: tool-augmented accuracy on `permutation` reaches 1.00 for $K \leq 16$ and decays only gracefully thereafter (0.97 at K=32, 0.87 at K=64, 0.69 at K=128; Section 5.6 traces the K=128 shortfall to transcription-fidelity errors copying the permutation list into code, not to incorrect composition logic), while all three CoT-based conditions on `permutation` remain at or near floor by $K = 8$ at 14B too (Table 2). A model that reliably solves the task's representation is now in hand, and it still cannot propagate that representation across steps without an external memory to write to — this is the strongest evidence in the current pilot against the task-miscalibration reading, and correspondingly the strongest support for reading permutation's CoT-condition floor as a genuine, depth-driven state-propagation failure rather than a badly-posed prompt.

A.2 Per-step error rate fits: compliance conditioning, D_{VALID}, falsifiability, and the 0.5B retry pilot

Compliance-conditioned fits, in full. The result splits cleanly by model size, and it is not the split we expected going in. For **Qwen2.5-3B, Qwen2.5-14B, and Mamba-2.8B**, natural compliance on modular is already 89–100% at every K we could fit (e.g. Qwen2.5-3B: 100/100 compliant at $K = 2$, 93/100 at $K = 64$, 100/100 at $K = 128$) — the model reliably writes all K state-update lines. Restricting to only those compliant completions changes the fit by less than 0.006 in $\hat{\epsilon}$ and essentially not at all in residual (3B: 0.101/1.250 vs. 1.247 all-completions; 14B: 0.107/1.165 vs. 1.165) — compliance was never the bottleneck for these three models on modular, and the large, non-decaying residual is a robust, compliance-independent signature of genuine, near-total arithmetic failure despite full compliance. The same pattern holds on **permutation** for these three models: Qwen2.5-3B and -14B stay 68–100% compliant through $K = 128$, and the compliant-only fit ($0.148/0.885$, $0.146/0.628$) is indistinguishable from the all-completions fit in Table 2 — these models dutifully write out every step and still land on the wrong final ordering essentially every time, exactly the "fluent, executable, consistently wrong" pattern Section 5.6 documents independently for the tool-augmented condition. **Qwen2.5-0.5B** is the genuine exception, and only for 0.5B does the format-compliance story we originally told for all three sizes actually hold: natural compliance on modular collapses from 100% ($K = 2$) to 0% by $K = 8$, leaving too few compliant completions to fit past $K = 4$, and compliance on permutation is 0% at *every* K we tested, leaving no compliant completions to fit at all. For 0.5B specifically, the large raw residual really does reflect a compliance artifact, not (yet) a tested claim about Proposition 2.

This reframes the earlier reading of **register**'s residual, too, as a confirmation rather than a separate story: refitting on naturally-compliant completions only *improves* the fit at every size (0.5B: $0.774/0.203$ on the three lowest K where compliance survives; 3B: $0.119/0.134$; 14B: $0.032/0.018$, versus $0.146/0.328$, $0.135/0.146$, $0.026/0.070$ all-completions) — removing the zero-accuracy non-compliant completions from the average makes the smooth per-step-noise envelope fit *better*, the opposite of what modular and permutation show, and confirms register is the one form in this pilot where Proposition 2 is being tested cleanly by the raw numbers already in Table 2. Register remains the exception where per-step noise, not a harder failure, is the operative mechanism, consistent with Section 4.2.

D_{valid}: how far does state-tracking actually get, independent of K? D_{emitted} (Table 2) asks only whether the model writes enough lines, not whether they are *right*. We computed D_{valid} — the longest correct-from-step-1 prefix before the trace first departs from the true trajectory, `scoring.py`'s `step_correct` (Section 4.2) — for every completion already collected, and it sharpens the picture further. For **modular** and **permutation** on Qwen2.5-3B and -14B, mean D_{valid} is essentially *constant in* K: on modular, Qwen2.5-14B's mean D_{valid} is 1.0 at K=2 and stays within 0.1 of 1.0 at every other K through 128; on permutation it is 0.9–1.1 at every K for the same model, and Qwen2.5-3B's is 0.6–0.8 throughout. The model gets the trivially easy *first* transition right and fails at the second, whether the task asks for 2 steps or 128 — direct evidence that the failure here is not "ran out of realized depth" in any sense, since realized depth never meaningfully exceeds about one step regardless of how much is requested or how much is written. **Register** looks structurally different by this measure too, not just in raw accuracy: mean D_{valid} for Qwen2.5-14B grows from 1.9 (K=2) to 7.1 (K=16) before plateauing, and for Qwen2.5-3B from 1.6 to a peak of 7.6 at K=64 — the model genuinely propagates correct state for several real steps before losing the thread, growing (up to a point) as more steps are requested, exactly the qualitative signature a per-step-noise model should produce and modular/permutation do not. Mamba-2.8B is a partial ex-

ception worth flagging rather than smoothing over: its D_{valid} on modular and permutation matches the transformers' flat, near-zero shape, but on register it does *not* show the Qwen models' clean growth (1.0 at K=2, falling to 0.2 by K=8, then a noisy partial recovery to 1.2 by K=128 rather than a monotonic climb) — a disanalogy from the transformers that Section 5.4's more favorable reading of Mamba's raw accuracy fits does not surface, and that any future architecture- generality claim for register specifically should account for rather than elide.

Is Proposition 2's budget claim actually falsifiable, or is it true by how D_{realized} is measured? A fair question about the weak, necessary-condition reading of Proposition 2 (a task requiring depth D_{task} fails unless $D_{\text{realized}} \geq D_{\text{task}}$ is realized) is whether it is close to tautological once D_{realized} is read off the completion post hoc: if "realized" just means "whatever the model happened to do," the claim risks being unfalsifiable by construction. We checked this directly rather than leave it as a rhetorical question: across all 8,400 forced-full-trace completions collected (every model, form, and K), we counted cases where $D_{\text{emitted}} < K$ (fewer than K state-update lines written) yet the final answer was still correct — a direct violation of the necessary-condition reading, since fewer steps were emitted than the task nominally requires. This happened in 164 cases (2.0%), concentrated almost entirely in **register** (157 of 164) and largely explained by the instance-degeneracy nuance disclosed in Section 4.1: for 99 of the 164 (60%), the true state at the position where the model stopped emitting new lines *already equals* the true final state, i.e. the instance's remaining nominal steps were genuinely inconsequential (a chain of `IFZERO` no-ops, most often), not evidence the model computed ahead of what it wrote. The remaining 65 cases (0.8% of all completions checked) are not explained by that specific check and could reflect either subtler near-degeneracy our simple test misses or occasional coincidental matches; we report the number rather than explain it away. The claim is therefore not tautological in practice: it was directly checked against 8,400 completions and held in 98.0% of them, with the large majority of exceptions attributable to a disclosed, independent property of the task generator rather than to how D_{realized} is defined. This is a distinct question from whether the *auxiliary* $(1-\epsilon)^K$ envelope holds once the necessary condition is met, which Table 2 already shows is form-dependent and often does not.

Closing the 0.5B gap: does forced compliance change the answer? Table 2 left one real gap: Qwen2.5-0.5B has too few naturally-compliant completions to fit modular past $K = 4$ and none at all on permutation. We ran a small, targeted follow-up — not the full pilot sweep, $n = 25$ instances per cell rather than 100 — at the modular collapse point and two extremes ($K \in \{8, 32, 128\}$) and across permutation's range ($K \in \{2, 32, 128\}$), retrying generation up to three times per instance (one greedy attempt matching the original condition, then up to two temperature-sampled attempts, T=0.8) and keeping the first attempt that emitted at least K state-update lines. Compliance did not recover under retry: 2/25, 2/25, and 1/25 instances became compliant on modular at K=8, 32, 128 respectively, and 3/25, 0/25, 0/25 on permutation at K=2, 32, 128 — inspecting the retry attempts directly shows why: sampled retries mostly still emit 0–1 state-update lines regardless of K (one retry reached 10 of 128 required steps; every other retry we inspected stayed at 0–1), so this is not a case where a little more sampling variance would have closed the gap. And on the rare instances that did become compliant, accuracy was 0/8 across every cell — the same near-total failure the larger models show *with* full natural compliance (Table 2). The honest summary: 0.5B's gap is now smaller but not fully closed. We still lack a well-powered compliance-forced accuracy estimate at this size (the compliant counts above are too small to fit an envelope to), but what little compliant data retries did produce is consistent with, not contrary to, the larger models' pattern — and the retry behavior itself is now additional evidence for a real failure rather than a merely undertested one: at 0.5B, compliance does not appear to be a shallow sampling artifact a few retries can fix, which is a different and arguably more informative finding than simply obtaining a clean envelope fit would have been.

Why does register differ from modular and permutation? The three forms were designed as depth-only instantiations of one abstract task, but D_{valid} shows register is qualitatively different, not just easier — correct-prefix length grows with K for register but sits flat at ≈ 1 for the other two. Our best, unconfirmed hypothesis is differential pretraining exposure: register-machine state resembles ordinary variable mutation in source code, plausibly far more common in pretraining than modular arithmetic mod a fixed small modulus or abstract permutation composition. If right, the three forms vary in an orthogonal operation-familiarity dimension this pilot cannot separate from depth — weakening, though not undermining, the claim that $r = D_{\text{task}}/D_{\text{realized}}$ alone unifies all

three forms' behavior, since the confound would if anything work against finding a floor on the two forms where we do find one.

A.3 Architecture generality: full breakdown

One methodological caveat must be stated up front, because it changes which conditions are informative: unlike the three Qwen2.5-Instruct models, the Mamba-2.8B checkpoint is a *base*, non-instruct pretrained model. Its tool-augmented results reflect this cleanly and uninformatively for the architecture question: 0% of completions contain a fenced code block, at every K, for every form (vs. 20–100% for the smallest, least capable Qwen model) — this is an instruction-following failure, not an architecture-specific one, and we exclude tool-augmented from the architecture comparison entirely rather than let it masquerade as an SSM result. No-scratchpad is similarly hard to interpret alone (floor at every K for every form, 0.00–0.02), since a base model may simply not produce the requested one-line answer format regardless of architecture. Forced-full-trace is the condition we lean on: it only requires continuing the one-line "after step i: state = ..." pattern already demonstrated in the prompt's few-shot example, a completion-style task base models handle reasonably, so a failure here is more likely to reflect an actual inability to propagate state than an instruction-following gap.

Figure 1 and Table 2 show the result is form-dependent in exactly the way Proposition 3 would need it to be for a meaningful test: **modular** collapses to near floor at every K including $K = 2$ (0.00–0.02), matching the qualitative pattern seen at all three transformer sizes and with a similarly large, non-decaying-envelope residual ($\hat{\epsilon} = 0.066$, residual 1.77, vs. 0.97–1.25 for the transformers) — like Qwen2.5-3B and -14B (Table 2) rather than like Qwen2.5-0.5B, Mamba-2.8B keeps emitting the trace at moderate K (98/100 compliant at K=32) and still fails almost completely, so this residual is not primarily a compliance artifact here either, though its compliance does eventually collapse at the highest K (33/100 at K=64, 0/100 at K=128) where the transformers' does not. **Permutation** is at floor at every K (0.00–0.01) with a fitted residual of 0.884 — matching Qwen2.5-0.5B and -3B's 0.885 to three decimal places, the same degenerate, floor-at-every-K fit for the same reason. **Register** is the form that matters most here: forced-full-trace accuracy decays gracefully rather than collapsing outright (0.55 at $K = 2$, 0.26 at $K = 4$, 0.08 at $K = 8$, floor by $K = 16$), and the fitted envelope ($\hat{\epsilon} = 0.142$, residual 0.168) sits almost exactly on top of Qwen2.5-3B's ($\hat{\epsilon} = 0.135$, residual 0.146) and is a substantially better fit than Qwen2.5-0.5B's (residual 0.328) — this 2.8B SSM's per-step decay envelope on the one form where any model attempts sustained state tracking resembles a same-scale transformer's far more than it resembles a divergent, architecture-specific curve (though Section 5.3's D_{valid} analysis finds this convergence holds for the accuracy envelope specifically, not for how the correct-prefix length itself grows with K, where Mamba-2.8B does diverge from the Qwen models' clean growth — a disanalogy this raw-accuracy fit does not surface). Taken together — the same three-way split by form, the same near-identical degenerate fit for permutation, and a register fit that lands inside the transformer size range rather than off on its own — this is evidence *for* Proposition 3, not merely an absence of evidence against it: the state-tracking ceiling this paper argues for looks like a property of bounded-depth sequential computation generally, not of attention specifically. We hold this conclusion at the confidence the design actually supports: one SSM checkpoint at one size, with the base-vs-instruct confound disclosed above limiting which conditions count as evidence, is not the same claim as architecture-generality demonstrated across an SSM size sweep or with an instruction-tuned SSM checkpoint controlling for the confound directly; both are natural follow-ups this result motivates rather than settles.

A.4 Representational vs. behavioral state capacity: full per-K numbers

For **register**, the prediction holds clearly at every size once K moves past the trivial end: at $K = 2$ probe and behavioral accuracy are close for all three models (0.5B: 0.87/0.63, 3B: 0.88/0.78, 14B: 0.96/0.98 — 14B's behavioral accuracy is if anything marginally *above* its own probe estimate here, within the noise of two different completions being probed vs. scored), but the gap opens sharply by $K = 8$ (0.5B: 0.71/0.05, 3B: 0.79/0.27, 14B: 0.78/0.44) and is enormous by $K = 16$ (0.5B: 0.78/0.02, 3B: 0.72/0.05, 14B: 0.65/0.14). The gap narrows again only at the highest K, and only because both curves are approaching floor together, not because behavioral accuracy catches up. For **permutation**, probing was only viable for 3B and 14B — 0.5B's forced-full-trace format compliance on this form is too sparse at every K to fit any probes at all, consistent with Section 5.2's existing finding that permutation is at floor from $K = 2$ for this model. For the two

larger models the gap is present at every K and strikingly *constant* rather than narrowing: probe accuracy holds at roughly 0.23–0.53 across the full range (3B: 0.50 at $K = 2$ down to a 0.23–0.26 plateau by $K = 8$; 14B: 0.53 down to a 0.23–0.27 plateau) while behavioral accuracy is at floor (0.00–0.04) at every K including $K = 2$ — the model's hidden states retain decodable per-position state information throughout the depth sweep even though no CoT-based condition ever produces a correct final answer for this form (Section 5.2, Section 5.6). This is the clearest representational/ behavioral dissociation in the current pilot, precisely because the gap does not close with K the way register's does. For **modular**, there is essentially no sustained gap: probe accuracy is modestly above floor at $K = 2$ for two of the three models (0.20 at 0.5B, 0.28 at 14B; 3B is already at 0.00) and collapses to track behavioral accuracy, itself at floor from $K = 2$, by $K = 8$–16 for all three — consistent with modular's failure being present from the shallowest depths tested rather than a gradually eroding but initially intact trackable state, though Section 5.3 shows the mechanism behind that early floor differs by size: a genuine compliance collapse only for 0.5B, and a persistent arithmetic failure despite full compliance for 3B and 14B.

One data-quality caveat: 0.5B's register probe curve is noisy and incomplete at higher K — K=32 is missing entirely (too few format-compliant completions to fit a reliable probe) and the fitted values are non-monotonic (0.78 at K=16, 0.88 at K=64, then 0.33 at K=128), unlike the smooth monotonic decay 3B and 14B show for the same form. We report it as measured rather than smoothing it, but would not lean on that single model's high-K register values in isolation.

A.5 Setup: full implementation-bug disclosure

All models run locally via Hugging Face `transformers` on 4×NVIDIA RTX 2080 Ti (11GB) GPUs. Four implementation details are worth disclosing in full, the first two because they affect wall-clock and generation batch size but not the reported numbers (decoding stays greedy and deterministic throughout), the third and fourth because they *did* change reported numbers and are disclosed as corrections, not footnotes: (i) at $K \geq 64$, prompt length plus the generation budget makes full-batch (8-way) generation exceed GPU memory, so batch size is reduced to 2 ($K \geq 64$) or 1 ($K \geq 128$ or generation budget $\geq$3000 tokens); (ii) models are pinned to a single GPU via `device_map` rather than the default multi-GPU balanced placement, since the latter forces a cross-device pipeline hop on every generated token even for models that fit on one card; (iii) the tool-augmented condition's generation budget was initially a fixed 512 tokens regardless of K, unlike the CoT conditions' budgets, which scale with K. This silently truncated otherwise-correct code at higher K for models that write a per-step unrolled update rather than a loop (observed directly: Qwen2.5-3B scored 0.99 on `modular` K=16 tool-augmented with correct, complete code, then 0.00 at K=32 with the identical correct update pattern cut off mid-generation). The budget now scales with K the same way the CoT conditions' does ($15 \times K + 200$ tokens, capped at 4096); all tool-augmented cells reported in Section 5.6 were regenerated under the corrected budget; (iv) a separate bug affected the input side rather than the generation budget: `batched_generate`'s tokenizer call truncated the *prompt* to 4096 tokens (`transformers`' default right-side truncation), which is invisible unless a prompt actually exceeds that length. Exactly one (form, K) cell does, across every condition and model size: `permutation` at K=128, whose rendered prompts run 4,528–4,775 tokens depending on condition, discarding the trailing ∼400–700 tokens of every prompt in that cell — for the tool-augmented condition specifically, this discarded the instruction to write code itself, compounded by that instruction appearing only once, before rather than after the (here, long) data block. The combination produced completions that echoed or hallucinated a continuation of the truncated input rather than attempting the task, which read as a capability collapse but was a harness artifact. Fixed by raising the truncation cap to 8192 tokens with an assertion against future silent truncation, and by repeating the tool-augmented instruction after the data block; all 12 affected cells (`permutation` K=128, 4 conditions × 3 model sizes) were regenerated under the fix and are the numbers reported in Section 5. No other (form, K) cell in the pilot exceeds the corrected cap.

A.6 Tool-augmented condition: full per-model breakdown

At 0.5B, the three forms occupy three qualitatively different points of failure. For **modular**, the model essentially always writes a code block (1.00) that almost always runs (0.86) — the tool-use behavior itself is intact — but the generated code is frequently wrong, especially at higher K: inspecting completions shows the model switching from a correct sequential update (`s = (s`

`* a + b) % m`, repeated per step) to an incorrect closed-form sum of independent terms as K grows, i.e. the same difficulty in tracking sequential state shows up in the code it writes, not only in its direct-answer reasoning. For **register**, both the code-writing and execution rates roughly halve (0.51, 0.22) relative to modular, suggesting the model is markedly less reliable at translating this task form into code in the first place. For **permutation**, the model writes a code block only some of the time (0.33 averaged over K, but sharply non-monotonic: 0.92 at $K = 2$, dropping to 0% at $K = 8$–16, then rising again to 0.89 at $K = 128$) and at low-to-moderate K it *never* produces a parseable result: manual inspection shows that although the prompt explicitly instructs storing the answer in a variable named `result`, the model instead writes code that chains string `.replace()` calls on a comma-joined order string, which does not correctly compose a permutation and never assigns `result`. At $K = 128$ specifically the failure mode is different: 89% of completions now execute to a result, but every one of those hardcodes the untouched identity ordering `"0,1,2,3,4,5"` as `result` rather than attempting composition at all — a give-up pattern, not the `.replace()` misconception seen at lower K, and one that happens to be trivially executable, which is why the execution rate rebounds at exactly the K where the model has most clearly stopped trying.

At 3B, tool-use mechanics improve sharply across the board (code present and executing 81–100% of the time on every form, vs. 20–86% at 0.5B), but final accuracy tracks that improvement unevenly. **Register** now executes essentially every time (1.00) but final accuracy (0.22) lags execution rate by far more than **modular** does (0.81 executes vs. 0.57 correct) — the code runs, it is just often computing the wrong thing, consistent with Section 5.2's finding that register's failure is substantive (wrong arithmetic) rather than a refusal to engage. **Modular**'s headline result (Section 5.2) is the 99%-then-68% arc from K=16 to K=128; classifying the wrong completions at K=128 specifically (32 of 100) shows three distinct mechanisms in roughly equal measure, not one: 16 abandon the unrolled per-step code entirely and hardcode a guessed answer (e.g. `result = str(34)`, the same give-up pattern seen in forced-full-trace's format collapse, now appearing in generated code); 13 write an otherwise-correct unrolled update but *omit* some of the 128 (a_i, b_i) pairs when transcribing them from the prompt into code (e.g. 125 update lines instead of 128), so the arithmetic itself is fine but the trajectory silently diverges from the point of omission onward — a copying-fidelity failure that is specific to long inputs and distinct from both the give-up pattern and from 0.5B's wrong-closed-form pattern; and a residual 3 completions transcribe all 128 pairs correctly and still land on the wrong final value. That the dominant failure mode changes with K *and* with model size — wrong algorithm at 0.5B, give-up or faithful-transcription-failure at 3B — is itself evidence that "tool-augmented accuracy" is not one mechanism to be explained by a single theory term, and we report the breakdown rather than only the top-line number for exactly this reason. For **permutation**, the model now reliably formulates the task as executable code (1.00 has-code, 0.99 executes, up from 0.5B's 0.33/0.13, averaged over K) but is still correct only 1% of the time: the tool-use failure that dominated at 0.5B (not even trying to write parseable code, at least at low-to-moderate K) is gone at 3B, and a pure algorithmic failure — fluent, executable, consistently wrong permutation-composition logic — dominates instead, with the same identity-ordering give-up pattern documented above for 0.5B's K=128 cell recurring here too (64 of 100 completions at K=128 hardcode `"0,1,2,3,4,5"`). This is strong evidence at 3B against reading permutation's floor as mere task miscalibration (Section 5.2): the model is demonstrably engaging with the task representation, and still cannot solve it. This is a failure that precedes state-tracking entirely for the 0.5B model, and looks more like a genuine algorithm-formulation gap for the 3B model — either way, it is reported here as a form-specific limitation of the tool-augmented condition rather than evidence about $D_{\text{task}}/D_{\text{realized}}$ directly.

At 14B, this algorithm-formulation gap closes outright: has-code and executes both reach 1.00, and final accuracy jumps to 0.93 averaged over K (1.00 for $K \leq 16$) — the sharpest break from the 0.5B/3B pattern anywhere in Table 3. Section 5.2 discusses what this resolves for the permutation-floor question; the residual K=128 shortfall traces to a transcription-fidelity failure — copying 128 six-element permutations from prompt into code — rather than a return of the algorithmic failure seen at smaller sizes. **Modular** and **register** at 14B, by contrast, show scale improving tool-use mechanics further (1.00/1.00 and 0.98/0.97 has-code/executes) without a comparably clean improvement in final accuracy: modular's average (0.67) is non-monotonic across K rather than the clean 99%-then-68% arc seen at 3B, and register's (0.36) decays roughly gracefully from 0.98 at $K = 2$ to floor by $K = 128$. Why tool-augmented code-generation gains convert so much more completely into accuracy gains for permutation than for modular or register at this scale is not yet explained by the current pilot and is worth a targeted follow-up rather than speculation here.