SEMANTIC QUOTA CONSTRAINTS FOR SPECIFICATION-FAITHFUL STRUCTURED GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Grammar-constrained generation guarantees structural validity but does not automatically guarantee quantitative specification fidelity. A well-formed program can contain the wrong number of required constructs, and a schema-valid prefix can still terminate before required members or lower bounds are satisfied. We introduce semantic quota constraints, a parameter-free decode-time mechanism that tracks quantitative requirements—counts, required operators or keys, and collection cardinalities—in parser or container state. Stop blocking prevents premature termination, quota blocking prevents exact-count overshoot, and steering improves timely completion under finite decode budgets.

In a controlled program-generation environment, grammar-only decoding reaches 0.5% All-req, validity blocking reaches 92.7%, and the full quota layer reaches 100.0%, while every terminating rollout satisfies every parsed quantitative requirement under explicit grammar assumptions. Incremental quota tracking is $O(k)$ per step; we report the classical counter-product size only as the compiled baseline this avoids.

On real structured-generation systems, enforcement is backend- and schema-dependent. A sampling-free compiler-language probe finds violating `minProperties`/`maxProperties` completions admitted by pinned Outlines 1.3.2, while llguidance 1.7.6 and XGrammar 0.2.7 block the corresponding violating continuation in the tested micro-schemas. A scoped container-state adapter repairs the observed Outlines lower-bound failures (53/60 to 57/60 matched full-schema validity; 16 repairs and 0 breaks at twenty draws) without replacing the base structural decoder. Structured validity and quantitative specification fidelity are separable, and incremental quota state is a practical mechanism for enforcing the latter.

1 INTRODUCTION

Grammar-constrained generation enforces structural validity but leaves an important class of semantic constraints unenforced. A decoder that never emits an ungrammatical continuation can still return a program with the wrong number of loops, an object missing a required field, or a payload that undershoots `minProperties`. Syntax validity is not specification fidelity.

We introduce *semantic quota constraints*, a parser-state decoding mechanism for quantitative requirements. We use the term for specification conditions that can be tracked incrementally in parser or container state—exact or lower-bound counts, required operators, required object members, and collection cardinalities—not for execution correctness, variable scoping, or arbitrary logical semantics. The layer composes with any host that exposes a grammar- or schema-valid action set: it blocks stopping while a requirement is unmet, removes actions that would overshoot an exact count, and steers toward unmet requirements. It adds no parameters and does not retrain the host. Because the intended guarantee is binary—every parsed requirement met—we report prompt-level satisfaction (All-req) and the mean per-requirement rate (Sat.) alongside.

A controlled program-generation environment lets us establish formal guarantees and isolate the mechanism. In that environment, specification satisfaction is exactly measurable, parser state is fully observable, and we can ablate each mask. Grammar-only decoding reaches 0.5% All-req; adding quota blocking—the validity filters that make every terminating output faithful—raises this to 92.7%;

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.

enabling steering, which is not required for that validity guarantee, reaches 100.0% by eliminating residual budget hits across seeds 42/43/44 (Section 5). Theorem 1(2) then says that every terminating rollout satisfies every parsed requirement. A confirmatory full-split run reaches 99.4% All-req at the standard action cap; residual misses are budget hits, not requirement overshoots (Appendix D).

Three questions structure the rest of the paper. **(RQ1) Validity:** can online parser or container-state quotas guarantee that any terminating output satisfies parsed quantitative requirements? **(RQ2) Completion:** how much does steering matter for reaching a faithful terminating output before a finite decode budget? **(RQ3) Transfer:** do real structured-generation systems exhibit backend-dependent gaps on quota-relevant constraints, and can a scoped online adapter repair the gaps that are actually present?

The same failure mode and repair arise in deployed structured-generation systems, but support is backend-dependent (Section 6): cardinality keywords appear in 691/952 SchemaStore documents, a sampling-free compiler-language probe isolates an Outlines lower-bound leak that llguidance and XGrammar block on the tested micro-schemas, and a scoped adapter repairs the measured Outlines failures. The synthetic DSL is the instrument where we can prove and ablate; JSON Schema is the transfer argument.

Contributions.

1. **Mechanism.** A parameter-free quota layer over parser or container state: stop-blocking and quota-blocking for counting validity, plus steering for timely completion (Section 3).
2. **Formal guarantee.** Terminating-output fidelity under explicit quota-safe and closure-safe grammar assumptions (Theorem 1), with a restricted termination certificate for conforming rollouts (Theorem 2).
3. **Controlled isolation.** Causal ablations separate structural validity, quota validity, and timely completion (Section 5).
4. **Transfer.** Quota-relevant constraints are common in SchemaStore; enforcement support is backend-dependent; a scoped container-state adapter repairs observed lower-bound failures without replacing the base structural decoder (Section 6).

A certified termination bound for conforming rollouts (Theorem 2) and frozen-host checks on `Qwen/Qwen2.5-0.5B-Instruct` (66.5% All-req), `Qwen/Qwen2.5-7B-Instruct` (89.0% All-req), and `HuggingFaceTB/SmolLM2-1.7B-Instruct` (28.0% All-req) are supporting results; all are scoped, and we do not treat them as headline claims. Real-language program synthesis (e.g. Spider/SQL) remains outside this submission.

2 Related Work

Grammar-constrained decoders prevent syntactically invalid continuations by masking tokens or actions that cannot extend the current prefix into a valid program. PICARD applies incremental parsing for constrained text-to-SQL, Synchromesh uses constrained semantic decoding for code-like targets, and Outlines, guidance, llguidance, and more recent engines such as XGrammar compile structure into finite-state or grammar masks (Scholak et al., 2021; Poesia et al., 2022; Willard & Louf, 2023; Microsoft Guidance Contributors, 2023; Guidance-AI, 2026; Dong et al., 2024; Koo et al., 2024). These methods guarantee syntax or structured-output validity. Whether they also enforce quantitative counts—exactly two loops, a required operator, or a nonempty `required` set—is implementation-dependent; Section 6 measures that gap rather than assuming it.

Lexically constrained decoding tracks token- or predicate-level constraints during search. Grid Beam Search and dynamic beam allocation maintain constraint-satisfaction state over beams and prune candidates that cannot meet remaining lexical obligations; NeuroLogic and its lookahead variants similarly enforce logical predicates by deterministic beam pruning, not merely by adding a soft score (Hokamp & Liu, 2017; Post & Vilar, 2018; Lu et al., 2021; 2022). Semantic quota constraints differ in representation and point of enforcement: they track quantitative properties of parser or container state and intersect those requirements with the structurally valid action set at each decode step.

In principle, count constraints can be expressed by intersecting a grammar with a counting automaton (Hopcroft et al., 2006; Willard & Louf, 2023; Koo et al., 2024). That product is classical; we do not

claim the finite counter as a contribution. Quota blocking is the same intersection evaluated online in $O(k)$ per step—the validity filters—rather than compiled into an explicit state set, and it is not a PDA product over the RNNG stack. Appendix F only measures how large that classical product would be on our prompts. The full layer additionally steers toward unmet requirements so that a faithful output is reached before a finite action cap. Broader semantic-control methods that preserve reasoning traces or apply probabilistic logical constraints (Banerjee et al., 2025; Ahmed et al., 2025) target a different obligation than incremental quotas; we do not treat them as interchangeable baselines.

3 Semantic Quota Constraints

The mechanism is independent of any one grammar. At each decode step the host proposes a structurally valid set $G(s)$. The quota layer maintains a running tally $Q(s)$ of parsed requirements against the same parser or JSON container state the structural decoder already uses. Write $M_{\text{stop}}(s)$ for the stop mask and $M_{\text{quota}}(s)$ for the exact-count overshoot mask. Validity is their intersection:

$$A_{\text{valid}}(s) = G(s) \cap M_{\text{stop}}(s) \cap M_{\text{quota}}(s).$$

The AST decoder then applies a completion policy:

$$A_{\text{full}}(s) = \begin{cases} A_{\text{valid}}(s) \cap M_{\text{prog}}(s) & \text{if an unmet reachable quota exists,} \\ A_{\text{valid}}(s) \cap M_{\text{close}}(s) & \text{if all requirements are met,} \\ A_{\text{valid}}(s) & \text{otherwise.} \end{cases}$$

Here M_{prog} is progress steering and M_{close} is post-satisfaction closure (Appendix A). An *unforced window* is a state where at least one requirement remains unmet but no valid descendant from the current expansion point can reach an unmet quota type before the current node's structural minimums; then M_{prog} leaves A_{valid} unchanged. If the layer would return an empty set, the implementation records a safety-fallback event and restores $G(s)$ for that step; Theorem 1(1) says this event is unreachable under the theorem's assumptions.

- *Stop-blocking.* Terminators (`<STOP>`, an object-closing token, or EOS at an incomplete root) are masked while any tracked requirement is unmet.
- *Quota-blocking.* An action that would increment an already satisfied exact count is removed, preventing overshoot.
- *Steering and closure.* While some requirement remains unfilled and is reachable from the current expansion point, the layer restricts to actions that make progress toward it, or to a minimum-cost path that reaches one. Once every requirement is met, it forces a shortest valid closure path to a terminator.

Stop-blocking and quota-blocking are counting *validity*: they are what Theorem 1(2) uses. JSON transfer does not steer. Steering is *timely completion*: it does not change the set of faithful terminating outputs, but it is what makes a rollout reach one of those outputs before a finite action cap. The layer adds no parameters. Any host that can score a structurally filtered action set can use it: a trained native decoder, or an external model restricted to currently valid action strings (Appendix I). Online blocking is $O(k)$ in the number of tracked quotas; the classical compiled product of the same counters is $\prod_i (q_i{+}1) \times 2^{n_{\text{ops}}}$ (Appendix F).

3.1 Controlled environment

To prove terminating-output soundness and isolate the three masks, we use a synthetic AST-synthesis DSL where every requirement is exactly checkable. Each example pairs a natural-language prompt with a JSON AST. The grammar includes program, class, function, block, control flow, assignment, call, literal, variable, arithmetic, comparison, break, and continue node types. Prompts describe exact or at-least-one requirements over structural counts and operator mentions. A representative prompt is *"use exactly 2 for-loop(s), include exactly 1 if-statement(s), with 1 try/except block(s), 9 variable assignments, perform div"*, parsed as `counts: {Assign: 9, Continue: 1, For: 2, If: 1, Try: 1}` and `operators: [Div]`. The full 123-action rollout is in Appendix J.

The action language is RNNG-style (Dyer et al., 2016): `NT`(t) opens a node, `GEN`(v) emits a terminal or operator, and `REDUCE` closes a subtree. The shared grammar interface enforces parent–child rules, terminal typing, and min/max child counts. The operational `Program` root is chunked at a finite capacity; reconstruction concatenates chunk children into one logical variadic `Program`. The checked-out IID dataset contains 8,000 training, 1,000 validation, and 1,000 test examples. JSON transfer is independent of this DSL (Section 6).

4 Formal Soundness

The headline guarantee is soundness: if generation terminates, the parsed requirements are satisfied. Proofs are in Appendix A. The definitions below are for the finite AST grammar used as the controlled environment. Theorem 3 is the matched object lower-bound adapter, not the array micro-suite.

Definition (Quota-blockable node types). The nine types `FunctionDef`, `Class`, `For`, `While`, `If`, `Assign`, `Try`, `Break`, and `Continue` are *quota-blockable*: once a type's quota is filled, its `NT` action is masked for the remainder of the rollout.

Definition (Quota-safe grammar). A grammar is quota-safe if no parent node type with positive minimum child count has a valid-child set consisting entirely of quota-blockable node types. Quota safety is checked mechanically for the deployed grammar, together with reachability of every quota-blockable type from `Program`.

Definition (Closure-safe grammar). A grammar is *closure-safe under quota blocking* if, for every reachable quota-blocking pattern, the directed relation induced by the closure policy's retained cheapest-child transitions is acyclic. The deployed AST grammar is closure-safe: the relation is acyclic under all 2^9 blocking patterns the nine quota-blockable types can produce.

Theorem 1 (Soundness of quota-masked decoding)**.** *Consider decoding from the initial state after <BOS> on a grammar that is quota-safe and closure-safe under the deployed quota-blocking relation, with the grammar mask and quota layer of Section 3, for a prompt whose parsed requirement set is nonempty. Then:*

1. *at every step the quota-filtered valid set $A_{\text{valid}}(s)$ is nonempty, and the implementation's raw-grammar safety fallback never fires;*
2. *every rollout that emits <STOP> satisfies every parsed requirement (all exact counts met, all required operators used);*
3. *every rollout that reaches a state in which all requirements are satisfied emits <STOP> after finitely many further actions under the closure policy.*

Soundness does not bound the time to *reach* the all-requirements-met state. Exact-count masking changes insertion costs during a rollout (quota interference), which is why blocking alone—already sufficient for terminating-output validity—can still exhaust a global action cap. Steering is the empirical policy that reduces those nonterminations; it is not required for Theorem 1(2). Separately, the progress mask can steer only toward a quota reachable from the current expansion point; in those unforced windows an arbitrary host may nest indefinitely (Remark 1 in Appendix A).

A supporting certificate search yields a per-prompt bound W for *conforming* rollouts—rollouts that, in every unforced window, follow the closure rule (Appendix A).

Theorem 2 (Certified termination bound for conforming rollouts)**.** *Fix a prompt satisfying the assumptions of Theorem 1, and let the certificate search terminate on it with value W. If $B \geq W$, then every conforming rollout emits <STOP> within B actions and, by Theorem 1(2), satisfies every parsed requirement.*

Coverage is limited: 361/1,000 IID prompts are certified at the reported search budget, and every non-certified case is search-limited rather than a demonstrated policy failure (Appendix B). We treat termination-in-budget as an empirical complement to Theorem 1, not as the central theoretical claim. Theorem 2 is a verified finite-horizon guarantee on conforming rollouts; it does not say that every host terminates within the reported action budget.

5 CONTROLLED EVIDENCE

The DSL is the environment in which we can turn each mask on and off. **Valid** is the percent of samples that reconstruct one logical `Program`, emit `<STOP>`, and do not hit the action budget. **All-req** is the percent of prompts whose parsed requirements are all satisfied—the binary quantity Theorem 1(2) speaks to. **Sat.** is the mean per-prompt fraction of requirements met, scored even on truncated trees. **Type F1** is multiset F1 over AST node-type bags of the reference and generated trees (terminals and operators included; no extra alignment). **Sim Hash** is one minus normalized Hamming distance between 64-bit path-token SimHash fingerprints of the two trees. **C-sub miss** counts All-req failures among prompts whose full-policy certificate search terminates with $W \leq B$—the certificate-covered subset used for finite-horizon validation. Native decoders use a frozen MiniLM prompt encoder (Wang et al., 2020) and a 9.2M-parameter transformer, trained as in Appendix J and decoded with a 600-action cap, temperature 1.0, and no top-k or nucleus truncation: sampling is categorical over the quota-filtered action set. Ablations use one seeded sample of 200 IID-test prompts (67 simple, 66 medium, 67 complex).

5.1 WHAT EACH MASK CONTRIBUTES

Condition	Run	N	Valid	Sat.	All-req	Type F1	Sim Hash	B-hit	C-sub miss
Grammar only	`tfm_grammar[3 seeds]`	600	0.5%±0.4	50.3%±2.6	0.5%±0.0	0.526±0.001	0.881±0.004	99.5%±0.4	225
Quota blocking	`tfm_quota_block[3 seeds]`	600	92.7%±3.4	99.0%±0.5	92.7%±3.4	0.882±0.010	0.884±0.009	7.3%±3.4	5
Full layer	`tfm_full[3 seeds]`	600	100.0%±0.0	100.0%±0.0	100.0%±0.0	0.844±0.002	0.875±0.001	0.0%±0.0	0

Table 1: Causal ablation on the shared 200-prompt IID sample, seeds 42/43/44 (N=600). Quota blocking is counting validity (terminating outputs are faithful). The full layer additionally steers for timely completion within the action cap.

Table 1 is the main controlled result. Grammar-only decoding—structural masks without quota—reaches 0.5% All-req: 597 of 600 samples exhaust the budget, and the 50.3% Sat. score is chance satisfaction in truncated trees. Quota blocking raises All-req to 92.7%. Its 44 failures across 600 rollouts are exactly its 44 budget hits: once a rollout terminates, the validity filters have already secured every parsed requirement. That is Theorem 1(2) in the table. Steering does not tighten that guarantee. It is a timely-completion policy that removes residual nonterminations, and the full layer then reaches 100.0% All-req, Sat., and Valid across all three seeds.

Steering optimizes timely completion under a finite budget, not imitation of the reference AST. Relative to quota blocking, the full layer therefore lowers reference-structure similarity (Type F1 $0.882 \rightarrow 0.844$; Sim Hash $0.884 \rightarrow 0.875$ in Table 1). That is a completion-policy trade-off, not a loss of specification fidelity, and we do not read these scores as naturalness or coherence.

The 99.0% versus 92.7% Sat./All-req gap on the blocking row is partial credit on unfinished trees, not a second success criterion. Blocking All-req declines from 95.5% on simple prompts to 89.6% on complex prompts, while the full layer remains at 100% in every stratum—the interference regime that soundness does not bound. Rejection sampling under the same seed-42 transformer and the same production B=600 cap solves 0/200 prompts by 64 draws (Appendix C): grammar-only trajectories almost always exhaust that cap (Table 1), so the baseline is a statement about rejection sampling under the evaluation budget, not about unbounded rejection.

5.2 COST OF THE MASKS

Table 2 reports the cost of obtaining that correctness. Grammar-only always exhausts the 600-action cap. Quota blocking is the cheapest condition: it terminates earlier (mean 289 actions) and is the only row faster than grammar-only. Steering is the expensive term: it is the cost of timely completion, not of counting validity. The full layer is about $5\times$ slower per action than grammar-only, but it emits 37% as many actions (mean 220) and is the only condition with zero budget hits. The 100.0% All-req row is therefore not free; it buys completion within the cap at a per-action steering cost, with no extra peak memory. On that same timed sample the completion policy also moves reference-structure similarity: Type F1 $0.874 \rightarrow 0.845$ and Sim Hash $0.879 \rightarrow 0.877$ (Table 30 in Appendix F), matching

Condition	Wall (min)	Act/s	Mean act.	All-req	Valid	B-hit
Grammar only	34.8	57.5	600.0	0.5%	0.0%	100.0%
Quota blocking	13.9	69.0	288.8	88.0%	88.0%	12.0%
Full layer	64.8	11.3	219.8	100.0%	100.0%	0.0%

Table 2: Wall-clock cost of the three native conditions on the same 200-prompt sample (seed-42 transformer, Apple MPS, model load excluded). Quota blocking is validity; Full layer adds steering for timely completion. Peak RSS is 4.3 GB and peak device memory 227 MB in every condition. Blocking All-req is 88.0% on this one seed versus 92.7% in the three-seed ablation.

the three-seed ablation direction ($0.882 \rightarrow 0.844$). Incremental quota filtering is an $O(k)$ mask cost on top of the base engine (Table 37). On the same sample, blocking All-req falls from 100% at $k \leq 3$ count quotas to 76.9% at $k \geq 7$ because more prompts exhaust the cap, while the full layer stays at 100% in every bin (Appendix F).

5.3 Confirmatory full split

Condition	Run	N	Valid	Sat.	All-req	Type F1	Sim Hash	B-hit	C-sub miss
Transformer full, IID-1000	`tfm_full_iid1000`	1000	99.4%	99.8%	99.4%	0.847	0.878	0.6%	0

Full-IID complexity	N	Valid	Sat.	All-req	B-hit
simple	333	100.0%	100.0%	100.0%	0.0%
medium	333	100.0%	100.0%	100.0%	0.0%
complex	334	98.2%	99.5%	98.2%	1.8%

Table 3: Full-layer confirmation on all 1,000 IID-test prompts (replacement seed-42 transformer). All six failures are complex-prompt budget hits, not requirement overshoots.

On the complete IID split the same full layer reaches 99.4% Valid and All-req (Table 3): 100.0% on simple and medium prompts, 98.2% on the 334 complex prompts. All six misses are budget hits, and all six have reference streams longer than the 600-action cap (647–805 actions; Table 15). They are a diagnosis of the action cap, not a counterexample to Theorem 1(2). Architecture matching, compact OOD probes (96.0–99.5% All-req; overlapping rows), and budget-cap diagnostics live in Appendices C and D.

5.4 Host compatibility

The same masks can wrap an external text model. On the 200-prompt sample, frozen `Qwen/Qwen2.5-0.5B-Instruct` (Qwen Team, 2024) reaches 0.0% / 14.0% / 66.5% All-req under grammar, blocking, and the full layer (Table 17 in Appendix E). The same family at 7B reaches 89.0% All-req full-layer; a different family, `HuggingFaceTB/SmolLM2-1.7B-Instruct` (Allal et al., 2025), reaches 28.0% (Tables 18–19). Every terminating rollout on all three hosts satisfies every parsed requirement; residual misses are budget hits. The 7B row is a capacity gain within one family, not a claim that larger models implement cardinality. The 7B and SmolLM2 hosts are full-layer only.

6 Transfer: JSON Schema Cardinality

If the only evidence were the DSL, a reviewer could dismiss the problem as artificial. This section asks three questions of a real specification language. First: are quantitative constraints common? Second: when a backend emits no sampled cardinality failure, is that *enforcement support* or *empirical compliance*? Third: where a compiler leak is present, does a scoped quota adapter repair it?

Mechanism transfer is direct. The matched repair cell uses stop-blocking (`required`, `minProperties`); the prefix-parser micro-suite also supports selected item-count bounds. Neither path steers. Prevalence establishes that the problem is not a toy. The paired repairs—60 draws and 300 draws—are a diagnostic instance, not a population effect-size estimate.

6.1 Quota-relevant constraints are common

We exhaustively parse the 952 JSON documents in the official SchemaStore checkout at commit `8a7f1de10fb52fef096aa5f199fd5ba30abdba8a`, retrieved 2026-07-28 (SchemaStore Contributors, 2026). Of these, 691/952 (72.6%) contain at least one of the nine predeclared cardinality keywords in Table 4. Presence requirements dominate (`required`: 66.9%), but bounded repetition and distinctness are also substantial (`minItems` 23.9%, `uniqueItems` 22.1%, `maxItems` 9.7%). The checkout contains no `minContains`, `maxContains`, or `dependentRequired` occurrence; those zeroes are corpus outcomes, not post-hoc exclusions. A zero observed violation rate at larger host scale, or under a different compiler, is *empirical compliance*; it does not by itself establish *enforcement support* (the violating continuation is excluded by a backend-native membership test).

Keyword	Schemas	Prevalence	Occurrences	Median	p90	Max
`minItems`	228/952	23.9%	2918	1	2	10
`maxItems`	92/952	9.7%	764	2	10	100000
`minProperties`	78/952	8.2%	257	1	1	2
`maxProperties`	33/952	3.5%	70	1	5	25
`required`	637/952	66.9%	11955	2	5	39
`uniqueItems`	210/952	22.1%	2211	–	–	–
`minContains`	0/952	0.0%	0	–	–	–
`maxContains`	0/952	0.0%	0	–	–	–
`dependentRequired`	0/952	0.0%	0	–	–	–

Table 4: Cardinality-keyword prevalence in the pinned 952-schema SchemaStore checkout. "Schemas" counts documents containing the keyword at least once; "Occurrences" counts appearances at actual subschema positions.

Observed bounds are usually small: among integer-valued occurrences, `minItems` has median 1 and p90 2; list-valued `required` declarations have median 2 and p90 5. This is the regime in which explicit finite counter state is plausible, although the long tail (a `maxItems` of 100,000) warns against treating every bound as cheap. The same corpus also shows why a general schema product is hard: 712/952 schemas (74.8%) contain `$ref`, and 135/952 (14.2%) contain a resolvable reference cycle (Table 5). The adapter below therefore restricts to directly tracked positions and does not claim to resolve composition or cyclic references.

Corpus property	Schemas	Prevalence
At least one listed cardinality keyword	691/952	72.6%
At least one `$ref`	712/952	74.8%
Resolvable `$ref` cycle	135/952	14.2%

Table 5: SchemaStore reference structure alongside cardinality prevalence. The 14.2% cycle rate is scoped to locally resolvable references.

6.2 A measured gap in a pinned decoder

We pin `outlines==1.3.2`, `outlines_core==0.2.14`, `transformers==4.57.6`, and `jsonschema==4.26.0`, using `Qwen/Qwen2.5-0.5B-Instruct` at a frozen revision (Outlines Contributors, 2026; Qwen Team, 2024). Outlines presents JSON Schema as a structured-output type with a broad valid-structure claim; its pinned backend lists several cardinality keywords but not a complete per-keyword support matrix. We treat documentation as motivation and measure behavior independently.

Before compilation or generation we froze a SHA-ranked selection of up to ten SchemaStore documents per observed keyword, requiring a complete document of at most 16 KiB and excluding `$ref`. The union contains 33 documents; compiler failures stay in the denominator.

Compiler coverage. On this pinned path, 15/33 frozen real schemas fail to compile; only 18 reach generation. One spends at least 1,800 seconds in regex construction before a recorded timeout. Compiler coverage is itself a material boundary of FSM-compiled structured decoding.

For each compiled document we draw four outputs with fixed per-document seeds, temperature 1, and a 512-token cap, then apply the standard Draft-7 validator. "Shape" removes all nine predeclared cardinality keywords while retaining the rest of the schema. "Isolated gap" means that removing only the named keyword changes a failure to a pass.

Across 72 generated outputs, 70 parse, 66 pass the shape schema, and 62 pass the complete schema. The isolated differences are uneven rather than a blanket failure: 2/20 outputs in the `required` stratum and 2/16 in the `minProperties` stratum exhibit a keyword-isolated gap, while no isolated gap appears for `minItems`, `maxItems`, or `uniqueItems` (Table 6). The selected `maxProperties` document does not compile. These are results for one frozen selection, model, library version, and sampling configuration.

Keyword	Selected	Compiled	Outputs	Parse	Shape	Full	Isolated gap
`minItems`	10	5	20	20/20 (100.0%)	19/20 (95.0%)	19/20 (95.0%)	0/20 (0.0%)
`maxItems`	8	4	16	15/16 (93.8%)	13/16 (81.2%)	13/16 (81.2%)	0/16 (0.0%)
`minProperties`	7	4	16	16/16 (100.0%)	16/16 (100.0%)	14/16 (87.5%)	2/16 (12.5%)
`maxProperties`	1	0	0	–	–	–	–
`required`	10	5	20	19/20 (95.0%)	19/20 (95.0%)	17/20 (85.0%)	2/20 (10.0%)
`uniqueItems`	10	7	28	28/28 (100.0%)	25/28 (89.3%)	25/28 (89.3%)	0/28 (0.0%)
`minContains`	0	0	0	–	–	–	–
`maxContains`	0	0	0	–	–	–	–
`dependentRequired`	0	0	0	–	–	–	–

Table 6: Pinned Outlines JSON-Schema probe. Keyword strata overlap. "Full" is Draft-7 validity; "Shape" validates after removing all predeclared cardinality keywords; "isolated gap" removes only the row keyword. Compiler failures remain in "Selected."

The pattern matches the DSL: the structural decoder is mostly succeeding at well-formed JSON, and the residual errors are quantitative lower bounds—exactly the class stop-blocking is designed to close.

The same 33 documents, model, seeds, and 512-token cap on `llguidance==1.7.6` (Guidance-AI, 2026) compile a different, non-nested subset (12/33 vs. Outlines 18/33; intersection 6). Isolated cardinality gaps are 0/48 on compiled llguidance cells and 4/72 on Outlines (Table 31 in Appendix G). We do not pool the libraries: the leak is a property of what a given compiler builds. Isolated Outlines misses also fall with model size on a frozen compiler (4/72, 2/72, 0/72 at 0.5B/1.5B/3B; Table 32); sampling can hide a compilation hole without closing it. Zero sampled failures is not enforcement. A sampling-free compiler-language probe (Table 35; protocol in Appendix G) finds violating `minProperties`/`maxProperties` completions admitted by pinned Outlines 1.3.2, while llguidance 1.7.6 and XGrammar 0.2.7 block the corresponding continuation. The llguidance 0/48 cell is therefore enforcement on the compiled subset, not luck. Official Draft 7 dedicated files add compile coverage, with 0 dedicated-file language leaks (Table 36; (JSON Schema, 2026)). We do not pool backends, and we do not claim that every engine needs an adapter.

6.3 A PREDECLARED LOWER-BOUND ADAPTER

After measuring the gap and before implementing the adapter, we froze its scope to exactly those two leak sites: `required` and `minProperties`. The matched evaluation includes every base-compiled selected document containing either keyword (15 schemas, four draws each), with the same model, prompt, temperature, per-document seeds, and 512-token bound. The remaining keywords are not added post hoc: three showed no isolated gap, one selection did not compile, and three are absent from the corpus.

The implementation composes with, rather than replaces, the pinned Outlines logits processor. A lightweight JSON prefix parser maintains a container stack and the directly reachable schema position through `properties`, `items`, and `additionalProperties`. After Outlines filters a step, the adapter removes any object-closing token whose decoded fragment would close a directly tracked object with a missing required name or fewer than `minProperties` distinct names; it also excludes EOS until the root value is complete. This is the JSON analogue of blocking a terminator while a lower-bound quota is unmet: counting validity, without steering for timely completion. The matched cell is lower-bound-only; array `minItems`/`maxItems` blocking is used in the micro-suite, not in the 60- or 300-draw estimates. The frozen matched cell remains `required`/`minProperties`. A conservative preflight checks 488 actual subschema positions for explicit lower/upper and closed-property contradictions and finds none; it is not a general satisfiability procedure.

Theorem 3 (Scoped JSON terminating-output soundness)**.** *For every directly tracked object reached by the adapter's container/schema stack, any terminating root JSON value satisfies the enforced* `required` *and* `minProperties` *lower bounds for that object. The statement is narrower than Theorem 1: composition operators remain the base processor's responsibility, and this adapter does not establish a nonempty valid set or a schema analogue of Theorem 2.*

Array item-count support in the micro-suite is outside this theorem.

On the measured 60 draws the added filter causes zero empty-mask events and removes 1,382 quota-unsafe closing candidates. It closes all four keyword-isolated failures. Full Draft-7 validity moves from 53/60 to 57/60; parse validity stays 58/60 and B-hit stays 2/60 (Table 34). The `minProperties` stratum moves from 14/16 to 16/16, and the overlapping `required` stratum from 46/52 to 49/52. The three residual failures are outside the enforced subset: two incomplete prefixes under the 512-token bound, and one otherwise complete instance that violates `pattern`. Table 34 in Appendix G records the matched cell.

Repeating the matched adapter at twenty draws per document (n=300) records 16 repairs and 0 breaks (full validity 268/300 to 284/300; McNemar $p\approx0.00018$; Table 33 in Appendix G). The 16 repairs sit on 3 of 15 schemas, so the schema-clustered interval on the rate still includes zero. Mean tokens move from 66.8 to 70.7. This remains a mechanism probe on a frozen pin: a paired existence of a repair cell, not a population-effect estimate.

7 Limitations

The ablation and host rows use a 200-prompt sample; the 1,000-prompt split confirms only the full layer on one replacement seed-42 transformer. The 7B and SmolLM2 hosts are full-layer on that sample. The JSON study covers one SchemaStore commit, a size- and reference-restricted selection, Outlines 1.3.2, llguidance 1.7.6, and XGrammar 0.2.7 on the micro-suite. Compiler coverage is a material boundary (18/33 Outlines, 12/33 llguidance). The matched adapter does not resolve composition or `$ref` (74.8% of the checkout), is not ported to llguidance, and has no schema termination certificate. The official Test Suite supplies compile and isolated-failure coverage, not JSONSchemaBench generation throughput (Geng et al., 2025). Spider/SQL remains outside this submission.

8 Conclusion

Structured validity and quantitative specification fidelity are separable obligations. Grammar masks guarantee the first; they do not automatically guarantee the second. Semantic quota constraints add the missing incremental state—counts, required operators or keys, and collection cardinalities—in the parser or container the structural decoder already maintains. Stop-blocking and quota-blocking make every terminating output faithful under explicit grammar assumptions. Steering is a completion policy under a finite budget, not a second correctness criterion, and it trades some reference-structure similarity for timely termination. On real structured-generation backends, enforcement support is implementation-dependent: a backend can compile a stricter subset, mask a violating token, or merely happen not to sample a leak. A scoped online adapter is warranted where that leak is present, not as a universal replacement for the base engine. What remains open is relational structure, schema composition, and `$ref` resolution—engineering of reachable state, not a new counting theory.

AI USE STATEMENT

In this work, we used generative AI tools for drafting and editing prose, formatting LaTeX, and searching related literature. We have not used generative AI tools to generate the evaluation datasets, to formulate the theorems, or to produce the reported experimental numbers. Other required-disclosure tasks listed in the ICLR 2027 AI Policy for Authors are not applicable to this work. Additionally, we used generative AI tools only for the recommended writing and search tasks named above. We have reviewed all AI-assisted text and take responsibility for the final content of this work, including text, claims, and artifacts.

ETHICS STATEMENT

This work uses a synthetic program-generation DSL and publicly available SchemaStore documents. It does not involve human subjects, personally identifiable information, or private user content. Reported schema results are aggregated counts over a pinned public checkout. We do not foresee additional risks beyond those of standard structured-generation research.

REPRODUCIBILITY STATEMENT

An anonymized supplementary artifact is available at `https://anonymous.4open.science/r/corelm-reproducibility-83EB/`. It contains the code, data, saved summaries, and checkpoint hashes. Proofs of Theorems 1–2 are in Appendix A. Certificate coverage and replay audits are in Appendix B. Training and decoding details are in Appendix J. Generated tables record checkpoint, evaluation-code, and source-bundle hashes.

REFERENCES

Kareem Ahmed, Catarina G. Belem, Padhraic Smyth, and Sameer Singh. Semantic probabilistic control of language models. *arXiv preprint arXiv:2505.01954*, 2025.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, et al. SmolLM2: When smol goes big — data-centric training of a small language model. `https://arxiv.org/abs/2502.02737`, 2025.

Debangshu Banerjee, Tarun Suresh, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. CRANE: Reasoning with constrained LLM generation. In *Proceedings of ICML*, 2025. arXiv:2502.09061.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. XGrammar: Flexible and efficient structured generation engine for large language models. *Proceedings of Machine Learning and Systems*, 7, 2024. arXiv:2411.15100.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *Proceedings of NAACL-HLT*, 2016. arXiv:1602.07776.

Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. JSONSchemaBench: A rigorous benchmark of structured outputs for language models. *arXiv preprint arXiv:2501.10868*, 2025.

Guidance-AI. llguidance: Fast grammar-constrained generation. Software pin: llguidance 1.7.6, 2026. URL `https://pypi.org/project/llguidance/1.7.6/`.

Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of ACL*, 2017. arXiv:1704.07138.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson, 3 edition, 2006.

JSON Schema. JSON schema test suite. `https://github.com/json-schema-org/JSON-Schema-Test-Suite`, 2026. Draft 7 tests.

Terry Koo, Frederick Liu, and Luheng He. Automata-based constraints for language model decoding. *arXiv preprint arXiv:2407.08103*, 2024.

Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. NeuroLogic decoding: (un)supervised neural text generation with predicate logic constraints. In *Proceedings of NAACL-HLT*, 2021. arXiv:2010.12884.

Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, Noah A. Smith, and Yejin Choi. Constrained text generation with lookahead heuristics. In *Proceedings of NAACL-HLT*, 2022. arXiv:2112.08726.

Microsoft Guidance Contributors. Guidance: A guidance language for controlling large language models. `https://github.com/guidance-ai/guidance`, 2023.

Outlines Contributors. Outlines, version 1.3.2. Python Package Index, 2026. URL `https://pypi.org/project/outlines/1.3.2/`. Released July 20, 2026; accessed July 28, 2026.

Gabriel Poesia, Alex Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. In *Proceedings of ICLR*, 2022. arXiv:2201.11227.

Matt Post and David Vilar. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of NAACL-HLT*, 2018. arXiv:1804.06609.

Qwen Team. Qwen2.5: A party of foundation models. `https://qwenlm.github.io/blog/qwen2.5/`, September 2024.

SchemaStore Contributors. SchemaStore: A collection of JSON schemas. GitHub repository, 2026. URL `https://github.com/SchemaStore/schemastore`. Commit 8a7f1de10fb52fef096aa5f199fd5ba30abdba8a; retrieved July 28, 2026.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of EMNLP*, 2021. arXiv:2109.05093.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, 2020. arXiv:2002.10957.

Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023.

A Formal Proofs and Steering Costs

Costs used by the steering masks. For a node type t, let $c(t)$ be the minimum number of actions needed to open and completely close one t subtree under the grammar's minimum-child constraints. For terminal nodes, $c(t) = 3$: $\texttt{NT}(t)$, $\texttt{GEN}(v)$, $\texttt{REDUCE}$. For nonterminals,

$$c(t) = 2 + \mathbf{1}_{t \in \{\texttt{BinaryOp},\texttt{Compare}\}} + m(t) \min_{u \in \mathcal{C}(t)} c(u),$$

where $m(t)$ is the minimum child count and $\mathcal{C}(t)$ is the set of valid child types. The implementation computes this by monotone fixed-point relaxation because the grammar is cyclic (Table 7).

For a quota target τ, the progress mask uses the analogous complete-subtree-containing-target cost $c_\tau(u)$. The decoder precomputes $c(\cdot)$ and $c_\tau(\cdot)$ once on the raw grammar (empty blocked set); at-quota types are removed from the candidate action set before the min is taken, not by recomputing these costs. Child sets $\mathcal{C}(t)$ are untyped allowed-type collections and $m(t)$ is a count, not an ordered slot vector. Then $c_\tau(\tau) = c(\tau)$, $c_\tau(u) = \infty$ for terminals $u \neq \tau$, and for nonterminals

$$c_\tau(u) = 2 + \mathbf{1}_{u \in \{\texttt{BinaryOp},\texttt{Compare}\}} + c_\tau(v^\star) + \max(m(u) - 1, 0) \min_{v \in \mathcal{C}(u)} c(v),$$

where $v^\star$ minimizes c_τ over $\mathcal{C}(u)$ (all ties retained). This is the shortest grammar-valid completion of u that inserts at least one τ and then fills remaining required children at ordinary minimum cost. Operator quotas reuse the same mask over `BinaryOp`/`Compare` parents rather than a GEN-token

recurrence. The layer's progress mask keeps, at each expansion point, the child actions minimizing c_τ toward an unfilled quota type (minimum over unmet targets); the closure mask, active once all requirements are met, keeps STOP, then REDUCE, then a cheapest complete child by $c(\cdot)$. These costs define the deployed progress and closure masks. They are not a proof device for terminating-output soundness: clause (2) of Theorem 1 uses only stop-blocking and quota-blocking. Clause (3) uses post-satisfaction closure, not count-progress steering. The complete 37-type table appears in Table 7.

Remark 1 (Why a raw fixed-point budget bound fails)**.** *Soundness does not bound how long a rollout takes to* reach *the all-requirements-met state, and no closed-form bound built from raw-grammar fixed-point costs does either.* (a) Quota interference: *exact-count masking changes minimum insertion costs during the rollout, so raw costs underestimate decode-time costs for adversarial requirement combinations (e.g., inserting* `Break` *costs 7 actions through* `Class` *on the raw grammar, but 12 once a* `Class` *quota fills).* (b) Unforced windows: *the progress mask can only steer toward an unfilled quota reachable from the current expansion point; while completing an* `Assign`*, no quota-blockable type is reachable below it and* `REDUCE` *is illegal until two children exist, so the layer leaves the grammar-valid set unrestricted and an arbitrary rollout may nest expressions to any depth. The frozen-host budget misses of Section 5.4 occur precisely in these windows.*

Definition (Conforming rollout). A rollout is *conforming* if, at every unforced window (a step at which the quota layer leaves the grammar-valid set unrestricted), it selects an action the closure rule would allow: REDUCE if grammar-valid, otherwise a cheapest complete child, otherwise any valid GEN. At all other steps it may select any action retained by the layer.

Proof of Theorem 1. Count-progress steering is a timely-completion policy. It is not used in clauses (2) or (3). Clause (2) is terminating-output soundness and uses only stop-blocking and quota-blocking. Clause (3) is post-satisfaction closure. Clause (1) is shown for the deployed layer, whose progress and closure masks only restrict an already nonempty grammar-valid set.

(1) The quota validity layer removes only premature <STOP> actions and at-quota quota-blockable NT actions. Progress and closure, when they fire, restrict an already nonempty A_{valid} to a nonempty subset or leave it unchanged. Emptiness of A_{valid} could therefore occur only if every grammar-valid action belonged to the union of those two removable sets. That is impossible. The grammar offers <STOP> only at an open Program whose minimum child count is met; at that boundary REDUCE remains valid, including when Program has reached its maximum child capacity and further child expansions are illegal. Quota safety excludes a positive-minimum parent whose only remaining children are at-quota NT actions, and REDUCE is never removed by the validity layer.

(2) <STOP> is masked whenever any exact count is below quota or any required operator is unused, so a rollout can emit <STOP> only from a state in which every requirement is met. Exact counts cannot overshoot because NT(q) is masked once the count reaches quota. Progress steering is not invoked in this argument.

(3) Once all requirements are met, the closure mask is active at every step: it forces <STOP> when grammar-valid, else REDUCE when grammar-valid, else restricts to the raw-cost-cheapest unblocked children (all ties retained). Define $R(s)$ as the longest remaining closure path from state s to <STOP>, maximizing over retained ties and GEN choices. R is well defined and finite: closure-safety says the tied-cheapest unblocked-child relation is acyclic under every one of the 2^9 blocking patterns the quota layer can produce, so each forced completion of a missing child terminates; every open node needs at most its bounded minimum child count of such completions before REDUCE is forced; and at the root, <STOP> is forced as soon as Program's minimum child count is met, with the one boundary case in which Program reaches maximum capacity (forcing a root REDUCE and a fresh Program) followed by a single cheapest child completion and <STOP>. Every retained closure transition $s \to s'$ satisfies $R(s') \le R(s) - 1$, so the rollout reaches <STOP> within R actions. Here Program is an operational parser root with finite chunk capacity: a root REDUCE empties the parser stack and the grammar opens a fresh Program. AST reconstruction concatenates the children of every completed chunk and the final implicit-root chunk into one logical variadic Program. Thus the returned AST and the quota counters are defined over the same complete action stream.

Proof of Theorem 2. This bound is for the deployed policy, including count-progress steering; it is not a clause of terminating-output soundness. The certificate search explores the graph of decode states (parser stack, last action, quota progress) reachable under that policy: at each state it invokes the

Node type	Min actions	Terminal
Dict	2	no
List	2	no
Tuple	2	no
Add	3	yes
And	3	yes
Break	3	yes
Continue	3	yes
Div	3	yes
Eq	3	yes
Gt	3	yes
Gte	3	yes
IntLiteral	3	yes
Lt	3	yes
Lte	3	yes
Mul	3	yes
Neq	3	yes
Or	3	yes
StringLiteral	3	yes
Sub	3	yes
Var	3	yes
Block	5	no
Call	5	no
Param	5	no
Return	5	no
Class	7	no
Else	7	no
Except	7	no
ParamList	7	no
Program	7	no
Assign	8	no
BinaryOp	9	no
Compare	9	no
For	11	no
FunctionDef	12	no
If	12	no
Try	12	no
While	12	no

Table 7: Minimum complete-subtree costs $c(t)$ used by the steering and closure masks, obtained by the fixed-point relaxation in Appendix A rather than as static integers.

deployed masking implementation itself to obtain the filtered valid set and the branch that produced it, expands all retained actions at forced and steered steps, and exactly the conforming subset at unforced windows. Successor states are deduplicated, exploration is memoized per state, and a state repeating along a single rollout aborts the search as uncertified (a cycle would make the worst case unbounded). When the search terminates, every conforming rollout is a path in the explored finite acyclic graph, and W is the length of its longest path to <STOP>; hence every conforming rollout reaches <STOP> within $W \leq B$ actions. By Theorem 1(2), every such terminating rollout is requirement-faithful. Because branch classification is read from the policy's own decision trace rather than re-derived, the certificate applies to the deployed implementation without a transcription gap.

Proof of Theorem 3. For each tracked object, the seen-name set grows monotonically. The adapter admits a closing fragment only when every directly applicable required name is in that set and its cardinality reaches the applicable lower bound. A closed object cannot subsequently lose a name, and EOS is unavailable until the root value is complete. Thus every tracked object in a terminating value satisfies both enforced lower bounds.

B CERTIFICATE COVERAGE AND SENSITIVITY

Both theorems assume a nonempty parsed requirement set: for requirement-free prompts the layer never blocks <STOP> and never forces closure. Terminating-output soundness (Theorem 1(2)) does not use count-progress steering. Mechanically, 0 of the 1000 IID test prompts parse to an empty requirement set. Table 8 reports the certifier's mutually exclusive status taxonomy. A search-cap status is not evidence that the worst case is unbounded.

Prompt set	N	Certified	Ineligible	Policy	Search cap	Config	$\max W$
Representative IID sample	200	76	0	0	124	0	218
IID full	1000	361	0	0	639	0	218
OOD depth	200	34	0	0	166	0	203
OOD count	200	66	0	0	134	0	203
OOD length	200	0	0	0	200	0	–

Table 8: Certificate coverage and mutually exclusive status categories. "Policy" denotes a demonstrated intrinsic failure such as a reachable cycle or safety fallback; "Search cap" denotes a diversity, state, visit, or path guard.

Across all five audited prompt-set rows, the search finds zero demonstrated policy-intrinsic failures: every non-certified case is search-limited. On a fixed 100-prompt sensitivity sample, $4\times$ search caps raise coverage from 43/100 to 54/100 without exposing a policy-intrinsic failure (Table 9).

Search caps	States	Visits	Path	Diversity	Certified	max W	Policy
Standard	2,500,000	12,000,000	5,000	<10	43/100 (43.0%)	191	0
$4\times$ caps	10,000,000	48,000,000	20,000	<40	54/100 (54.0%)	223	0

Table 9: Certificate sensitivity on the fixed 100-prompt IID subsample. The $4\times$ row scales every declared search guard.

Table 10 closes the empirical hook: every certified conforming native replay terminates within its own W. Conformance is a sufficient worst-case envelope rather than a behavioral prediction: only 9.5–11.0% of native rollouts conform at every unforced decision, yet a majority of certified non-conforming rollouts still finish within W.

Decoder	Seed	N	Cert.	Conf.	Unforced	Non-conf.	Cert.+conf. $\le W$	Non-conf. $\le W$	Replay err.
GRU	42	200	76	19/200 (9.5%)	19965	1854	18/18	40/58	0
GRU	43	200	76	22/200 (11.0%)	19679	1682	21/21	36/55	0
GRU	44	200	76	22/200 (11.0%)	19573	1556	20/20	35/56	0
Transformer	42	200	76	21/200 (10.5%)	18827	1485	17/17	40/59	0
Transformer	43	200	76	22/200 (11.0%)	18569	1349	18/18	40/58	0
Transformer	44	200	76	21/200 (10.5%)	19454	1678	18/18	36/58	0

Table 10: Replay bridge between Theorem 2 and the six saved native full-layer runs. "Conf." means every saved action in every unforced window belongs to the certificate's conforming subset.

C REJECTION SAMPLING, ARCHITECTURE CONTROL, AND OOD

Rejection sampling supplies a practitioner-facing baseline for the same seed-42 transformer, representative 200-prompt sample, temperature, and 600-action cap. Reusing the frozen grammar-only draw as attempt one, then drawing up to 63 additional samples, the strict Valid+All-req rule solves 0/200 prompts by 64 draws (Table 11).

Parameter-matched GRU and transformer both reach 100.0% Valid, Sat., and All-req under the full layer (Table 12). Compact evaluation-only OOD probes of the replacement seed-42 transformer on

Success definition	N	Mean draws	Median	Unsolved @64	Action ×
Requirements only	200	63.69	64.0	199/200 (99.5%)	173.5×
Valid + All-req	200	64.00	64.0	200/200 (100.0%)	174.3×

Table 11: Grammar-only rejection sampling on the representative IID sample ($n_{\max}$=64). Action compute compares total decoded actions through first success (or the cap) with one full-layer sample per prompt.

Condition	Run	N	Valid	Sat.	All-req	Type F1	Sim Hash	B-hit	C-sub miss
GRU full	`gru_full[3 seeds]`	600	100.0%±0.0	100.0%±0.0	100.0%±0.0	0.851±0.002	0.882±0.002	0.0%±0.0	0
Transformer full	`tfm_full[3 seeds]`	600	100.0%±0.0	100.0%±0.0	100.0%±0.0	0.844±0.002	0.875±0.001	0.0%±0.0	0

Table 12: Parameter-matched architecture control under the full quota layer on the shared 200-prompt sample, seeds 42/43/44.

depth, count, and length splits from a fresh synthetic pool retain 99.5%, 99.5%, and 96.0% All-req (Table 13). The splits are constructed from that pool as: depth, train depth ≤ 6 vs. test ≥ 8; count, all counted types ≤ 2 in train vs. any type ≥ 4 in test; length, reference action length ≤ 250 vs. ≥ 350. All ten OOD failures are budget hits on complex prompts whose reference streams exceed 600 actions.

Condition	Run	N	Valid	Sat.	All-req	Type F1	Sim Hash	B-hit	C-sub miss
OOD depth full	`ood_depth_full`	200	99.5%	99.5%	99.5%	0.846	0.879	0.5%	0
OOD count full	`ood_count_full`	200	99.5%	99.5%	99.5%	0.844	0.872	0.5%	0
OOD length full	`ood_length_full`	200	96.0%	97.7%	96.0%	0.857	0.889	4.0%	0

Table 13: Compact OOD probe for the replacement seed-42 transformer under the full quota layer. Rows overlap (422 unique prompts in 600 row-slots) and are not independent replications.

D Full-IID Diagnosis and Root-Only Starvation

Decision-trace replay finds that all six full-IID budget hits are policy-consistent, that 46.0–47.5% of their 600 decisions occur in unforced windows, and that one or more structural quotas remain unfilled at exhaustion (Table 14). The six reference streams themselves require 647–805 decode steps. A focused rerun at B=1000 preserves each saved 600-action prefix and makes all six outputs valid and All-req. The full IID reference-length distribution puts the operating point in context: nearest-rank 99th percentile 596, maximum 805, and 990/1,000 references fit within 600 actions (Table 15).

Root-only quota starvation. Across the six native hits, the only unfilled structural count types are `FunctionDef` (6/6) and `Class` (3/6); `For`, `While`, `If`, `Try`, and `Assign` are never unfilled. The grammar makes `FunctionDef` and `Class` direct children of `Program` only, whereas the other listed structural types can also be inserted below `Block`. Thus, from inside a block, no child expansion can reach a root-only quota: the layer forces `REDUCE` when legal, but enters the unforced case of Remark 1b while the current node's minimum-child obligations make `REDUCE` illegal. The same asymmetry appears among the 67 frozen LLM failure replays (`FunctionDef` unfilled in 62, `Class` in 40). One untested extension is a fourth forcing tier: when every remaining count quota is Program-only, force the cheapest completion of current minimum-child obligations until `REDUCE` becomes legal, then force a shortest closure path back to `Program`.

Index	Target nodes	Target steps	Unforced	Unfilled at $B = 600$	$B = 1000$ (steps)
726	283	721	283/600 (47.2%)	FunctionDef 1/5	valid, all-req (673)
751	260	661	285/600 (47.5%)	Class 3/4; FunctionDef 0/4	valid, all-req (669)
776	281	701	283/600 (47.2%)	Class 2/3; FunctionDef 3/8	valid, all-req (696)
786	254	647	282/600 (47.0%)	FunctionDef 3/5	valid, all-req (642)
822	256	656	278/600 (46.3%)	FunctionDef 2/3	valid, all-req (648)
824	315	805	276/600 (46.0%)	Class 1/3; FunctionDef 0/6; operators Sub	valid, all-req (723)

Table 14: Decision-trace diagnosis of the six full-IID budget hits. Target steps count the reference action stream including <STOP>.

Split	N	Ref. p99	Ref. max	Refs. ≤ 600	B-hits	Hit refs. > 600
IID test	1000	596	805	990/1000 (99.0%)	6	6/6
OOD depth	200	617	956	194/200 (97.0%)	1	1/1
OOD count	200	617	956	196/200 (98.0%)	1	1/1
OOD length	200	761	956	186/200 (93.0%)	8	8/8

Table 15: Reference decode-length audit for the complete IID split and the evaluated OOD prompts. Every observed IID and OOD budget hit has an over-cap reference.

E LLM Replay Taxonomy, Hosts, and Budget Validation

Because the masks bind the host at every step, the 67 Qwen misses cannot come from early stopping, quota overshoot, or out-of-vocabulary emissions. Replay through the traced policy confirms that all 67 failures are budget hits and non-conforming; 66 take non-conforming actions in unforced windows, while one enters post-satisfaction closure at action 596 (Table 22). Nine failures are certified budget-ok and 58 are search-limited. Complexity concentration is as expected: All-req falls from 79.1% on simple prompts to 56.7% on complex prompts (Table 24). The 7B full-layer run is stronger (89.0% All-req, 11.0% B-hit) with the same terminating-output soundness: every terminating rollout satisfies every parsed requirement (178/178), and all 22 All-req misses are truncations (Table 20). The SmolLM2 full-layer run is weaker (28.0% All-req) but has the same shape: every terminating rollout satisfies every parsed requirement, and All-req falls from 49.3% on simple prompts to 11.9% on complex prompts (Table 21). Its 144 truncations (72.0% B-hit) and 32 C-sub misses are host-capacity failures, not requirement overshoots.

F Compiled Counters and Scaling

Quota blocking is the online evaluation of a *classical* finite counter automaton whose state is the mixed-radix encoding of the exact-count vector. We measure that product only as a baseline characterization of the counting machine we already evaluate online; we do not claim the automaton as a contribution. Table 25 reports product size and compile cost on the paper's 200-prompt sample. All 200 machines materialize; the largest counts-only table is 11.6 MB. Replay of the saved quota-blocking rollouts finds 0 illegal count increments and 0 <STOP> emissions while counts are unmet. Operator bits multiply $|Q|$ (median 5,472, max 3.04×10^6) but do not change the increment table. This is not a PDA product with the RNNG stack.

Table 26 stratifies the native efficiency JSONL by the number of count quotas k. Grammar-only always exhausts the cap. Blocking All-req falls with k because more rollouts hit the budget while remaining faithful if they terminate; steering keeps both completion and All-req at 100%. The same ordering appears by $q_{\max}$ (Table 27).

A k-stratified timing subset keeps 17 prompts from each bin of the same 200-prompt draw (51 prompts; one prompt with $k{=}0$ is skipped). Wall-clock preserves the 200-prompt efficiency ordering (Table 28); blocking's residual failures sit entirely in the $k{\geq}7$ bin (Table 29).

Index	Complexity	Actions	B=600/B=1000 identical
106	simple	114	yes
144	simple	45	yes
159	simple	112	yes
347	medium	234	yes
462	medium	339	yes
477	medium	121	yes
487	medium	155	yes
491	medium	132	yes
580	medium	196	yes
598	medium	384	yes
604	medium	372	yes
652	medium	122	yes
675	complex	394	yes
762	complex	227	yes
802	complex	299	yes
869	complex	415	yes
876	complex	317	yes
903	complex	204	yes
970	complex	189	yes
997	complex	385	yes

Table 16: Budget-invariance probe: 20 of the 994 already-terminating full-IID rollouts, sampled uniformly without replacement and rerun at B=1000. Every action stream is identical to the B=600 original.

Condition	Run	N	Valid	Sat.	All-req	Type F1	Sim Hash	B-hit	C-sub miss
LLM grammar	`llm_grammar`	200	11.0%	5.2%	0.0%	0.109	0.592	89.0%	76
LLM quota blocking	`llm_quota_block`	200	14.0%	47.7%	14.0%	0.333	0.613	86.0%	56
LLM full	`llm_full`	200	66.5%	68.6%	66.5%	0.492	0.738	33.5%	9

Table 17: Frozen `Qwen/Qwen2.5-0.5B-Instruct` compatibility check on the same 200-prompt sample. The host scores only grammar- and quota-filtered action strings.

G JSON Schema: Second Library, Scale, and Extra Draws

Mask-probe protocol. Each micro-schema has a violating prefix and a legal completion. Outlines membership is a full match of the complete violating string against the pinned `outlines_core==0.2.14` regex. llguidance 1.7.6 and XGrammar 0.2.7 consume only the JSON prefix, then test whether the first token of the violating suffix is in the mask. These backend-native tests answer the same operational question: whether the compiled backend permits the next step of a known violating completion. Some `required` grammars reject schema-valid key orders, so part of those backends' completeness is a stricter compiled language. Adapter columns use `analyze_json_prefix`. Absence of a sampled failure is not enforcement.

Official suite. Dedicated Draft 7 files are counted separately from composition tests (`allOf`, `properties` interaction). Outlines compiles 8/17 `required.json` tests and 0 of the `minProperties`/`minItems`/`maxItems` files, which often omit `type`; llguidance compiles 17/17, 8/10, 4/6, and 4/6. Adapter catches are 6/6 invalid `required.json` instances and the integer-bound `minItems`/`minProperties` invalids; they miss float-coded bounds (`1.0`) and `maxProperties`. This is compile and isolated-failure coverage, not a generation leaderboard.

H Discussion

What is established is terminating-output fidelity for parsed quantitative requirements under explicit grammar assumptions, a causal ablation of validity versus completion, and transfer evidence that

Condition	Run	N	Valid	Sat.	All-req	Type F1	Sim Hash	B-hit	C-sub miss
Qwen2.5-7B full	`qwen7b_full`	200	89.0%	90.3%	89.0%	0.712	0.841	11.0%	4

Table 18: Frozen `Qwen/Qwen2.5-7B-Instruct` full-layer check on the same 200-prompt sample and decode protocol. Grammar-only and quota-blocking conditions were not rerun on this host.

Condition	Run	N	Valid	Sat.	All-req	Type F1	Sim Hash	B-hit	C-sub miss
SmolLM2 full	`smollm2_full`	200	28.0%	33.1%	28.0%	0.296	0.673	72.0%	32

Table 19: Frozen `HuggingFaceTB/SmolLM2-1.7B-Instruct` full-layer check on the same 200-prompt sample and decode protocol. Grammar-only and quota-blocking conditions were not rerun on this host.

quota-relevant constraints are common and that enforcement support is backend-dependent. Online parser or container state is the right representation because the quantities are incremental: they change when a node opens, a key is seen, or an item is added, and they are already present in the structural decoder's stack. Model scale does not close a compilation hole: isolated Outlines misses can vanish under a larger host while the regex language is unchanged (Table 32). Steering trades reference-structure similarity for timely termination; it is not a naturalness metric. Scope limits are those of Section 7: no `$ref`, no composition, no relational constraints across distant subtrees, no claim that online masking dominates GPU-native FSMs, and no claim that every serving backend needs an adapter. Useful next engineering is reference resolution, composition, and wiring the same quota filter through production structured-decoding APIs.

I Host Action-Scoring Interface

Frozen instruction hosts never emit free text. At each step the parser state determines the grammar-valid RNNG action strings (`NT`(t), `GEN`(v), `REDUCE`, `<STOP>`); the quota layer intersects that set as in Section 3; and the host scores only the surviving candidates. The default decoder (`action_score` in `scripts/run_llm_host.py`) builds a token trie over those strings, evaluates each candidate as the sum of conditional token log-probabilities under the frozen causal LM, applies temperature 1.0 with no top-k or nucleus truncation, and samples one action. Ties are broken by the candidate order returned by the grammar. The prompt is the natural-language specification used by the native decoder; the model is not asked to produce JSON or Python. Action budget is 600; LLM token budget is 8192. Schematic: parser state $\rightarrow$ valid action strings $\rightarrow$ quota filtering $\rightarrow$ candidate scoring $\rightarrow$ sampled action $\rightarrow$ parser transition.

J Training Configuration and Full DSL Example

Prompt-visible requirements. Prompt-visible requirements cover counts of classes, functions, for-loops, while-loops, if-statements, else blocks, try/except blocks, assignments, break/continue, arithmetic operators {`Add`, `Sub`, `Mul`, `Div`}, and comparison operators {`Eq`, `Neq`, `Lt`, `Gt`, `Lte`, `Gte`}. Programs are generated with 2–4 top-level items for simple examples, 3–6 for medium, and 5–8 for complex. Canonical top-level ordering removes structural ambiguity from the synthetic target ASTs.

Full-layer complexity	N	Valid	Sat.	All-req	B-hit
simple	67	92.5%	93.8%	92.5%	7.5%
medium	66	83.3%	84.6%	83.3%	16.7%
complex	67	91.0%	92.4%	91.0%	9.0%

Table 20: Qwen2.5-7B full-layer results regrouped by predeclared prompt complexity.

Full-layer complexity	N	Valid	Sat.	All-req	B-hit
simple	67	49.3%	53.3%	49.3%	50.7%
medium	66	22.7%	28.3%	22.7%	77.3%
complex	67	11.9%	17.6%	11.9%	88.1%

Table 21: SmolLM2 full-layer results regrouped by predeclared prompt complexity.

Failure category (all full-layer failures)	Count
Budget exhausted after non-conforming actions in unforced windows	66
Budget exhausted during post-satisfaction closure	1
Total failures	67
Certificate stratum	**Count**
search_limited	58
certified	9
Replay statistic	**Value**
Failures also recorded as budget hits	67/67
Certified budget-ok failures	9/67
Non-conforming rollouts	67/67
Median actions in unforced windows	394/600
Median unforced share	65.7%
Post-satisfaction closure entry	action 596, 5 slots remain
Replay inconsistencies	0

Table 22: Replay-based taxonomy of all 67 All-req failures in the saved Qwen2.5-0.5B full-layer run (67/200 misses), not the 7B or SmolLM2 hosts.

Source run	Samples	Certified	Budget-ok	C-sub miss	All failures	$\max W$
`tfm_full[3 seeds]`	600	228	228	0	0	218
`gru_full[3 seeds]`	600	228	228	0	0	218
`tfm_full_iid1000`	1000	361	361	0	6	218
`llm_full`	200	76	76	9	67	218
Total	2400	893	893	9	73	

Table 23: Certificate-based budget validation for designated full-layer rows. C-sub miss is the All-req failure count on certified prompts with $W \leq B$. Native full-layer rows have zero C-sub misses; the full-IID row's six failures are all outside the certified subset.

Full-layer complexity	N	Valid	Sat.	All-req	B-hit
simple	67	79.1%	80.0%	79.1%	20.9%
medium	66	63.6%	65.0%	63.6%	36.4%
complex	67	56.7%	60.8%	56.7%	43.3%

Table 24: Frozen LLM full-layer results regrouped by predeclared prompt complexity.

k	n	median $\lvert Q\rvert$	p90 $\lvert Q\rvert$	max $\lvert Q\rvert$	median compile (ms)
1-3	39	30	78	192	0.03
4-6	96	576	2,268	18,144	0.07
7+	65	10,944	92,160	367,200	0.64
all	200	992	23,040	367,200	0.08

Table 25: Compiled counts-only counter automata on the 200-prompt sample, reported as a classical-product baseline, not as a claimed construction. Compile times are for materializing the increment table on CPU.

k	All-req (%)			Mean act.			B-hit (%)		
	Grammar	Blocking	Full	Grammar	Blocking	Full	Grammar	Blocking	Full
1-3	2.6	100.0	100.0	600.0	104.3	97.7	100.0	0.0	0.0
4-6	0.0	90.6	100.0	600.0	266.4	195.8	100.0	9.4	0.0
7+	0.0	76.9	100.0	600.0	432.5	328.4	100.0	23.1	0.0

Table 26: Native efficiency sample regrouped by number of count quotas k (seed-42 transformer, N=200). Grammar / Blocking / Full are grammar-only, quota blocking (validity), and blocking plus steering (timely completion).

$q_{\max}$	All-req (%)			Mean act.			B-hit (%)		
	Grammar	Blocking	Full	Grammar	Blocking	Full	Grammar	Blocking	Full
1-5	5.3	100.0	100.0	600.0	67.9	65.2	100.0	0.0	0.0
6-12	0.0	97.1	100.0	600.0	177.3	139.8	100.0	2.9	0.0
13+	0.0	80.5	100.0	600.0	393.0	293.9	100.0	19.5	0.0

Table 27: Same sample regrouped by maximum exact count $q_{\max}$. Column names match Table 26.

Condition	Wall (min)	Act/s	Mean act.	All-req	Valid	B-hit
Grammar only	10.1	50.5	600.0	0.0%	0.0%	100.0%
Quota blocking	3.4	66.2	260.9	90.2%	90.2%	9.8%
Full layer	15.5	10.7	195.2	100.0%	100.0%	0.0%

Table 28: Wall-clock cost of the three native conditions on the k-stratified 51-prompt subset (Apple MPS, model load excluded). Quota blocking is validity; Full layer adds steering.

k	All-req (%)			Mean act.			B-hit (%)		
	Grammar	Blocking	Full	Grammar	Blocking	Full	Grammar	Blocking	Full
1-3	0.0	100.0	100.0	600.0	114.9	105.5	100.0	0.0	0.0
4-6	0.0	100.0	100.0	600.0	246.4	191.5	100.0	0.0	0.0
7+	0.0	70.6	100.0	600.0	421.4	288.4	100.0	29.4	0.0

Table 29: Timed subset regrouped by k (17 prompts per bin). Column names match Table 26.

Condition	N	All-req	Type F1	Sim Hash
Grammar only	200	0.5%	0.526	0.890
Quota blocking	200	88.0%	0.874	0.879
Full layer	200	100.0%	0.845	0.877

Table 30: Paired Type F1 and Sim Hash on the same 200-prompt timed sample as Table 2 (seed-42 transformer). Steering is a completion policy, not imitation of the reference AST.

Library	Compile	Outputs	Shape-valid $\wedge$ $\neg$full
Outlines 1.3.2	18/33	72	4
llguidance 1.7.6	12/33	48	0

Table 31: Pinned Outlines 1.3.2 versus llguidance 1.7.6 on the same 33 frozen SchemaStore documents, `Qwen/Qwen2.5-0.5B-Instruct`, seeds, and 512-token cap. Compile subsets are not nested (intersection 6). Isolated cardinality gaps are never pooled.

Qwen2.5-Instruct	Compile	Shape-valid $\wedge$ $\neg$full	Isolated `required`	Isolated `minProperties`
0.5B	18/33	4/72	2/20	2/16
1.5B	18/33	2/72	2/20	0/16
3B	18/33	0/72	0/20	0/16

Table 32: Outlines decoder-gap pin with the compiler frozen, varying only Qwen2.5-Instruct size. Compile coverage is identical (18/33) because the compile cache is reused. Isolated misses can vanish by sampling without the compiler changing.

Adapter	N	Full valid	Repairs / breaks	B-hit
Outlines	300	268/300	–	16/300
Outlines + quota	300	284/300	16 / 0	12/300

Table 33: Matched Outlines adapter at 20 draws per document (15 schemas). "Full valid" is Draft-7 validity. McNemar $p\approx 0.00018$ on the 16/0 discordant cells. All 16 repairs sit on 3 of 15 schemas. A mechanism probe, not a population-effect estimate.

Decoder	Matched scope	Schemas	N	Valid	All-req	B-hit
Outlines	Matched union	15	60	58/60 (96.7%)	53/60 (88.3%)	2/60 (3.3%)
Outlines + quota	Matched union	15	60	58/60 (96.7%)	57/60 (95.0%)	2/60 (3.3%)
Outlines	`required`	13	52	50/52 (96.2%)	46/52 (88.5%)	2/52 (3.8%)
Outlines + quota	`required`	13	52	50/52 (96.2%)	49/52 (94.2%)	2/52 (3.8%)
Outlines	`minProperties`	4	16	16/16 (100.0%)	14/16 (87.5%)	0/16 (0.0%)
Outlines + quota	`minProperties`	4	16	16/16 (100.0%)	16/16 (100.0%)	0/16 (0.0%)

Table 34: Matched adapter evaluation under the predeclared `required`/`minProperties` scope. "Valid" is parse success; "All-req" is full Draft-7 validity, not a DSL quota score. "Outlines" reuses the frozen base draws; "Outlines + quota" uses the same schema/draw seeds with the composed lower-bound filter. Keyword scopes overlap. A mechanism probe, not a population estimate.

Constraint	Adapter blocks viol.	Outlines accepts viol.	llguidance blocks	XGrammar blocks
`required`	yes	no	yes	yes
`minProperties`	yes	yes	yes	yes
`maxProperties`	no	yes	yes	yes
`minItems`	yes	no	yes	yes
`maxItems`	yes	no	yes	yes

Table 35: Sampling-free compiler-language probes on five tiny schemas. "Outlines accepts viol." is complete-string membership of the violating JSON under the pinned regex compiler. llguidance and XGrammar columns ask whether the violating first suffix token is absent from the mask at the prefix. Adapter columns are the container-state prefix parser. Absence of a sampled failure is not enforcement.

Dedicated file	Tests	Adapter catches invalid	Outlines compile	Outlines leak	llguidance compile
`required.json`	17	6/6	8/17	0	17/17
`minProperties.json`	10	1/2	0/10	0	8/10
`maxProperties.json`	10	0/3	0/10	0	8/10
`minItems.json`	6	1/2	0/6	0	4/6
`maxItems.json`	6	1/2	0/6	0	4/6

Table 36: Official JSON Schema Test Suite (Draft 7) dedicated keyword files. Outlines leaks are compiled-language accepts of gold-invalid instances. Adapter misses on `minItems`/`minProperties` are the float-coded bound `1.0`; `maxProperties` is out of adapter scope. Compile coverage is not generation coverage.

Constraint	Adapter prefix (μs)	Outlines regex compile (ms)	llguidance compile (ms)
required	7.8	0.02	0.26
minProperties	7.1	0.01	0.07
maxProperties	9.1	0.01	0.06
minItems	4.9	0.01	0.06
maxItems	4.6	0.00	0.05

Table 37: One-shot backend compile versus per-step adapter prefix check on the five micro-schemas (CPU). Incremental quota filtering is an $O(k)$ mask cost on top of the base engine.

Component	Configuration	Evidence
GRU decoder	4 GRU layers, hidden 512, action/depth embedding 512, cross-attention heads 4	9,209,919 trainable parameters
Transformer decoder	2 decoder layers, $d_{model} = 512$, feed-forward 2048, 8 heads	9,212,479 trainable parameters
Parameter match	Absolute difference 2,560 parameters	0.03% of the two-model mean
Training	AdamW, LR 0.0003, batch 32, max 40 epochs, patience 8, native seeds 42/43/44	Frozen prompt encoder sentence-transformers/all-MiniLM-L6-v2; single NVIDIA L4 GPU per run
Native decoding	Sampled decoding with temperature 1.0, reduce bias 0.0, max action budget 600; no top-k or nucleus (top-p) truncation—categorical over the quota-filtered action set	Strict evaluations use one shared seeded simple-random sample of 200 IID-test prompts drawn without stratification (seed 20260727); its realized allocation is 67 simple, 66 medium, 67 complex
LLM hosts	Qwen/Qwen2.5-0.5B-Instruct, Qwen/Qwen2.5-7B-Instruct, and HuggingFaceTB/SmolLM2-1.7B-Instruct; max actions 600, max LLM tokens 8192	Each host scores only grammar-valid action strings, further filtered by the quota layer where enabled. The 7B and SmolLM2 hosts are evaluated in the full-layer condition only

Table 38: Model, training, and decoding details extracted from saved configs and run summaries.

Full constrained rollout for the Section 3.1 example.

```
<BOS>
NT(Program)
NT(For)
NT(Block)
NT(Assign)
NT(Var)
GEN(i)
REDUCE
NT(IntLiteral)
GEN(1)
REDUCE
REDUCE
NT(Assign)
NT(Var)
GEN(count)
REDUCE
NT(StringLiteral)
GEN(test)
REDUCE
REDUCE
NT(Assign)
NT(Var)
GEN(x)
REDUCE
NT(StringLiteral)
GEN(hello)
REDUCE
REDUCE
NT(Continue)
GEN(Continue)
REDUCE
NT(Try)
NT(Block)
NT(Assign)
NT(Var)
GEN(arr)
REDUCE
NT(BinaryOp)
GEN(Div)
NT(Var)
GEN(val)
REDUCE
NT(Var)
GEN(result)
REDUCE
REDUCE
REDUCE
NT(Assign)
NT(Var)
GEN(idx)
REDUCE
NT(StringLiteral)
GEN(ok)
REDUCE
REDUCE
NT(If)
NT(Block)
NT(Assign)
NT(Var)
GEN(i)
REDUCE
NT(Var)
GEN(result)
REDUCE
REDUCE
NT(Assign)
NT(Var)
GEN(data)
REDUCE
NT(IntLiteral)
GEN(2)
REDUCE
REDUCE
NT(Assign)
NT(Var)
GEN(x)
REDUCE
NT(Var)
GEN(n)
```

```
REDUCE
REDUCE
NT(Assign)
NT(Var)
GEN(temp)
REDUCE
NT(StringLiteral)
GEN(hello)
REDUCE
REDUCE
NT(For)
NT(Var)
GEN(temp)
REDUCE
NT(IntLiteral)
GEN(5)
REDUCE
NT(IntLiteral)
GEN(5)
REDUCE
REDUCE
REDUCE
NT(Block)
NT(Break)
GEN(Break)
REDUCE
REDUCE
REDUCE
REDUCE
NT(Block)
NT(Break)
GEN(Break)
REDUCE
REDUCE
REDUCE
REDUCE
NT(IntLiteral)
GEN(5)
REDUCE
NT(IntLiteral)
GEN(42)
REDUCE
REDUCE
<STOP>
```

Reconstructed AST (depth-limited).

```
Program
  For
    Block
      Assign
        Var [i]
        IntLiteral (1)
      Assign
        Var [count]
        StringLiteral (test)
      Assign
        Var [x]
        StringLiteral (hello)
      Continue
      Try
        Block
          ...
          ...
          ...
        Block
          ...
    IntLiteral (5)
    IntLiteral (42)
```