

Telegraph Reasoning: Lintable Traces for Mechanically Verified Chain-of-Thought

Anonymous ACL submission

Abstract

Reasoning models produce long chain-of-thought traces. When the final answer is wrong, the reader cannot easily point at the exact step that broke. Asking another large language model to find the error works in part, but the model disagrees with itself across reruns and misses certain error types entirely. We introduce Telegraph Reasoning, a discrete grammar for reasoning traces that lets a small rule-based linter and a symbolic algebra system check every step. The grammar has seven tags. The linter has eleven rules; four semantic rules use SymPy to check variable scope, units, numeric equations, and model-emitted verification claims. On a corpus of 266 traces with injected errors, the linter catches 99.5 percent of the errors. A self-verifying language model catches 66.3 percent. A frontier judge model catches 87.2 percent. The linter takes nine milliseconds per trace, returns the same answer on every run, and uses no language model at verification time.

1 Introduction

Reasoning models accumulate long chain-of-thought traces. The literature has made these traces shorter (RL-trained efficient reasoning, length-compression heuristics) and made them denser (latent-state compression, dense memento tokens). Neither line of work makes the traces *checkable*. When the model gives a wrong answer, the reader cannot run a unit test on the trace to find the broken step.

The closest existing approach is to ask a second language model to verify each step (Ling et al., 2023). The verifier reads the trace, comments on each line, and votes on whether the trace is consistent. This works on easy errors but breaks down on three categories that come up often in math: a variable used before it is defined, a unit dropped or changed mid-derivation, and two equations that bind the same name to different values.

These are exactly the categories where symbolic re-execution would settle the question deterministically. The verifier is also non-deterministic in practice: reruns at temperature zero disagree on roughly ten percent of traces.

We introduce **Telegraph Reasoning** (TE), a discrete grammar for reasoning traces designed for mechanical verification. TE traces are sequences of tagged lines. Seven tags suffice for arithmetic, algebraic, and unit-dimensional reasoning. An 11-rule linter checks structural and semantic properties of the trace; the load-bearing rule reruns every *check* claim in the trace through a SymPy verifier. The verification is deterministic by construction and runs in single-digit milliseconds.

Contributions.

1. A discrete grammar (TE_REASONING_V1) for LLM reasoning traces, with an 11-rule linter and a SymPy-based symbolic verifier (§3).
2. An empirical demonstration that Claude Sonnet 4.6 emits TE traces at 100% structural compliance on GSM8K and 96.4% on MATH500 with prompt-only 5-shot in-context learning, at parity accuracy with strong NL chain-of-thought baselines on simple math (§5.1).
3. The paper’s headline result: on a 266-trace adversarial-error corpus, the linter plus SymPy verifier catches **99.5%** of injected errors. The closest LLM-driven baseline catches **66.3%**; a frontier LLM judge catches **87.2%**. The linter wins by 33 percentage points and 12 percentage points respectively, with sub-10-millisecond latency (§5.2).

The contribution is a substrate, not a competitor. RL-trained efficiency methods make traces

shorter; TE makes traces checkable. The two are orthogonal and combine naturally.

2 Related Work

Discrete structured verification. Natural Program (Ling et al., 2023) is the closest precedent: a natural-language deductive format with LLM-driven step-by-step self-verification. We share its structure-then-verify mindset but make the structure formal (a seven-tag grammar) and the verifier deterministic (an 11-rule linter plus SymPy). Natural Program cannot be unit-tested; the linter can. TraceGuard (Guo et al., 2026) also imposes structure on reasoning traces but for backdoor detection, not correctness.

Compressed and latent reasoning state. A separate family compresses reasoning into dense or continuous representations: Coconut (Hao et al., 2024) reasons in continuous hidden state; CCoT (Cheng and Van Durme, 2024) uses dense contemplation tokens; SPOT (Chu et al., 2026) decodes latent pause tokens via a frozen LM head into readable keywords; MEMENTO (Kontonis et al., 2026) compresses reasoning blocks into dense mementos that share a hidden KV channel. None of these methods produces a reasoning state amenable to mechanical verification: SymPy cannot verify a keyword soup, and a hidden KV channel cannot be linted. TE’s design choice is the prerequisite for the deterministic auditability we measure.

RL-trained efficiency. BCR (Yang et al., 2026) trains models to solve multiple problems in a shared context window with implicit budget pressure, reducing per-problem token usage by up to 62% while preserving accuracy. Other length-compression methods (Wei et al., 2026; Bian et al., 2026; Hu et al., 2026; Cao et al., 2026) achieve similar reductions through RL with explicit length penalties. None of these methods makes the reasoning state itself verifiable. TE is orthogonal: a TE-format model trained with BCR-style multi-problem batching is a natural follow-up, and we list it as future work in §7.

3 Telegraph Reasoning

A TE trace is a sequence of tagged lines. Seven tags cover arithmetic, algebraic, and unit-dimensional reasoning: GIVEN (problem inputs), GOAL (target), STEP (one-line natural-language

justification, optionally citing prior IDs), EQ[ID] (a SymPy-parseable equation), CHECK[ID] (a verification claim of one of five kinds: *arith*, *unit*, *domain*, *consistency*, or *bound*), OPEN/RESOLVE (case-split markers), and ANS (the final answer). The grammar is deliberately small. The full BNF is in Appendix A.

3.1 The linter

The linter has eleven rules in two layers. Each rule emits a stable error code (TE_E001–TE_E012) on violation; TE_E009 is reserved (see below). The seven *structural* rules check trace shape and need only the parse tree:

- **Rule 1 (TE_E001):** *given*, *goal*, and *ans* appear in order.
- **Rule 2 (TE_E002):** all IDs across *eq*, *check*, *open* are unique.
- **Rule 3 (TE_E003):** every *from* reference points to a previously defined ID (closure).
- **Rule 8 (TE_E008):** no unresolved *open* at *ans*.
- **Rule 10 (TE_E010):** every line begins with a recognized tag (no untagged text).
- **Rule 11 (TE_E011):** *ans* is derivable from the *eq* chain.
- **Rule 12 (TE_E012):** no NL hedging tokens (*maybe*, *actually*, *wait*) inside tagged lines.

Rule 9 (TE_E009) is reserved for v2 dimensional analysis; the error-code numbering is kept stable for forward compatibility.

The four *semantic* rules require SymPy:

- **Rule 4 (variable scoping, TE_E004):** every free symbol in *eq*, *check*, or *ans* must be declared in *given* or be the LHS of an earlier *eq*.
- **Rule 5 (unit consistency, TE_E005):** an *eq*’s declared unit must match the unit of its RHS, computed from constituent symbols’ given units.
- **Rule 6 (numeric correctness, TE_E006):** when an *eq*’s RHS resolves to a numeric value via the *given* bindings and prior *eq* chain, the linter substitutes and evaluates and asserts equality with the declared LHS within $\epsilon = 10^{-6}$.

173 • **Rule 7 (CHECK independence, TE_E007):**
 174 the load-bearing rule. For every
 175 `check[id]` the linter recomputes the
 176 asserted property directly from the trace
 177 AST — arithmetic equality, unit literal,
 178 domain predicate, cross-eq consistency, or
 179 bound clause — and flags any disagreement
 180 between the model’s claim and the linter’s
 181 recomputation.

182 Rule 7 is what distinguishes TE from prior
 183 structured-CoT formats such as Natural Program:
 184 rather than asking an LLM to self-verify each
 185 step, the linter independently re-executes every
 186 claim the model makes. Rule 6 is the empirical
 187 workhorse on the corpus we evaluate (§5.3); rule
 188 7 is the conceptual distinction that makes verifica-
 189 tion possible at all.

190 3.2 Worked example

191 Consider a trace that solves “How many eggs does
 192 Janet sell at the market?” with input 16 eggs total,
 193 3 eaten, 4 baked, sold at \$2 each:

```
194 GIVEN:
195     total := 16
196     eaten := 3
197     baked := 4
198     price := 2 [USD]
199 GOAL: revenue [USD]
200 STEP[s1]: subtract used eggs
201 EQ[e1]: sold = total - eaten - baked
202 CHECK[c1]: arith: e1.value == 9
203 EQ[e2]: revenue = sold * price
204 CHECK[c2]: arith: e2.value == 18
205 ANS: 18 [USD]
```

206 The linter parses each line, builds the scope
 207 from given, and for `check[c1]` re-runs
 208 `total - eaten - baked = 16 - 3 - 4 = 9`,
 209 agreeing with the asserted 9. For `check[c2]` it
 210 re-runs `sold · price = 9 · 2 = 18`. If a model
 211 had instead written `CHECK[c1]: arith:`
 212 `e1.value == 8` (an arithmetic slip), the linter
 213 would compute 9, disagree, and emit TE_E007 at
 214 line 9 with the offending values. A self-verifying
 215 LLM, by contrast, must *notice* that $16 - 3 - 4 \neq 8$
 216 from prose alone — which it does most of the time
 217 but not always.

218 3.3 SymPy verifier

219 SymPy parsing operates under a restricted
 220 whitelist of 18 functions and four constants (π ,
 221 e , i , ∞). Identifiers not in the whitelist auto-
 222 register as user symbols, preventing SymPy global-
 223 namespace shadowing of user variable names. A

two-second per-call timeout on `simplify` pre- 224
 vents pathological slowdowns; we observed no 225
 timeouts across the entire 266-trace evaluation 226
 corpus. Numeric tolerance is $\varepsilon = 10^{-6}$. The full 227
 whitelist and parser interface are in Appendix B. 228

4 Experimental Setup 229

We evaluate all conditions on Claude Sonnet 4.6 230
 via AWS Bedrock at temperature 0.2 for gener- 231
 ation, 0.0 for verification. Benchmarks are the 232
 full GSM8K test split ($n = 1,319$) and the full 233
 MATH500 test split ($n = 500$). Maximum gener- 234
 ation tokens are 1,024 for GSM8K and 2,048 for 235
 MATH500. 236

E1 (prompt-only accuracy). Four conditions: 237
R5 (TE scratchpad with 5 in-context examples), 238
R1 (standard chain-of-thought, no shots), **R2** (con- 239
 cise CoT, no shots), **R3** (JSON scratchpad, no 240
 shots). The asymmetric prompting gives base- 241
 lines a $\sim 1,000$ -token prompt-budget advantage. 242
 On GSM8K, the non-inferiority claim survives 243
 this conservative design; on MATH500 we re- 244
 port the resulting deficit rather than claiming non- 245
 inferiority. 246

E5 (auditability). A 266-trace adversarial corpus 247
 comprising 30 hand-crafted seeds, 166 pro- 248
 grammatic perturbations of correct R5 traces, 50 249
 random clean controls, and 20 hard-clean controls. 250
 Each adversarial trace contains exactly one in- 251
 jected error from one of five categories: arithmetic 252
 slip, lost variable, unsupported conclusion, wrong 253
 unit, contradicting equation. We compare four 254
 verification methods: **M1** (TE structural-only lin- 255
 ter), **M2** (TE + SymPy linter, full rules 1–12), **M3** 256
 (Natural Program self-verify on deterministically- 257
 translated traces), **M4** (LLM judge: Sonnet 4.6 258
 prompted to detect inconsistencies). M3 and M4 259
 run three times per trace at temperature 0.0 with 260
 majority vote; M1 and M2 are deterministic by 261
 construction. 262

Statistics. 95% paired-bootstrap CIs (10,000 263
 iterations); McNemar’s χ^2 with continuity correc- 264
 tion; Holm–Bonferroni correction at family-wise 265
 $\alpha = 0.05$. Power exceeds 0.99 under the observed 266
 effect. 267

5 Results 268

5.1 E1: prompt-only accuracy 269

Table 1 reports per-condition accuracy on both 270
 benchmarks. 271

Cond.	GSM8K	MATH500
R5 (TE)	96.4%	62.6%
R1 (Std CoT)	97.0%	68.6%
R2 (Concise CoT)	96.8%	68.4%
R3 (JSON)	97.0%	67.2%

Table 1: Prompt-only accuracy on the full GSM8K ($n = 1,319$) and MATH500 ($n = 500$) test splits. TE matches strong NL baselines on GSM8K and trails by ~ 5 –6 pp on MATH500. Tokens-per-correct and per-difficulty breakdowns are in Appendix D.

On GSM8K, TE matches the best NL baseline within -0.4 pp; the 95% CI clears the pre-registered non-inferiority margin by a wide margin. On MATH500, TE trails concise CoT by -5.80 pp (95% CI $[-8.20, -3.40]$). Our pre-registration designated this range as a FLAG band — a real deficit, not a gate-passing result — triggering planned SFT remediation rather than a non-inferiority claim. The deficit is uniform across difficulty levels rather than concentrated on hard problems. We characterize it as constant structural overhead from the TE scaffolding (GIVEN, CHECK, etc.), proportionally heavier on shorter problems. Structural compliance is 100% on GSM8K and 96.4% on MATH500 (structural-failure breakdown in Appendix D).

Failure-mode triage. Of 187 wrong R5 traces on MATH500: **77.5%** (145/187) are strictly difficulty-bound (R5 wrong *and* all three NL baselines wrong on the same problem; paired comparison); 13.4% (25/187) are TE-specific (TE wrong, all three baselines correct), of which roughly half are LaTeX-vs-SymPy notation mismatches recoverable with deeper extractor normalization; 6.4% (12/187) are extraction errors (no `ans` emitted before `MAX_TOKENS`); the remaining 2.7% (5/187) are real grammar slips. The ~ 5 pp MATH500 gap reflects the harder shared problem set plus the recoverable notation mismatches, not a fundamental TE-specific failure mode.

Per-token efficiency. R2 (concise CoT) dominates the per-token Pareto on both benchmarks: 55 / 148 tokens-per-correct (GSM8K / MATH500) versus TE’s 166 / 342. TE does not contribute on the efficiency axis. The contribution defended in this paper is *auditability* (§5.2).

E1 establishes that TE traces are emittable at frontier scale; the contribution is the auditability comparison in §5.2.

Method	TPR	FPR	Loc. ± 3	κ
M1 (struct. only)	0.0%	0.0%	0.0%	1.00
M2 (TE+SymPy)	99.5%	10.0%	90.3%	1.00
M3 (NP self-verify)	66.3%	2.9%	84.6%	0.89
M4 (LLM judge)	87.2%	5.7%	97.7%	0.93

Table 2: Auditability methods on the 266-trace adversarial corpus (196 adversarial, 70 clean). M2 is TE plus the full SymPy linter. M3 is Natural Program self-verify on deterministically-translated traces. M4 is a frontier LLM (Sonnet 4.6) prompted to detect inconsistencies in raw TE traces. M2’s 10% raw FPR drops to 7.1% after reclassifying two true catches that the gold-vs-extracted comparison missed; details in Appendix F.

Category	n	M2	M3	M4
arithmetic slip	46	100%	84.8%	84.8%
lost variable	46	100%	52.2%	69.6%
unsupported conclusion	46	100%	91.3%	95.7%
wrong unit	12	91.7%	16.7%	83.3%
contradicting equation	46	100%	50.0%	100%

Table 3: Per-category TPR. M2 catches every case in four of five categories and 11 of 12 wrong-unit cases. M3 (NP self-verify) collapses on the categories where SymPy does work that language-model judgment cannot do reliably.

5.2 E5: auditability

Table 2 reports the headline auditability comparison.

The linter wins by 33 pp. M2 catches 195 of 196 injected errors (Wilson 95% CI $[97.2\%, 99.9\%]$). M3 catches 66.3%; M4 catches 87.2%. The M2-vs-M3 paired-bootstrap difference is $+33.09$ pp (95% CI $[+26.5, +39.8]$); M2-vs-M4 is $+12.24$ pp ($[+7.7, +17.4]$). Both differences are significant after Holm–Bonferroni correction (adjusted $p < 10^{-4}$). We report the raw 10.0% FPR in Table 2. Two of the seven clean-control flags correspond to line-verifiable internal inconsistencies despite final-answer agreement with gold; under this stricter audit interpretation, effective FPR is 7.1% (Appendix F). Of the five strict-FPR cases, three are rule-5 parser failures on parametric problems where a free parameter is intentionally left undeclared; each has a documented v2 fix.

5.3 Where the linter wins: the mechanism

The aggregate TPR conceals a sharp per-category structure that is the paper’s main mechanism argument. Table 3 breaks TPR by corruption category.

The categories where M3 fails most — *lost variable* (52.2%), *wrong unit* (16.7%), and *con-*

337 *tradiating equation* (50.0%) — are precisely the
338 categories where SymPy does work that natural-
339 language verification cannot do reliably. Tracking
340 that a symbol used in an `eq` was never declared
341 in `given` requires a name-resolution pass over
342 the trace. Checking that two `eq` lines bind the
343 same variable to inconsistent values requires sym-
344 bolic substitution and simplification. Checking
345 that an `eq`’s declared unit matches the dimension-
346 ality of its right-hand side requires a unit registry.
347 M3 (and M4 less severely) approximates each of
348 these through language-model judgment, and the
349 approximation breaks down on the hardest cases.
350 M2 simply runs the underlying computation. The
351 single most dramatic gap is *wrong unit*, where
352 M2 catches 11 of 12 cases and M3 catches 2 of
353 12 — a 75 percentage-point gap on a small but
354 well-defined category.

355 **Per-rule ablation.** The mechanism story has a
356 sharper form: *which rules carry the load?* We run
357 four independent ablations, each disabling one se-
358 mantic rule and leaving the other three (and all
359 seven structural rules) intact. Rule 4 (variable
360 scoping): TPR drops by 0.5 pp. Rule 5 (unit
361 consistency): drops by 5.6 pp. Rule 6 (numeric
362 correctness on `eq`): drops by **21.4 pp** (TPR falls
363 from 99.5% to 78.1%). Rule 7 (CHECK indepen-
364 dence): drops by 2.6 pp. Rule 4’s small drop is
365 consistent with the data: lost-variable adversarials
366 also fail rule 6 (an undeclared symbol makes the
367 substitute-and- evaluate pass abort with a name-
368 resolution error), so rule 4 is mostly redundant
369 with rule 6 on this corpus. Rule 7 is the *con-*
370 *ceptual* distinction from LLM self-verification —
371 it re-executes every `check` claim from the AST
372 rather than asking the model to re-check itself —
373 but rule 6 carries the largest *empirical* load: 42
374 of 196 adversarials (21.4%) are detected solely by
375 rule 6’s substitute-and-evaluate pass, regardless of
376 whether the model emitted a `check` line at all.

377 **Two concrete catches.** (a) *Arithmetic slip (rule*
378 *6, the modal case).* A trace declares `EQ[e3]:`
379 `total = 16-3-4 = 8` (the model added
380 wrong), then writes `CHECK[c3]: arith:`
381 `e3.value == 8`. The `check` agrees with the
382 slipped value, so M3 (reading prose) signs off:
383 “compute 16 minus 3 minus 4 equals 8. Verify:
384 yes.” M2 evaluates rule 6 on `e3`: substitute and
385 re-evaluate $16 - 3 - 4 = 9 \neq 8$; `TE_E006` fires.
386 (b) *Wrong unit (rule 5).* `EQ[e2]: distance`
387 `= speed * time [miles] with speed in`

`m/s and time in seconds.` M3 reads the trace
and signs off (the LLM “knows” the formula).
M2 composes the RHS unit as $\text{m/s} \cdot \text{s} = \text{meters}$,
disagrees with the declared `miles`, and fires
`TE_E005`. Rules 6 and 7 catch the same family
of errors via different paths: rule 6 catches a slip
even when the model’s own `check` agrees with
the slipped value; rule 7 catches a slip when the
model’s `check` disagrees with the trace’s actual
arithmetic. On this corpus they overlap on $\sim 45\%$
of detected adversarials, but rule 6 is uniquely
responsible for 21.4% and rule 7 for 2.6%.

Generalization caveat. The corpus is synthetic
and the categories were chosen to span TE’s rule
taxonomy. We do not claim that the 99.5% num-
ber transfers verbatim to naturally-occurring LLM
failures whose error distribution may differ. The
mechanism story is what we expect to general-
ize: rule 6 catches arithmetic mismatches that the
model’s own self-verification cannot see, and that
property is independent of corpus distribution.

Determinism. M2 is verdict-deterministic by
construction: the same trace produces the same
flag set, the same issue codes, and the same line
numbers on every run. M3 and M4 do not have
this property even at temperature 0.0. Across three
independent runs of M3 on the 266 traces, the ver-
dict (FLAG vs. CLEAN) flipped on roughly 10% of
traces; for M4 the disagreement rate was roughly
7%. For audit pipelines that route flagged traces
to specific remediation routines, this is the opera-
tional difference between a unit-testable verifier
and a non-deterministic one.

Latency. Median 9 ms per trace, 95th percentile
81 ms, maximum 703 ms on a 1,500-token trace
with deep symbolic manipulation. Linting the full
266-trace corpus end-to-end takes under 3 seconds
on a single CPU. M3 and M4 require ~ 25 minutes
of wall-clock time per method (1,596 Bedrock
calls each) plus retries on the 6.4% of M3 calls
that return malformed JSON.

6 Limitations

1. **Synthetic adversarial corpus.** The 266-
trace corpus is built around five corruption
categories matched to TE’s rule taxonomy.
We do not claim that the 99.5% TPR gener-
alizes to naturally-occurring LLM failures
whose error distribution may differ. The
mechanism story (rule 6 catches arithmetic

mismatches the model’s self-verification cannot see) is independent of corpus distribution and we expect it to generalize; cross-domain stress tests are future work.

2. **TE prompt-only does not match NL chain-of-thought on harder math.** On MATH500 the gap is -5.80 pp (95% CI lower bound -8.20 pp), within the pre-registered FLAG band but a real deficit. Mitigation: SFT on the curated TE pool; matched- scale 2B-model SFT preliminary results are in Appendix I.

3. **Single-task math evaluation.** The 266-trace E5 corpus is math-only. Generalization to logic, code, and multi-trace settings is not tested.

4. **Single-model generation and judging.** All TE traces and all LLM-judge baselines (M3, M4) use Claude Sonnet 4.6. Cross-model verification is future work.

5. **Grammar v1 has known scope gaps.** Composite additive-unit expressions (e.g., $m \cdot (m + m)$), brute-force case enumeration, and quantifier-rich logic require v2 grammar extensions. M2 misses 1 of 12 wrong-unit cases for this reason.

6. **Asymmetric prompting.** TE uses 5-shot in-context examples; baselines use zero shots. The asymmetric design is the conservative direction for the non-inferiority claim (TE carries shot-tax overhead while baselines do not). Matched-shot baselines are a documented follow-up.

7 Conclusion

Telegraph Reasoning makes reasoning traces mechanically checkable. A small grammar plus a deterministic linter plus a SymPy verifier catches 99.5% of injected errors at sub-10-ms latency, beating LLM-driven self-verification by 33 percentage points. The contribution is a substrate, not a competitor: RL-trained efficiency methods make traces shorter; TE makes them checkable. The two are orthogonal and combine naturally in future work.

References

- Tingcheng Bian, Jinchang Luo, Mingquan Cheng, Jinyu Zhang, Xiaoling Xia, Ni Li, Yan Tao, and Haiwei Wang. 2026. TRiMS: Real-time tracking of minimal sufficient length for efficient reasoning via RL. *arXiv preprint arXiv:2603.17449*.
- Jie Cao, Tianwei Lin, Zhenxuan Fan, Bo Yuan, Ziyuan Zhao, Rolan Yan, Wenqiao Zhang, and Siliang Tang. 2026. Draft-thinking: Learning efficient reasoning in long chain-of-thought LLMs. *arXiv preprint arXiv:2603.00578*.
- Jeffrey Cheng and Benjamin Van Durme. 2024. Compressed chain of thought: Efficient reasoning through dense representations. *arXiv preprint arXiv:2412.13171*.
- Yunlong Chu, Minglai Shao, Yuhang Liu, Bing Hao, Yumeng Lin, Jialu Wang, and Ruijie Wang. 2026. SPOT: Span-level pause-of-thought for efficient and interpretable latent reasoning in large language models. *arXiv preprint arXiv:2603.06222*.
- Zhen Guo, Shanghao Shi, Hao Li, Shamim Yazdani, Ning Zhang, and Reza Tourani. 2026. TraceGuard: Process-guided firewall against reasoning backdoors in large language models. *arXiv preprint arXiv:2603.02436*.
- Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*.
- Chenzhi Hu, Qinzhe Hu, Yuhang Xu, Junyi Chen, Ruijie Wang, Shengzhong Liu, Jianxin Li, and Fan Wu. 2026. SmartThinker: Progressive chain-of-thought length calibration for efficient large language model reasoning. *arXiv preprint arXiv:2603.08000*.
- Vasilis Kontonis, Yuchen Zeng, Shivam Garg, Lingjiao Chen, Hao Tang, Ziyang Wang, Ahmed Awadallah, and Eric Horvitz. 2026. MEMENTO: Teaching LLMs to manage their own context. *arXiv preprint arXiv:2604.09852*.
- Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. 2023. Deductive verification of chain-of-thought reasoning. In *Advances in Neural Information Processing Systems*, volume 36.
- Songtao Wei, Yi Li, Zhikai Li, Xu Hu, Yuede Ji, Guanpeng Li, Feng Chen, and Carl Yang. 2026. LEAD: Length-efficient adaptive and dynamic reasoning for large language models. *arXiv preprint arXiv:2605.09806*.
- Bangji Yang, Hongbo Ma, Jiajun Fan, and Ge Liu. 2026. Batched contextual reinforcement: A task-scaling law for efficient reasoning. *arXiv preprint arXiv:2604.02322*.

```

trace    ::= header body ans
header   ::= 'GIVEN:' (decl)+ 'GOAL:' expr
body     ::= line+
line     ::= step | eq | check
          | open | resolve
step     ::= 'STEP[' id ':' text
          ('from' id (' ' id)*)?
eq       ::= 'EQ[' id ':' lhs '=' rhs
          ('[' unit ''])?
check    ::= 'CHECK[' id ':' kind ':' claim
kind     ::= 'arith' | 'unit' | 'domain'
          | 'consistency' | 'bound'
open     ::= 'OPEN[' id ':' subgoal
resolve  ::= 'RESOLVE[' id ':' 'by' id
ans      ::= 'ANS:' expr ('[' unit ''])?
decl     ::= name ':' value ('[' unit ''])?

```

Figure 1: BNF for TE_{v1}. Whitespace and indentation are significant only at the GIVEN block. Identifiers are [a-zA-Z_] [a-zA-Z0-9_]*.

A Grammar (TE_REASONING_V1)

The full BNF is shown in Figure 1. The grammar allows (and the linter rejects) deviations from the basic shape described in §3.

B SymPy verifier

The SymPy verifier accepts a restricted vocabulary: **functions** `sqrt`, `abs`, `min`, `max`, `floor`, `ceil`, `gcd`, `lcm`, `mod`, `log`, `exp`, `sin`, `cos`, `tan`, `Sum`, `Product`, `factorial`, `binomial`; **constants** π , e , i , ∞ . The whitelist was selected by inspection of GSM8K and MATH500: every function and constant that appears in any gold-derivation step on either benchmark is included, and nothing else. Identifiers not in the whitelist auto-register as user symbols. A 2-second per-call timeout on `simplify` prevents pathological slowdowns; we observed zero timeouts across all 266 traces. Numeric tolerance is $\epsilon_{\text{rel}} = 10^{-6}$, $\epsilon_{\text{abs}} = 10^{-9}$. Rational arithmetic is preserved end-to-end via `nsimplify(rational=True)`.

C Linter specification

The linter has eleven active rules; each emits a stable error code (TE_E001–TE_E012) on violation, with TE_E009 reserved as documented below.

Rule 1 (TE_E001, structural). Trace must contain GIVEN, GOAL, and ANS in order.

Rule 2 (TE_E002). All identifiers across eq, check, open must be unique.

Rule 3 (TE_E003). Reference closure: every from id must point to a previously defined ID.

Rule 4 (TE_E004, semantic). Variable scoping: every free symbol must be declared in given or be the LHS of an earlier eq.

Rule 5 (TE_E005). Unit consistency: an eq’s declared unit must agree with the unit computed from constituents.

Rule 6 (TE_E006). Numeric correctness: an eq’s RHS, evaluated under the scope chain, must equal the declared LHS within 10^{-6} .

Rule 7 (TE_E007, load-bearing). CHECK independence: every check[id]’s asserted property must agree with the linter’s recomputation from the trace AST.

Rule 8 (TE_E008). No unresolved open at ans.

Rule 10 (TE_E010). Tag-only-prefix discipline: each line must begin with a recognized tag.

Rule 11 (TE_E011). ANS must be derivable from the eq chain.

Rule 12 (TE_E012). No natural-language hedging tokens (*maybe*, *actually*, *wait*, *hmm*) inside tagged lines.

Rule 9 (TE_E009, reserved). Composite-unit dimensional analysis (additive units across polynomial expressions, e.g., $m \cdot (m + m)$); the error-code numbering is kept stable for forward compatibility. Not implemented in v1.

D E1 extended results

Tokens-per-correct. R5 (TE): 166 / 342 (GSM8K / MATH500). R1 (Std CoT): 141 / 359. R2 (Concise CoT): **55 / 148** (efficiency winner). R3 (JSON): 251 / 391. R2 dominates the per-token Pareto frontier on both benchmarks; TE does not contribute on the efficiency axis. The contribution defended in this paper is auditability (§5.2), not efficiency.

MATH500 per difficulty level. Table 4 shows accuracy by native MATH500 difficulty. The R5-vs-R2 gap ranges from -2.9 pp (level 3) to -7.8 pp (level 2), without a monotonic relationship to difficulty.

Level	R5	R1	R2	R3
1 ($n=43$)	74.4%	79.1%	81.4%	81.4%
2 ($n=90$)	72.2%	74.4%	80.0%	75.6%
3 ($n=105$)	62.9%	67.6%	65.7%	66.7%
4 ($n=128$)	61.7%	68.8%	66.4%	65.6%
5 ($n=134$)	53.0%	61.9%	60.4%	59.0%

Table 4: MATH500 accuracy by difficulty level.

Pair	ΔTPR (95% CI)	ΔFPR	McNemar p_{Holm}
M2 vs. M3	+33.09 [+26.5, +39.8]	+7.06	$< 10^{-5}$
M2 vs. M4	+12.24 [+7.7, +17.4]	+4.29	$< 10^{-4}$
M1 vs. M3	-66.45 [-73.0, -59.7]	-2.86	$< 10^{-5}$
M1 vs. M4	-87.20 [-91.8, -82.7]	-5.71	$< 10^{-5}$

Table 5: Paired comparisons; 10,000-iter paired-bootstrap on per-trace correctness. All pairs significant after Holm–Bonferroni at $\alpha = 0.05$.

Structural failures on MATH500. Of the 18 structural failures (3.6%), 14 are MAX_TOKENS=2048 truncation on hard problems where the trace ran out mid-line, and 4 are model-emitted prose between tagged lines (notably in brute-force enumeration problems where the model used STEP [sN] : to introduce a multi-line list).

Failure-mode triage. Of 187 wrong R5 traces on MATH500: **77.5%** (145/187) are strictly difficulty-bound (R5 wrong *and* all three NL baselines wrong on the same problem); **13.4%** (25/187) are TE-specific (TE wrong, all three baselines correct), of which roughly half are LaTeX-vs-SymPy notation mismatches recoverable with deeper extractor normalization; the remainder ($\sim 9\%$) are extraction errors, partial-disagreement cases, and real grammar slips.

E E5 extended results

The per-category breakdown (Table 3) and the per-rule ablation are reported in the main body (§5.3). This appendix supplies only the paired-comparison statistics not surfaced in the main text.

Paired-comparison statistics. Table 5 reports the four key paired comparisons.

F FPR diagnosis

M2’s raw FPR is 10.0% (7 of 70 clean controls flagged); this is the metric reported in Table 2. The stricter audit-interpretation FPR of 7.1% drops two flags: every flag whose accompanying issue code points to an internal inconsistency that an auditor could verify line-by-line counts as a true

TE tag	NP rendering
GIVEN	“Given:” followed by inputs
GOAL	“Find:” followed by target
STEP [id]	“Reasoning:” followed by text
EQ[id] : lhs=rhs	“Step N: lhs = rhs”
CHECK[id] : arith	“Verify:” ...
CHECK[id] : unit	“Verify (units):” ...
CHECK[id] : domain	“Verify (domain):” ...
CHECK[id] : consistency	“Verify (consistency):” ...
CHECK[id] : bound	“Verify (bound):” ...
OPEN[id]	“Sub-goal:” ...
RESOLVE[id] by r	“Resolved by step r.”
ANS	“Final answer:” ...

Table 6: Deterministic TE $\rightarrow$ NP mapping. The conversion is bijective at the tag level; no information is lost or added.

positive even if the final answer happens to coincide with the gold label. The rule is mechanical and applied without inspecting M3 or M4 verdicts.

Two audit-interpretation cases. (a) GSM8K trace whose CHECK[c4] : arith: eq_d.value == 33 asserts 33 for an eq that evaluates to $32/96 \cdot 100 = 33.\bar{3}$; the model rounded to 33 in both check and ans, matching gold after rounding, but the linter correctly flags the internal arithmetic-rounding inconsistency. (b) An ans that uses base-subscript notation 4343_6 rather than a SymPy-parseable bare numeral, which the linter flags as a format violation.

Five remaining false positives. (i) Parametric problems where a free parameter is intentionally left undeclared. (ii) eq_id.value notation in eq right-hand sides where the model reuses a prior eq’s value via shorthand. (iii) String-literal given entries used to document recurrences. (iv) Multi-step symbolic-manipulation chains where the model rebinds a variable in pedagogically meaningful ways (e.g., expanding $(x \cdot y \cdot z)^4$ as 16). (v) An unusual ans format. None of these represents a fundamental problem with the verification approach; each has a documented v2 fix.

G NP-translation mapping

The Natural Program protocol (Ling et al., 2023) operates on natural-language step-numbered solutions, not on TE traces. We use a fixed, rule-based mapping rather than asking an LLM to rewrite (which could fix or hide the very errors NP is meant to catch). Table 6 lists the full mapping.

H Adversarial corpus generation

The 266-trace corpus has four parts: (1) **30 hand-crafted seeds** with one error each from the five

677 categories (6 per category). (2) **166 program-**
678 **matic perturbations** of correct R5 traces from E1:
679 an automated perturber selects a random correct
680 trace, samples a category, and applies a category-
681 specific perturbation function (e.g., for arithmetic
682 slip: replace one $\in q$'s RHS-numeric-result with a
683 value off by 1–3 in the last digit). (3) **50 random**
684 **clean controls** sampled from correct R5 traces.
685 (4) **20 hard-clean controls** sampled from correct
686 R5 traces on MATH500 levels 4–5 (to test FPR
687 on more complex traces). The pre-registration
688 is E5_v1_2026_05_13; the full seed list, per-
689 turbation generators, and prompts are released in
690 supplementary materials.

691 **I E3: matched-scale SFT (preliminary)**

692 To clarify the contribution of TE-SFT versus the
693 base model's own reasoning at matched param-
694 eter scale, we additionally evaluate prompt-only
695 Qwen3.5-2B with the standard chain-of-thought
696 prompt on the same 1,819 problems. The SFT
697 model (R5_{sft}: Qwen3.5-2B + TE-SFT on 1,562
698 traces) achieves 72.1% on GSM8K and 36.6%
699 on MATH500 versus prompt-only Qwen3.5-2B at
700 78.9% and 48.8% ($\Delta = -6.81$ pp / -12.23 pp;
701 both significant under Holm–Bonferroni-corrected
702 McNemar at $p < 10^{-5}$; bootstrap 95% CIs
703 $[-9.55, -4.02]$ / $[-16.60, -8.00]$).

704 The SFT model compresses tokens-per-correct
705 by 64.5% / 67.7% (R5_{sft}: 167 / 268; prompt-only-
706 2B: 471 / 829) and reduces extraction-error rate
707 by 4–8 $\times$ (0.2% / 12% vs. 7.7% / 30%). Structural
708 compliance is 99.2% / 81.8%.

709 We characterize the accuracy regression as a
710 small-pool small-scale SFT limitation rather than
711 a TE-format limitation; future work scales the
712 SFT pool to 5,000–30,000 traces and the base
713 model to 4B–9B. The auditability finding (§5.2)
714 is independent of and unaffected by this matched-
715 scale tradeoff.