HORIZONLESS LOOKAHEAD FOR LANGUAGE MODEL CONTROL UNDER NESTED CONSTRAINTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Constrained generation asks a language model to produce text that follows a rule, such as a JSON schema. At every step it needs the lookahead, the probability that the string can still be completed inside the rule, which is intractable for the language model itself. Current methods approximate the language model by a hidden Markov model and the rule by a finite automaton, and store the lookahead in a table indexed by the remaining length. But the model decides for itself when to end, and rules such as JSON need a stack, which a finite automaton tracks only with exponentially many states. We compute the horizonless lookahead, summed over every length, under a constraint with a stack. In verification this is the probability that a probabilistic pushdown system ends in acceptance, in general hard to compute. A language model, which can end the string at every step, makes it cheap. One summary per level of brackets adds up every way to complete it, at every length, and depends only on the symbol that opened the level. Once the summaries are computed, each token costs about 10% more than unconstrained decoding, independent of the string length. The method covers every deterministic context-free language and, under a length cap, gives a mask that guarantees the output ends. It matches the exact conditional on a nested Dyck language. On JSON and RNA tasks it guarantees the constraint and raises content scores, putting an 8B model ahead of every published model on DeepJSONEval Format.

1 INTRODUCTION

Many uses of language models require the output to follow a rule. A JSON document must match its schema, and a program must parse. Constrained generation enforces such a rule, a formal language α, while keeping the model's own preferences (Geng et al., 2023; Willard & Louf, 2023; Lew et al., 2023; Loula et al., 2025). At each step it samples the next token in proportion to the model's probability of that token times the probability that the string can still be completed inside the rule,

$$p(x_t \mid x_{<t}, \alpha) \;\propto\; \underbrace{p(x_t \mid x_{<t})}_{\text{language model}}\; \underbrace{p(\alpha \mid x_{\le t})}_{\text{lookahead}}.$$

The lookahead sums over every continuation and is intractable for a language model (Fig. 2, left). Ctrl-G makes it tractable by approximating both sides (Zhang et al., 2024; Weng et al., 2025). A hidden Markov model, a surrogate whose future depends only on a small hidden state, replaces the language model. A finite automaton, which reads the string through finitely many states, replaces the rule. Together they form a finite Markov chain over pairs of states, so the lookahead depends only on the current pair and on how many tokens remain (Fig. 2, right). Ctrl-G therefore precomputes it as a table, with one row per pair and one column per remaining length, and at each step reads the row of the current pair. The table is finite because the automaton bounds the rows and a maximum length bounds the columns. Grammar-based methods keep the same length index (Maene & Van den Broeck, 2026; Marzouk & de la Higuera, 2022; Scribner et al., 2026) (Tab. 1).

The table rests on a maximum length, the *budget*, and on a finite automaton. The generation problem assumes neither. First, a language model decides for itself when to end, by emitting an end token, so the lookahead should sum over completions of every length. A budget changes that question, and a table indexed by it puts the length n of the string into the cost of building it. Second, many constraints are not regular. Brackets, JSON documents and function calls match each opening to its

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.

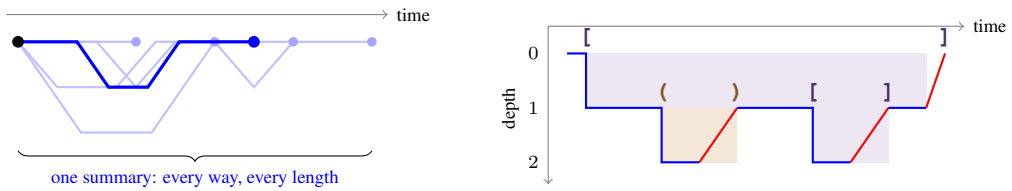

Figure 1: Left: the summary of one level adds up every way the level can be completed, going deeper and back, at every length (faint), so no budget is needed. Right: each level is shaded by the bracket that opened it, and levels opened by the same bracket share one summary.

	class	setting	precomputation	per token
budgeted				
CFG marginals (Marzouk & de la Higuera, 2022)	unambiguous CFG	exact	$O(n^2)$	$O(n^2)$
Ctrl-G (Zhang et al., 2024)	DFA	exact	$O(n)$	$O(1)$
PASTA-G (Scribner et al., 2026)	DPDA	exact	$O(n^3)$	$O(n^3)$
CTRL-CFG (Maene & Van den Broeck, 2026)	LL(1)	window-k bound	$O(1)$ in n	$O(k^3)$
this paper	**DFA / DCFL**	**exact**	$O(n)$ / $\tilde{O}(n)$	$O(1)$ / $\tilde{O}(n)$
horizonless				
this paper	**DCFL**	**exact to tolerance**	$O(1)$ **in** n	$O(1)$

Table 1: Cost in the string length n, worst case per token (amortised under ε-moves). DCFL: deterministic context-free languages. DPDA: deterministic pushdown automata.

close, which needs a stack, and a finite automaton can track the stack only with a state for every possible content, exponentially many in the stack depth. We therefore drop the budget, and replace the finite automaton by a nested-word automaton (NWA), a finite automaton with a stack that pushes a symbol at each opening bracket and pops it at the matching close (Alur & Madhusudan, 2004; 2009). The question becomes the *horizonless lookahead* under such a nested constraint, that is, the probability of completing inside the rule, summed over every length.

In counting terms, this quantity is a horizonless weighted count, the total model weight of the strings the NWA accepts, over every length. Model and automaton together form a probabilistic pushdown system, and the count is its probability of ending in acceptance, a classical quantity in verification (Esparza et al., 2006; Etessami & Yannakakis, 2009), whose fixed point also underlies prefix probabilities of probabilistic grammars (Jelinek & Lafferty, 1991; Stolcke, 1995). In general it is hard even to approximate, and plain iteration can need exponentially many rounds, because such a system may take arbitrarily long to end (Etessami & Yannakakis, 2009). A language model removes this difficulty. When every state can end the string, iteration is guaranteed to converge (Theorem 3.1).

For generation, the horizonless count of an NWA has exactly the structure a lookahead needs. A *level* is the stretch from an opening bracket to its matching close, one flat step of the staircase in Fig. 1 (right). Being horizonless, one *summary* adds up every way a level can be completed, whatever its length, so it does not depend on how many tokens remain and is computed once (Fig. 1, left). Being over an NWA, the count splits into one summary per level of the stack, as termination probabilities do (Esparza et al., 2006). Each summary depends only on the symbol that opened the level, the top of the stack, and not on what lies below (Fig. 1, right). The number of summaries grows with the number of stack symbols, not exponentially with the depth.

Together, the two ideas give an exact lookahead for the surrogate. The summaries are computed once, and each generated token then combines the summaries of the pending levels (Section 3). On a nested Dyck language, where the exact conditional is known, the lookahead matches it (Section 4.1). The algorithm covers every deterministic context-free language (Section 3.1). The same summary gives the budgeted lookahead and, with sums replaced by minima and products by additions, a mask that guarantees the output ends within a length cap (Section 3.2). The horizonless lookahead matches or improves the per-token cost of every method in Tab. 1, and is exact where CTRL-CFG approximates within a window. In practice, decoding with it adds about 10% to the cost of an unconstrained token (Section 4.4).

On real tasks with nested constraints, RNA structure prediction and JSON extraction, decoding with our method makes every output satisfy the constraint and raises content scores (Sections 4.3 and 4.2). On DeepJSONEval it puts an 8B model ahead of every published model on the Format leaderboard (Section 4.2).

2 PRELIMINARIES

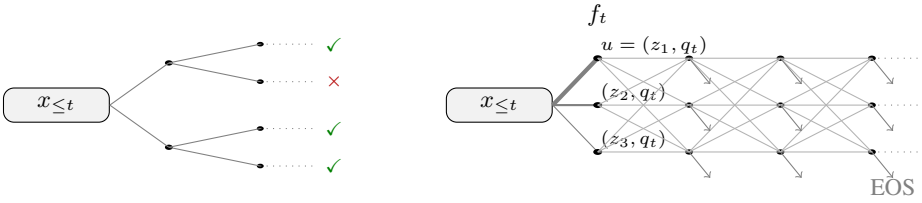

Figure 2: Left: under the language model, one branch per completion, checked only at its end. Right: the future as a walk on the joint states $u = (z, q)$, from q_t and a belief f_t over z.

Conditional probability under a constraint. A language model generates tokens $x_1 x_2 \cdots$ from an alphabet Σ and stops with an end symbol EOS. A constraint $\alpha \subseteq \Sigma^*$ is a language the whole output must belong to. Conditioning on it at step t is $p(x_t \mid x_{<t}, \alpha) \propto p(x_t \mid x_{<t})\, p(\alpha \mid x_{\le t})$, where

$$\boxed{p(\alpha \mid x_{\le t}) = \sum_{x_{>t}\,:\, x_{\le t}\, x_{>t} \in \alpha} p(x_{>t} \cdot \mathrm{EOS} \mid x_{\le t})}$$

is the probability that the model, continued from the prefix, produces a complete string inside the constraint. The sum ranges over completions of every length and is intractable for a language model. It is the quantity this paper computes.

A surrogate model and an automaton for tractability. The boxed probability is tractable when both the model and the constraint depend on the prefix only through finitely many states with the Markov property. Prefixes that reach the same pair of states share one value, and the future is a walk on finitely many joint states (Fig. 2, right). The language model has no such state, since it conditions on the whole prefix (Fig. 2, left). Following Ctrl-G (Zhang et al., 2024), it is replaced by a hidden Markov model fitted to it, the *surrogate*, with hidden states $z \in [h]$, one-step mass $p(a, z \to z')$ for emitting token a while moving from z to z', and end mass $p(\mathrm{EOS} \mid z)$, where $\sum_{a,z'} p(a, z \to z') + p(\mathrm{EOS} \mid z) = 1$. An arbitrary constraint has no finite state either, and Ctrl-G restricts to regular constraints, given by a deterministic finite automaton with states $q \in Q$, $|Q| = s$, transition $\delta(q, a)$ and accepting set F.

A state of the walk is then a joint state $u = (z, q)$, hs in all (Fig. 2). One step emits a token, moving the HMM from z to z' and the automaton from q to $\delta(q, a)$, and the string may end at u, accepted exactly when $q \in F$:

$$T(u, u') = \sum_{a \in \Sigma} p(a, z \to z')\, \mathbf{1}[\delta(q, a) = q'], \qquad \mathbf{e}(u) = \mathbf{1}[q \in F]\, p(\mathrm{EOS} \mid z).$$

Conditioning on the next token, a walk from u either ends now or takes one step and continues. With a maximum length, Ctrl-G counts in this way the walks from u to an accepted end in at most r more tokens,

$$\boldsymbol{\beta}_r = \mathbf{e} + T\, \boldsymbol{\beta}_{r-1}, \qquad \boldsymbol{\beta}_0 = \mathbf{e}, \tag{1}$$

one vector over the hs joint states for every remaining budget r, forming a table with one column per length.

Starting from the prefix. The lookahead needs the joint state after the prefix. The automaton's coordinate is read off by running it on $x_{\le t}$, giving q_t. The HMM's coordinate is hidden, so the prefix leaves a belief rather than a state, $f_t(z) = p(z_t = z \mid x_{\le t})$, maintained by the HMM's forward step on each observed token, $f_t(z') \propto \sum_z f_{t-1}(z)\, p(x_t, z \to z')$. The boxed probability is then the belief-weighted value,

$$p(\alpha \mid x_{\le t}) \;\approx\; \sum_z f_t(z)\, \beta_r(z, q_t),$$

where the weights $f_t(z)$ are the edges entering the first column of Fig. 2, right.

3 The Horizonless Algorithm

Two restrictions behind the table. The table equation 1 stays finite because Ctrl-G restricts constraints to finite automata, which bound its rows, and completions to a maximum length, which bounds its columns. Our method keeps the same surrogate, including its belief over hidden states, and lifts both restrictions. The constraint may carry a stack, and completions may have any length. We develop the lookahead in two steps. First, it factorises over the levels of the stack (Fig. 1, right). Second, each level contributes one summary that adds up every way to finish it (Fig. 1, left). The regular horizonless case is the special case with no stack.

Nested constraints. A pushdown alphabet $\Sigma = \Sigma_c \uplus \Sigma_\iota \uplus \Sigma_r$ sorts tokens into calls, which open a level, internals, and returns, which close one (Alur & Madhusudan, 2004; 2009). A nested-word automaton has states $q \in Q$ and stack symbols $\gamma \in \Gamma$, $|\Gamma| = p$. An internal token moves the state, $\delta_\iota(q, a) = q'$. A call moves the state and pushes a symbol, $\delta_c(q, a) = (q', \gamma)$. A return reads the top symbol and pops it, $\delta_r(q, \gamma, a) = q'$. The token's type alone therefore decides whether the stack pushes, stays or pops. This makes the automaton *input-driven*. Only returns read the stack, and they read only its top symbol, so the automaton has *top-only access*. At any moment the automaton is described by its state and its pending stack, (q, σ) with $\sigma \in \Gamma^*$, and the string is accepted if EOS arrives at $q \in F$ with σ empty. We split the one-token mass by token type, each part a matrix over joint states u:

$$T_\iota(u, u') = \sum_{a \in \Sigma_\iota} p(a, z \to z')\, \mathbf{1}[\delta_\iota(q, a) = q'],$$
$$C_\gamma(u, u') = \sum_{a \in \Sigma_c} p(a, z \to z')\, \mathbf{1}[\delta_c(q, a) = (q', \gamma)],$$
$$R_\gamma(u, u') = \sum_{a \in \Sigma_r} p(a, z \to z')\, \mathbf{1}[\delta_r(q, \gamma, a) = q'].$$

The automaton is a deterministic NWA here. Section 3.1 extends the method to deterministic pushdown automata.

Conditioning on the next token leaves infinitely many unknowns. The state of the walk is now the joint state u together with the pending stack σ, and the target is $\beta(u, \sigma)$, the probability of completing the string inside the constraint from there, at any length. Conditioning on the next token, as in equation 1, relates β at σ to β one level deeper after a call and one level shallower after a return. From the current stack the recursion therefore reaches every stack that extends it, infinitely many, and p^d at depth d even under a budget. The next token is the wrong event to condition on.

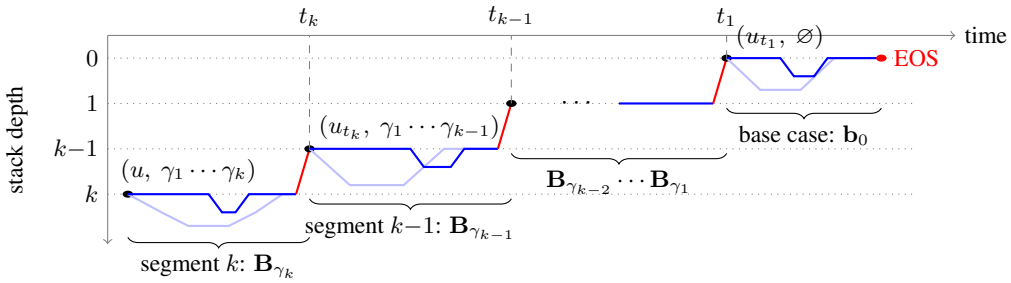

Figure 3: Segment k runs until γ_k is popped (red) at time t_k, staying at depth $\geq k$ in any shape and length (faint), and the next segment starts from u_{t_k}. At depth 0 the last segment ends with EOS.

Factorisation on stack depth. We condition instead on the first moment the stack is *shallower than it is now*, which is the return that pops the current top symbol. Fig. 3 plots the stack depth of a completion of $(u, \gamma_1 \cdots \gamma_k)$ over time. *Segment* k is the part of the completion until the stack first becomes shallower than depth k, at the moment t_k when γ_k is popped. Segment $k-1$ follows, and so on, until at depth 0 the last part ends with EOS. By top-only access, the automaton reads only the top of its stack during segment k, so $\gamma_{k-1}, \ldots, \gamma_1$ are never read and the segment has exactly the probability it would have with the stack γ_k alone. At t_k the future depends on the past only through the boundary state u_{t_k}, because the HMM is Markov in z and the automaton's next move depends

only on its state and its stack top. Summing over the boundary state,

$$\beta(u, \gamma_1 \cdots \gamma_k) = \sum_{u'} \underbrace{p\big(\text{segment ends in } u' \;\big|\; \text{starts in } u,\ \text{top} = \gamma_k\big)}_{=:\, \mathbf{B}_{\gamma_k}(u,u')} \beta(u', \gamma_1 \cdots \gamma_{k-1}).$$

The segment matrix $\mathbf{B}_\gamma$ therefore depends on γ alone. Unrolling to the empty stack, as for termination probabilities of probabilistic pushdown systems (Esparza et al., 2006, Lemma 3.4),

$$\boxed{\beta(u, \gamma_1 \cdots \gamma_k) = \big[\mathbf{B}_{\gamma_k} \mathbf{B}_{\gamma_{k-1}} \cdots \mathbf{B}_{\gamma_1}\, \mathbf{b}_0\big](u), \qquad \mathbf{b}_0(u) := \beta(u, \varnothing).} \tag{2}$$

Equation 2 computes β by a backward pass over the pending symbols, with one matrix $\mathbf{B}_\gamma$ per segment, applied from the empty stack up to the current one. What remains is to find $\mathbf{B}_\gamma$ for each stack symbol γ, and $\mathbf{b}_0$.

Segment $\mathbf{B}_\gamma$ is a block followed by the return that pops γ. Segment k runs until the stack first becomes shallower than depth k, and it ends with the return that pops γ_k. Everything before that return stays at depth $\geq k$ and ends at depth k. It is a block, which we also call a *well-matched block* (Fig. 4). Let $\mathbf{W}(u, u')$ be the total weight of the well-matched blocks from u to u', of any length and any inner nesting. By the Markov property at the boundary between the two parts,

$$\mathbf{B}_\gamma = \mathbf{W} R_\gamma, \qquad \mathbf{b}_0 = \mathbf{W}\mathbf{e}, \tag{3}$$

The second holds because, from the empty stack, a completion is a block at depth 0 followed by EOS in an accepting state. The end of the string plays the role of the last pop (automata that accept by final state at any depth add $\mathbf{b}_0$ at each level, App. C). $\mathbf{W}$ is the only unknown left.

A well-matched block is a sequence of steps. Within a block the stack is never shallower than its starting depth, so conditioning on the next step leaves two cases, exclusive because the alphabet is input-driven (Fig. 4): (i) one internal token; (ii) one call, then an inner block, then its return. The inner block is the same kind of block, one level deeper. The automaton never reads the symbols below its stack top, and neither the HMM nor the automaton depends on the depth. The well-matched blocks from u to u' therefore have total weight $\mathbf{W}(u, u')$ at every depth. One step thus has mass

$$S = T_\iota + \sum_{\gamma \in \Gamma} C_\gamma\, \mathbf{W}\, R_\gamma,$$

with $\mathbf{W}$ itself inside the second term. A block is some number ℓ of steps, so

$$\mathbf{W} = \sum_{\ell \geq 0} S^\ell = (I - S)^{-1}.$$

We assume every hidden state ends the string with positive probability, $\varepsilon = \min_z p(\text{EOS} \mid z) > 0$. Then the sum converges for the same reason a run of coin flips ends. At every step some mass leaves the block, so blocks of ℓ steps carry geometrically less mass as ℓ grows, and one inversion sums the blocks of every length. Substituting S puts $\mathbf{W}$ on both sides,

$$\boxed{\mathbf{W} = \Big(I - T_\iota - \sum_{\gamma \in \Gamma} C_\gamma\, \mathbf{W}\, R_\gamma\Big)^{-1},} \tag{4}$$

a fixed-point equation for $\mathbf{W}$, whose least solution is what must be found (cf. Esparza et al., 2006, Thm. 3.5).

Solving for $\mathbf{W}$. Multiplied out, equation 4 reads

$$\mathbf{W} = I + T_\iota \mathbf{W} + \sum_{\gamma} C_\gamma\, \mathbf{W} R_\gamma \mathbf{W}, \tag{5}$$

which uses only sums and products of nonnegative matrices. We iterate it from below, starting from $\mathbf{W}^{(0)} = 0$, and the iterates increase monotonically to $\mathbf{W}$.

Theorem 3.1 (error after j rounds, cf. Etessami & Yannakakis, 2009)**.** *If $\varepsilon = \min_z p(\text{EOS} \mid z) > 0$, then with the j-th iterate $\mathbf{W}^{(j)}$ of equation 5 in place of $\mathbf{W}$ in the chain equation 2–3, $0 \leq \beta - \beta^{(j)} \leq (1 - \varepsilon)^j$ for every u and σ.*

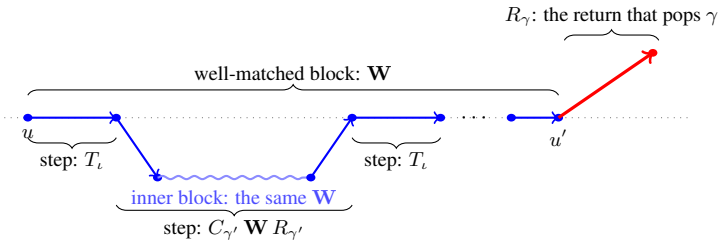

Figure 4: A segment: a block (blue, $\mathbf{W}$) and the return that pops γ (red, R_γ). Each step in the block is an internal token, or a call, an inner block (the same $\mathbf{W}$ one level deeper) and its return.

Algorithm 1 The horizonless algorithm and the cost of each line, with $\omega \leq 3$ the matrix multiplication exponent. No line depends on the position, the depth or the length. Full version: Algorithm 2.

Offline, once per automaton	
1: solve $\mathbf{W} = I + T_\iota \mathbf{W} + \sum_\gamma C_\gamma \mathbf{W} R_\gamma \mathbf{W}$ by fixed-point iteration	$O(p\,(hs)^\omega)$ per round
2: store $\mathbf{B}_\gamma = \mathbf{W} R_\gamma$ for each γ, and $\mathbf{b}_0 = \mathbf{W}\mathbf{e}$	$O(p\,(hs)^2)$ memory
Online, per token	
3: update the HMM belief	$O(h^2)$
4: call: push $\mathbf{B}_\gamma$ times the cached vector; return: pop; internal: keep	$O((hs)^2)$ or $O(1)$
5: score each candidate token by its lookahead from the cached vector, raised to an exponent c	$O(p\,(hs)^2 + h\lvert\Sigma\rvert)$

The j-th iterate includes every block shorter than j tokens, so a completion it misses has at least j tokens. *(Proof: App. A.)*

Online, the product in equation 2 is kept as one vector per pending level. A call multiplies the top vector by $\mathbf{B}_\gamma$, a return pops it, and each candidate token is scored against the belief f_t, so no step depends on the position or the length (Alg. 1).

3.1 The same summary for deterministic pushdown automata

The algorithm never needed the automaton to be an NWA. It needed two properties. First, the string itself must decide where each level begins and ends, as brackets do. The summary cuts a completion where the stack returns to a shallower depth, and every run on the string must cut it in the same places. Second, each accepted string must have a single run. The summary adds up runs weighted by the model, and a string with two runs would be counted twice.

Theorem 3.2 (coverage)**.** *Algorithm 1 computes the horizonless lookahead on every deterministic pushdown automaton, so the languages covered are exactly the deterministic context-free languages.*

A deterministic pushdown automaton has both properties, since its one run per string fixes the stack depth. Its stack top is carried in the state, so hs becomes $hs|\Gamma|$ in every cost (App. C).

The two properties also mark the boundary. The first bounds the method. Without it, the runs of a string split it into different levels, and no single summary exists. The second bounds the problem itself. Without it, exact weighted counting is #P-hard already without a stack (Álvarez & Jenner, 1993).

3.2 Budgets: budgeted weights and a length mask from the same summary

The budget question is still worth asking. Every real decode stops at some output cap, usually a generous one.

Budgeted weights. The budgeted lookahead is the weight of the valid completions of at most r tokens. With a stack no table over stack contents is finite, so the budgeted weights start from the summary. Marking every token with a variable turns $\mathbf{W}$ into a series whose term of each degree is the weight of the blocks of that length, and levels combine by adding their lengths (App. B). This costs $O(1)$ per token without a stack, as Ctrl-G does, and $\tilde{O}(n)$ per token with one, after

$\tilde{O}(n)$ precomputation. The cost still grows with the budget, and a generous cap is where it is least affordable.

A generous budget needs no budgeted weights. The budgeted and horizonless lookaheads differ only by completions longer than the budget. Every hidden state ends the string with probability at least ε, so these carry total weight at most $(1-\varepsilon)^{r+1}$ (Lemma A.1), which is negligible whenever $\beta \gg (1-\varepsilon)^{r+1}$ (measured on Dyck-2 in Tab. 9). What the horizonless weights cannot guarantee is that the string ends within the cap, for instance when the model loops. That needs only a mask.

The mask needs only the shortest completion. A token that leaves no valid completion within the budget must be masked, whatever the weights say. The summary gives the length of the shortest completion by a change of arithmetic.

Corollary 3.3 (shortest completion)**.** *Read equation 5 and equation 2–3 with every sum replaced by a minimum, every product by an addition, every transition of positive weight costing* 1*, zero-weight entries* $+\infty$*, and* EOS *costing* 0*. Then* $\mathbf{W}(u, u')$ *is the length of the shortest well-matched block from* u *to* u' *and* $\beta(u, \sigma)$ *is the length of the shortest completion of* (u, σ)*, and both are computed exactly, in polynomial time, by a shortest-path algorithm on these equations.* (Proof and algorithm: App. B.)

A candidate token is then viable within a remaining budget r exactly when $\beta \leq r - 1$ after it. This budget mask is exact, has no length index, and costs the same order as Algorithm 1.

4 Experiments

We evaluate the horizonless lookahead on a synthetic Dyck-2 language, where the constrained conditional is known exactly, and on two real benchmarks with nested constraints, extracting JSON under deep schemas (DeepJSONEval) and predicting RNA secondary structure. The local hard mask, a baseline, bans the tokens the automaton rejects at the next step, with no lookahead. App. E gives the setup.

4.1 Dyck-2: the horizonless lookahead recovers the exact conditional

Dyck-2 has two bracket types, each imposing its own rule on the level it opens (App. E), as each container in a schema admits its own keys. The language model is a small HMM that tracks the last three open brackets and forgets deeper ones. It is a finite-state imitation of a language model that follows nesting to a limited depth and then loses track of it. The model and the automaton are small enough that the constrained conditional can be computed exactly.

Nesting is where finite states run out. Our joint state space stays at 1,044 states at every depth, while a finite automaton unrolling the same language grows exponentially with the depth (Tab. 2, bottom left).

With the model itself as the surrogate, the lookahead is the conditional. Tab. 2 (top left) reports KL from the exact conditional. When the surrogate is the language model itself, the lookahead is exact and the KL is zero to floating point. The mask alone is .34 nats away.

A surrogate must be calibrated before it is used. On a real task the surrogate cannot be the model itself. It is a smaller HMM fitted on the model's samples. Its lookahead, the probability of satisfying the constraint, varies about twice as much across candidates as the true one. Regressing the true log-lookahead on the surrogate's, within each position, gives a slope of about .45 (Tab. 2, right, shows the raw pairs). Used as is, the surrogate is further from the conditional than the mask alone. Raising its lookahead to a fitted exponent, $c \approx .45$, rescales it and brings it ten times closer to the conditional than the raw surrogate. Both real benchmarks below fit c in $[0, 1]$ on development instances (App. E).

4.2 DeepJSONEval: deep schemas over natural text

DeepJSONEval gives a passage and a JSON schema three to seven levels deep, and the model must extract the passage into a document valid under the schema. Each object or array is a level with its own admissible keys, which is the structure of Section 4.1 at larger scale. We decode 500 of

lookahead reweighting the model	KL (nats)
mask ($c = 0$)	.337
surrogate, exponent $c = 1$	.463
surrogate, fitted exponent $c = .45$	.048
exact lookahead (the model itself)	$\leq 10^{-15}$

depth cap	unrolled DFA $\times$ HMM states
4	32,364
8	533,484
16	$1.4 \cdot 10^{8}$
32	$9.0 \cdot 10^{12}$

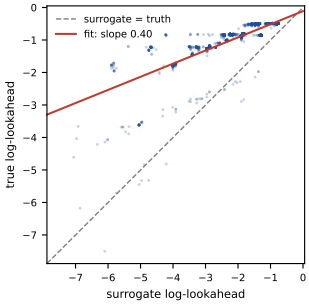

Table 2: Dyck-2. Top left: KL from the exact conditional, 10^5 samples, SE $\leq$ .02. Bottom left: states of a finite automaton unrolling the language to a depth cap, against 1,044 for our method at every depth. Right: true against surrogate log-lookahead ($h = 16$).

the 2,100 instances under the benchmark's prompt and its three metrics. Format checks that the document parses and validates against the schema, Detailed is a recursive field-match score that penalises missing and extra keys, and Strict requires an exact match.

Constraining an 8B model puts it ahead of every published model on Format, and lifts the content metrics at the same time. Tab. 3 places our rows on the benchmark's leaderboard. Unconstrained, the model is below every published row. Constrained with the budget mask and the horizonless weights (see below), Format is 100.00, Detailed rises 18 points to third place, and Strict rises 6. The content gain comes from documents that are now valid and can be scored.

At a generous cap only horizonless weights are affordable, and a budget mask guarantees a valid ending. We cap outputs at 2,048 tokens, against gold documents of at most 336, so the cap binds only on loops. One $\mathbf{W}$ takes up to 38 GB at the largest schema, and budgeted weights would need one per remaining-budget value. The horizonless weights alone leave five of the 500 documents unfinished at the cap, and the mask of Section 3.2 closes all of them.

	Format	Detailed	Strict
best published (Claude Sonnet 4)	99.05	90.73	57.90
median published	94.58	86.95	55.34
Llama-3.1-8B-Instruct (Grattafiori et al., 2024), unconstrained	79.40	72.22	44.40
+ rejection, best of 8	85.20	76.55	45.80
+ local hard mask	95.00	85.95	47.60
+ XGrammar (Dong et al., 2025)	97.40	88.31	50.00
+ horizonless weights only	99.00	89.06	49.60
+ budget mask only	100.00	89.35	49.60
+ mask and weights (ours)	**100.00**	**89.78**	50.40

Table 3: DeepJSONEval. Published rows from Zhou et al. (2025) (all twelve in App. D, best by Format). Ours on 500 of the 2,100 instances, SE $\leq$ .027.

4.3 RNA SECONDARY STRUCTURE: NESTED CONSTRAINTS BEYOND TEXT

Nested constraints are not only a text phenomenon, and in specialised domains with small vocabularies the surrogate is also easier to train. Biology writes structure in nested notations too, and in these notations validity is a matching condition over the whole string. We take RNA. Given a sequence over A, C, G, U, a small character-level model trained on the task writes its structure over three symbols. A bracket pair at positions i and j means that the two nucleotides bind, which requires complementary nucleotides, and a stack represents this naturally. The automaton pushes each opening bracket's position and allows a closing bracket only over a complementary nucleotide. With three output symbols, a small surrogate suffices, and each sequence is set up in under a minute. We decode 215 held-out sequences and report validity and base-pair F1 against the known structure.

Tab. 4 shows that the unconstrained model is valid on one sequence in six. The local hard mask gains four points, and sequential Monte Carlo with 48 particles reaches three in ten at 47 times the model calls. Our method makes every structure valid and raises F1 from .048 to .282.

	valid	F1 (macro)	F1 (micro)	LM calls
unconstrained	.163	.048	.060	131
local hard mask	.200	.063	.077	132
rejection, up to 8	.070	.045	.061	1,002
SMC, 48 particles[†]	.301	.125	.151	6,268
budget mask only	1.000	.280	.262	132
horizonless weights only[‡]	1.000	.282	.272	132
mask and weights (ours)	**1.000**	**.282**	**.272**	132

Table 4: RNA secondary structure, 215 held-out sequences. F1 is over all 215, with invalid structures scored as emitted. [†]Sequential Monte Carlo with a local proposal, four draws per sequence averaged. [‡]Identical to ours, since the weights vanish exactly where the mask forbids. Standard errors are at most .027 for validity and .023 for both F1 columns, bootstrapped for micro F1.

4.4 Coverage and cost

Our algorithm runs on published benchmarks with regular and context-free constraints. We run it on two benchmarks built on finite automata, the Embodied Agent Interface (EAI) and CommonGen, and one built on a context-free grammar, SMILES acrylate generation. On EAI, whose semantic constraints are not guaranteed by construction, it reproduces the published constrained generation results (Yidou-Weng et al., 2026). On CommonGen and SMILES every output satisfies the constraint (App. D).

The weights matter where allowed tokens differ in how often they still reach acceptance. On DeepJSONEval and RNA, as in most published benchmarks with nested constraints, the constraints are syntax, so almost every allowed token can still reach acceptance. The weights alone are then nearly always valid (Format 99.00, and on RNA identical to ours). Where the two come apart, the weights decide. Without weights, Dyck-2 strings are five times too long, EAI task success falls from 89.7 to 79.0, and CommonGen BLEU falls from 31.7 to 1.0 (App. D). Constraints on content, as in planning and reasoning, are where the weights are worth most, and nested benchmarks of that kind are still to be built.

Decoding costs about a tenth more than unconstrained decoding. Online, each token costs 1.10 times an unconstrained token on DeepJSONEval and 1.13 times on RNA. Offline, the summaries are precomputed once per automaton, within three minutes even at the largest, and the surrogate is fitted once per prompt on DeepJSONEval, at a median of 42 s, and once per benchmark elsewhere (App. E, Tab. 10).

5 Conclusion

The horizonless lookahead of constrained generation is exact for the surrogate, for every deterministic context-free language and costs about a tenth more than unconstrained decoding. Under a length cap, the same summary gives a mask that guarantees the output ends.

Two questions remain open. For unambiguous context-free languages that are not deterministic, the runs of a string no longer split it into the same levels. The lookahead can still be approximated, but with work that grows with the prefix, and whether it can be updated at a cost per token independent of the length is open. With a budget, the total cost of decoding a string of length n lies between $\Omega(n)$ and $\tilde{O}(n^2)$. Closing the gap would show whether a budget can cost as little as no budget.

The horizonless count shows what verification and language models can do for each other. In the theory of probabilistic pushdown systems the hard case is a system that may take arbitrarily long to end, where the probability of ending converges arbitrarily slowly. A language model can end the string at every step, which guarantees convergence. On its own, a language model cannot compute the probability that its continuation satisfies a rule. The count gives that probability, and with it

brings exact probabilistic reasoning into generation. The constraint is a rule written down, the model a distribution learned from data, and each makes the other's question answerable.

AI USE STATEMENT

We used generative AI tools for literature search, for writing and testing research code, for running experiments, for checking proofs and implementations against independent references, and for editing text. We have not used them to generate research ideas or claims without author verification. The authors reviewed and verified all AI-assisted work, including every proof, number and citation, and take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

Proofs of every theorem and corollary, with their precise hypotheses, are in Apps. A–C. App. E gives the machines, guides, exponent fits, decoding settings and the provenance of every number, including the checks of each horizonless solve. Code and result files will be released with the paper.

REFERENCES

Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, STOC04, pp. 202–211. ACM, June 2004. doi: 10.1145/1007352.1007390. URL http://dx.doi.org/10.1145/1007352.1007390.

Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3): 1–43, May 2009. ISSN 1557-735X. doi: 10.1145/1516512.1516518. URL http://dx.doi.org/10.1145/1516512.1516518.

Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. #NFA admits an FPRAS: Efficient enumeration, counting, and uniform generation for logspace classes. *Journal of the ACM*, 68(6):1–40, October 2021. ISSN 1557-735X. doi: 10.1145/3477045. URL http://dx.doi.org/10.1145/3477045.

Noam Chomsky and Marcel-Paul Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg (eds.), *Computer Programming and Formal Systems*, volume 35 of *Studies in Logic and the Foundations of Mathematics*, pp. 118–161. North-Holland, Amsterdam, 1963. ISBN 9780444534002. doi: 10.1016/S0049-237X(08)72023-8. URL http://dx.doi.org/10.1016/S0049-237X(08)72023-8.

Padideh Danaee, Mason Rouches, Michelle Wiley, Dezhong Deng, Liang Huang, and David Hendrix. bprna: large-scale automated annotation and analysis of rna secondary structure. *Nucleic acids research*, 46(11):5381–5394, 2018.

Yixin Dong, Charlie F Ruan, Yaxing Cai, Ziyi Xu, Yilong Zhao, Ruihang Lai, and Tianqi Chen. Xgrammar: Flexible and efficient structured generation engine for large language models. *Proceedings of Machine Learning and Systems*, 7, 2025.

Javier Esparza, Antonín Kucera, and Richard Mayr. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2, 2006.

Javier Esparza, Stefan Kiefer, and Michael Luttenberger. Newtonian program analysis. *Journal of the ACM*, 57(6):1–47, October 2010. ISSN 1557-735X. doi: 10.1145/1857914.1857917. URL http://dx.doi.org/10.1145/1857914.1857917.

Kousha Etessami and Mihalis Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1):1–66, January 2009. ISSN 1557-735X. doi: 10.1145/1462153.1462154. URL http://dx.doi.org/10.1145/1462153.1462154.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured nlp tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10932–10952, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

John E Hopcroft and Jeffrey D Ullman. Introduction to automata theory, languages and computation. adison-wesley. *Reading, Mass*, 1979.

Frederick Jelinek and John D. Lafferty. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–323, 1991. URL https://aclanthology.org/J91-3004/.

Donald E. Knuth. On the translation of languages from left to right. *Information and Control*, 8 (6):607–639, December 1965. ISSN 0019-9958. doi: 10.1016/s0019-9958(65)90426-2. URL http://dx.doi.org/10.1016/s0019-9958(65)90426-2.

Donald E. Knuth. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1): 1–5, February 1977. ISSN 0020-0190. doi: 10.1016/0020-0190(77)90002-3. URL http://dx.doi.org/10.1016/0020-0190(77)90002-3.

Greg Landrum et al. Rdkit: Open-source cheminformatics, 2006.

Alexander K Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash K Mansinghka. Sequential monte carlo steering of large language models using probabilistic programs. *arXiv preprint arXiv:2306.03081*, 2023.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li E Li, Ruohan Zhang, et al. Embodied agent interface: Benchmarking llms for embodied decision making. *Advances in Neural Information Processing Systems*, 37:100428–100534, 2024.

Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. Commongen: A constrained text generation challenge for generative commonsense reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1823–1840, 2020.

João Loula, Benjamin LeBrun, Li Du, Ben Lipkin, Clemente Pasti, Gabriel Grand, Tianyu Liu, Yahya Emara, Marjorie Freedman, Jason Eisner, et al. Syntactic and semantic control of large language models via sequential monte carlo. In *International Conference on Learning Representations*, volume 2025, pp. 64758–64791, 2025.

Jaron Maene and Guy Van den Broeck. Tractable constrained generation with LL(1) grammars. In *The Ninth Workshop on Tractable Probabilistic Models*, 2026. URL https://openreview.net/forum?id=TPD419MzWl.

Reda Marzouk and Colin de la Higuera. Marginal inference queries in hidden Markov models under context-free grammar constraints, 2022. URL https://arxiv.org/abs/2206.12862. Presented at LearnAut 2022 (Learning and Automata, ICALP 2022 workshop); no formal proceedings.

PaweĹ Parys, Sairam Vaidya, Taylor Berg-Kirkpatrick, and Loris D'Antoni. Constrained adaptive rejection sampling. *arXiv preprint arXiv:2510.01902*, 2025.

Carine Pivoteau, Bruno Salvy, and Michèle Soria. Algorithms for combinatorial structures: Well-founded systems and Newton iterations. *Journal of Combinatorial Theory, Series A*, 119(8): 1711–1773, November 2012. ISSN 0097-3165. doi: 10.1016/j.jcta.2012.05.007. URL http://dx.doi.org/10.1016/j.jcta.2012.05.007.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Max Scribner, Antonio Vergari, and Vaishak Belle. Pasta-g: Tractable grammar control of large language models. 2026.

Michael F Sloma and David H Mathews. Exact calculation of loop formation probability identifies folding motifs in rna secondary structures. *RNA*, 22(12):1808–1818, 2016.

Richard P. Stanley. *Enumerative Combinatorics*, volume 2 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, January 1999. ISBN 9780511609589. doi: 10.1017/cbo9780511609589. URL http://dx.doi.org/10.1017/CBO9780511609589. With an appendix by Sergey Fomin.

Andreas Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201, 1995. URL https://aclanthology.org/J95-2002/.

Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A Saurous, and Yoon Kim. Grammar prompting for domain-specific language generation with large language models. *Advances in Neural Information Processing Systems*, 36:65030–65055, 2023.

Gwen Yidou Weng, Benjie Wang, and Guy Van Den Broeck. TRACE back from the future: A probabilistic reasoning approach to controllable language generation. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 66617–66632. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/weng25b.html.

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

Gwen Yidou-Weng, Ian Li, Anji Liu, Oliver Broadrick, Yuchen Cui, Guy Van den Broeck, and Benjie Wang. Learning tractable distributions of language model continuations. *arXiv preprint arXiv:2511.16054*, 2025.

Gwen Yidou-Weng, Edward Sun, Tianyi Ma, Metin Alp Dogan, Benjie Wang, Allen Peng, Guy Van den Broeck, and Yuchen Cui. Meta-Ctrl: Guaranteed plan generation by decoupling syntactic and semantic constraints, 2026. URL https://arxiv.org/abs/2608.22149.

Honghua Zhang, Po-Nien Kung, Masahiro Yoshida, Guy Van den Broeck, and Nanyun Peng. Adaptable logical control for large language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 115563–115587. Curran Associates, Inc., 2024. doi: 10.52202/079017-3670. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/d15c16cf5619a2b1606da5fc88e3f1a9-Paper-Conference.pdf.

Honghua Zhang, Benjie Wang, Marcelo Arenas, and Guy Van den Broeck. Restructuring tractable probabilistic circuits. In Yingzhen Li, Stephan Mandt, Shipra Agrawal, and Emtiyaz Khan (eds.), *Proceedings of The 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pp. 2566–2574. PMLR, 03–05 May 2025. URL https://proceedings.mlr.press/v258/zhang25f.html.

Zhicheng Zhou, Jing Li, Suming Qiu, Junjie Huang, Linyuan Qiu, and Zhijie Sun. Deepjsoneval: Benchmarking complex nested json data mining for large language models. *arXiv preprint arXiv:2509.25922*, 2025.

Carme Álvarez and Birgit Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107(1):3–30, January 1993. ISSN 0304-3975. doi: 10.1016/0304-3975(93)90252-o. URL http://dx.doi.org/10.1016/0304-3975(93)90252-O.

A Proofs for Section 3

Throughout, $\varepsilon = \min_z p(\mathrm{EOS} \mid z) > 0$, and every run-sum below is a string mass, because a deterministic automaton gives every string at most one run.

Algorithm 2 The horizonless algorithm in full, with the cost of each line and $\omega \leq 3$ the matrix multiplication exponent.

Offline, once per automaton

1: build T_ι, C_γ, R_γ, $\mathbf{e}$ from the HMM and the automaton
2: $\mathbf{W} \leftarrow 0$
3: **for** $j = 1, \ldots, j^*$ **do** $\quad j^* = \left\lceil \ln(1/\delta) / \ln \frac{1}{1-\varepsilon} \right\rceil$ for error δ (Theorem 3.1)
4: $\quad \mathbf{W} \leftarrow I + T_\iota \mathbf{W} + \sum_\gamma C_\gamma \, \mathbf{W} R_\gamma \mathbf{W}$ $\quad O(p\,(hs)^\omega)$
5: **end for**
6: store $\mathbf{B}_\gamma \leftarrow \mathbf{W} R_\gamma$ for each γ, and $\mathbf{b}_0 \leftarrow \mathbf{W}\mathbf{e}$ $\quad O(p\,(hs)^2)$ memory

Online, after each generated token, with cache $\mathbf{v}_0 = \mathbf{b}_0$, $\mathbf{v}_i = \mathbf{B}_{\gamma_i} \mathbf{v}_{i-1}$ for the pending $\gamma_1 \cdots \gamma_k$, and $\mathbf{v}'$ the cache top after a candidate

7: $f_t \leftarrow$ HMM forward step on the new token; $q_t \leftarrow$ automaton step on it $\quad O(h^2)$
8: **if** the token is a call pushing γ: push $\mathbf{v}_{k+1} \leftarrow \mathbf{B}_\gamma \mathbf{v}_k$ $\quad O((hs)^2)$
9: **if** the token is a return: pop $\mathbf{v}_k$ $\quad O(1)$
10: score each candidate a by $p(a \mid x_{\leq t}) \left(\sum_{z,z'} f_t(z)\, p(a, z \to z')\, \mathbf{v}'(z', q') \big/ \sum_{z,z'} f_t(z)\, p(a, z \to z') \right)^c$ $\quad O(p\,(hs)^2 + h|\Sigma|)$
11: if the stack is empty, EOS keeps its model probability when $q_t \in F$ and is banned otherwise $\quad O(1)$

The chain, Equation 2–3. Fix $(u, \gamma_1 \cdots \gamma_k)$. An accepted completion cannot pop γ_k before first returning to depth k, so it decomposes uniquely as a well-matched block at depth k, the return popping γ_k, a block at depth $k-1, \ldots$, the return popping γ_1, a block at depth 0, and the end of the string in an accepting state. The positions of these returns are functions of the string (the alphabet fixes the depth after every prefix), distinct completions are distinct strings with one run each, and the segments compose by choosing the intermediate joint state and multiplying weights. Summing over all completions gives $\boldsymbol{\beta} = \mathbf{W} R_{\gamma_k} \cdots \mathbf{W} R_{\gamma_1} \mathbf{W}\mathbf{e}$.

The fixed-point equation. A well-matched string is empty, or begins with an internal token followed by a well-matched string, or begins with a call pushing some γ whose matching return pops γ and encloses a well-matched string, followed by a well-matched string. The matching return is determined by the string, so the decomposition is unique, and adding masses gives equation 5, which is equation 4 once the inverse exists. For a deterministic automaton the inverse exists under $\varepsilon > 0$. Every row of $T_\iota + \sum_\gamma C_\gamma X R_\gamma$ sums to at most $1 - \varepsilon$ for every $0 \leq X \leq \mathbf{W}$, because the two kinds of step, an internal token or a complete call–block–return, are prefix-free first moves, so their masses are of disjoint events and each emission withholds the end-of-string mass. (This is consistent with entries of $\mathbf{W}$ exceeding 1, as in the example below, because $\mathbf{W}$ sums strings that are prefixes of one another.)

Lemma A.1 (survival)**.** *For every z and $\ell \geq 0$, the total mass of* EOS*-free emission sequences of length ℓ from z is at most $(1-\varepsilon)^\ell$, and so is that of any sub-collection of them.*

Proof. With $m_\ell(z)$ that mass, $m_0 \equiv 1$ and $m_{\ell+1}(z) = \sum_{a \neq \mathrm{EOS}} \sum_{z'} p(a, z \to z')\, m_\ell(z') \leq (1-\varepsilon) \max_{z'} m_\ell(z')$, since by the normalisation of Section 2 the one-step masses from z sum to $1 - p(\mathrm{EOS} \mid z) \leq 1 - \varepsilon$, with ε a minimum over all states, so $m_\ell \leq (1-\varepsilon)^\ell$ by induction. A sub-collection only removes nonnegative terms. □

Lemma A.2 (iterates contain the short blocks)**.** *Let $\mathbf{W}^{(0)} = 0$ and $\mathbf{W}^{(j+1)} = I + T_\iota \mathbf{W}^{(j)} + \sum_\gamma C_\gamma \mathbf{W}^{(j)} R_\gamma \mathbf{W}^{(j)}$. Then $\mathbf{W}^{(j)}(u, u')$ is the mass of a set of well-matched blocks from u to u', each counted once, that contains every block of fewer than j tokens. The iterates increase entrywise to $\mathbf{W}$, the least nonnegative solution of equation 5.*

Proof. Give each block the height of its unique decomposition. The empty block has height 1, an internal token followed by a block w has height $1 + h(w)$, and a call–block w_1–return followed by a block w_2 has height $1 + \max(h(w_1), h(w_2))$. By induction on j, $\mathbf{W}^{(j)}$ is the mass of the blocks of height at most j, since the three terms of the update build exactly these and uniqueness of the decomposition counts each once. Height is at most one more than the number of tokens, so every

block of fewer than j tokens has height at most j. Every block has finite height, so the iterates increase to $\mathbf{W}$. Any nonnegative solution dominates every iterate by the same induction, so $\mathbf{W}$ is the least one. □

Proof of Theorem 3.1. All operators are nonnegative and $\mathbf{W}^{(j)} \leq \mathbf{W}$, so $\beta^{(j)} \leq \beta$. A completion counted by β but not by $\beta^{(j)}$ contains a block that $\mathbf{W}^{(j)}$ misses, hence by Lemma A.2 a block of at least j tokens, before the string ends. Completions are disjoint EOS-terminated events, all inside the event "no EOS among the first j emissions", whose probability is at most $(1-\varepsilon)^j$ by Lemma A.1. This is a survival form of the bound of Etessami & Yannakakis (2009, Thm. 3.1) on the iterates. Error δ therefore needs $j = O(\varepsilon^{-1}\log(1/\delta))$ rounds. For $\mathbf{W}$ itself the same argument gives only $0 \leq \mathbf{W} - \mathbf{W}^{(j)} \leq \varepsilon^{-1}(1-\varepsilon)^j$, because the entries of $\mathbf{W}$ sum non-disjoint strings (a string and its extensions all count) and the tail over lengths is a geometric series. □

Remark: when the model need not end. Equation 5 is a monotone system of the kind studied for recursive Markov chains (Etessami & Yannakakis, 2009), and Newton's method converges faster on such systems (Esparza et al., 2010; Pivoteau et al., 2012). When $\varepsilon = 0$ no bound in j alone exists. For the fair-coin Dyck system (call and return each $\frac{1}{2}$, no internals) the iteration $\mathbf{W} \leftarrow 1 + \mathbf{W}^2/4$ from 0 converges to 2 with error $e_{j+1} = e_j - e_j^2/4$, of order $4/j$.

B THE SAME SUMMARY UNDER A BUDGET

Proof of Corollary 3.3. The chain and equation 5 use only sums, products and distributivity (segments compose by choosing the intermediate state and combining costs), so they hold verbatim with $\min$ for the sum, $+$ for the product, cost 1 per transition of positive weight, $+\infty$ for zero-weight entries, cost 0 for EOS, and the identity I read as 0 on the diagonal and $+\infty$ off it. Then $\mathbf{W}(u,u')$ becomes the length of the shortest well-matched block from u to u', R_γ adds 1 for the return, $\mathbf{b}_0$ is the shortest depth-0 continuation ending in an accepting state, and $\beta(u,\sigma)$ is the length of the shortest completion. "Some completion has length at most r" equals "$\beta \leq r$" exactly, the minimiser being the witness. The round count j^* of Algorithm 2 is specific to probabilities. Offline the min-plus fixed point is a system of superior functions, monotone and at least the maximum of their arguments because lengths are nonnegative, so Knuth's generalisation of Dijkstra's algorithm (Knuth, 1977) solves it in polynomial time, each summary item finalised once. A backward shortest-path formulation does not suffice, since the call–block–return term is a hyperedge. Shortest completions can be exponentially long in s and p, so values are stored in binary. Online, one min-plus matrix–vector product per call and a lookup per return, as in Algorithm 1. □

The budgeted weights. Attaching x to every token turns equation 4 into $\mathbf{W}(x) = \big(I - xT_\iota - x^2\sum_\gamma C_\gamma \mathbf{W}(x) R_\gamma\big)^{-1}$, whose coefficient $\mathbf{W}_\ell$ is the weight of the well-matched blocks of exactly ℓ tokens. Since $\mathbf{W}(x)$ solves a polynomial system, each entry is an algebraic power series (Chomsky & Schützenberger, 1963; Stanley, 1999), with $\mathbf{W}_0 = I$ and $\mathbf{W}_\ell = T_\iota \mathbf{W}_{\ell-1} + \sum_\gamma C_\gamma \sum_{j=0}^{\ell-2} \mathbf{W}_j R_\gamma \mathbf{W}_{\ell-2-j}$, a block starting with an internal token, or with a call, an inner block of unknown length j, its return and a remainder. Evaluating the recurrence directly costs $O(n^2 p\,(hs)^3)$, the unknown split being the quadratic. Divide and conquer on the series splits $\mathbf{W}(x)$ into a low and a high half and solves the high half from a linear series equation whose right-hand side comes from the low half. The linear equation splits the same way, since its low half is the same equation to half the order. Each level therefore costs $O(1)$ truncated products, $\mathcal{T}(m) = 2\mathcal{T}(m/2) + O(p\,(hs)^\omega M(m))$ with $M(m)$ the cost of multiplying two series of length m, and all coefficients to order n cost $O(n\log^2 n \cdot p\,(hs)^\omega)$. Online, the chain becomes a chain of truncated series. An internal token shifts the budget index, a return pops, a call pushing γ costs $(hs)^2$ scalar convolutions of length at most n, $\tilde{O}(n(hs)^2)$ with fast multiplication, and candidate calls share that convolution per stack symbol. Over a string with $\Theta(n)$ pushes the total is $\tilde{O}(n^2(hs)^2)$, while strings with bounded nesting depth decode in $\tilde{O}(n)$ once ledgers are memoised by stack content, because a ledger depends only on the stack, of which there are at most $\min(n, \sum_{d\leq D} p^d)$ distinct ones, a constant in n but exponential in D. Any algorithm that keeps exact per-level length ledgers writes $\Omega(n^2)$ entries on adversarial nesting, so within this design the online bound is tight. Whether

an unrestricted $\Omega(n^2)$ holds is open, the gap being $[\Omega(n), \tilde{O}(n^2)]$. For comparison, Marzouk & de la Higuera (2022) compute the weight of the accepted strings of one length for an unambiguous grammar by a forward pass that covers every length up to n, at their stated cost $O(|G|\, h^3 n^2)$. They sketch a prefix variant without giving it, so conditioning on a new prefix reruns the pass, again quadratic per token. The circuit for the grammar alone is cubic per length (Zhang et al., 2025, Thm. 4.2), before its product with the model. Two contrasts are worth recording. The n-dependence belongs to the fixed-length question, not to the summary, since the "at most r" question of Corollary 3.3 has no n anywhere. And an *exact*-length mask is harder than a budget mask. The set of completion lengths is ultimately periodic, but its threshold can be exponential, since the shortest completion alone can be exponentially long, so an explicit table can need exponential size.

C Coverage: proof of Theorem 3.2

Precise hypotheses. Unambiguity cannot be dropped, because counting the strings of an NFA exactly is span-L-complete, hence #P-hard under Turing reductions (Álvarez & Jenner, 1993), although it can be approximated (Arenas et al., 2021). After the normalisation below, the automaton has no ε-moves other than deterministic pops. Every run of a string reaches the same stack depth after every prefix. Every accepted string has exactly one accepting run.

Final-state acceptance. Section 3 accepts when EOS arrives at $q \in F$ with the stack empty. A deterministic pushdown automaton may instead accept in a final state with symbols still on the stack. For example, $\{a^i b^j : i \geq j \geq 1\}$ is accepted by final state but never with an empty stack. This needs one change to the chain and none to $\mathbf{W}$. A completion from depth k either ends at depth k, a block followed by EOS in an accepting state, or first pops γ_k and then completes from depth $k-1$. The two cases are disjoint, so $\boldsymbol{\beta}_k = \mathbf{We} + \mathbf{W} R_{\gamma_k} \boldsymbol{\beta}_{k-1}$, with the same $\mathbf{B}_\gamma$. The added term is the stored $\mathbf{b}_0$, so a push in Algorithm 2 becomes $\mathbf{v}_{k+1} \leftarrow \mathbf{b}_0 + \mathbf{B}_\gamma \mathbf{v}_k$, one vector addition. Theorem 3.1 and Corollary 3.3 carry over unchanged. In min-plus the addition becomes a minimum with $\mathbf{b}_0$. Every machine in our experiments accepts with an empty stack, so no reported number changes.

Proof. Let the automaton have no ε-moves other than deterministic pops, let every run of a string reach the same stack depth after every prefix, and let every accepted string have exactly one accepting run. (1) The push/pop schedule of a string is then a function of the string, computable online by simulating any one live run, so "a segment that returns to its depth" is a property of the string and the split into levels of App. A is the same for every run. (2) A pushdown transition reads the stack top, so the level-local state is (z, q, X) with X the top symbol. Carrying the top in the state makes every transition a function of the state and the token, a push buries X and opens a level, and the fold at the matching pop ties the push to the pop through the buried symbol, exactly as C_γ and R_γ do for an NWA. The joint space has dimension $hs|\Gamma|$, and the summary is defined on it. (3) With one accepting run per accepted string, run-sums over complete runs are string masses. A deterministic automaton has a single run on every prefix, so Algorithm 1 applies with the changes below. For a deterministic pushdown automaton the string fixes the entire run, and its ε-moves can be made pop-only. An ε-move is a function of the state and the top symbol, both from finite sets, so an ε-computation that returned to the same pair with a strictly deeper stack would repeat forever. Infinite ε-computations can be excluded (Hopcroft & Ullman, 1979, Lemma 10.3), so the net number of pushes in any ε-computation is at most $|Q||\Gamma|$, and each ε-push can be folded onto the following input transition. A token then pushes a bounded number of symbols and opens a bounded number of levels. Pops are unbounded per token but never outnumber pushes, so each token costs amortised $O(1)$ chain steps. An ε-pop is deterministic, so the push/pop schedule stays a function of the string and each ε-pop is one more lookup in the chain. Amortisation does not cover a candidate that is scored but never emitted. For candidates, the landing level and state of the ε-pop cascade are tabulated per open level at push time ($O(s)$ per push), so each candidate costs $O(1)$. This covers every deterministic context-free language, i.e. the LR(k) languages (Knuth, 1965). □

D Results on every benchmark

This appendix reports every benchmark with the full set of arms, including the mask alone. DeepJSONEval and RNA are the benchmarks of Section 4. EAI and CommonGen are built on finite

automata and run inside the authors' own decoders, and SMILES is built on a grammar with genuine nesting.

D.1 DEEPJSONEVAL

Tab. 5 places every arm on the benchmark's full leaderboard.

	Format	Detailed	Strict
Claude Sonnet 4	99.05	90.73	57.90
Magistral Medium 2506	98.10	90.36	59.81
DeepSeek R1 0528	97.90	89.57	57.33
Gemini 2.5 Pro	97.52	89.00	56.19
Qwen3 235B A22B	97.14	88.33	56.19
DeepSeek R1 Distill Llama 70B	95.44	88.15	58.29
Magistral Small 2506	93.71	85.75	53.52
Qwen3 30B A3B	93.71	84.66	52.76
Hunyuan A13B	93.33	83.11	48.33
Llama 4 Maverick	92.19	84.17	54.48
Qwen3 14B	91.81	83.39	51.24
Qwen3 32B	87.81	80.18	50.67
Llama-3.1-8B-Instruct, unconstrained	79.40	72.22	44.40
+ rejection, best of 8	85.20	76.55	45.80
+ local hard mask	95.00	85.95	47.60
+ XGrammar 0.2.7	97.40	88.31	50.00
+ horizonless weights only	99.00	89.06	49.60
+ budget mask only	100.00	89.35	49.60
+ mask and weights (ours)	**100.00**	89.78	50.40

Table 5: DeepJSONEval leaderboard (Table 2 of Zhou et al. (2025)) with our rows, on the full benchmark for published models and our 500 instances for ours.

D.2 RNA SECONDARY STRUCTURE

Tab. 4 reports every RNA arm, including the mask alone. Sequential Monte Carlo uses a local one-step proposal with systematic resampling when the effective sample size falls below 24, and its standard errors average four draws per sequence first.

D.3 EAI: VIRTUALHOME ACTION SEQUENCING

The EAI task (Li et al., 2024) asks Llama-3-8B-Instruct for a plan of grounded actions that reaches a household goal. The constraint is a per-prompt finite automaton over actions with a budget of 20 actions, and the guide is a hidden Markov model fitted on the model's own plans. We decode the 342 prompts greedily and score with the official EAI evaluator over six evaluator seeds. The standard error is the evaluator's hash-order noise, and the evaluator drops 37 prompts whose gold plan fails its own goal test. The published method is Meta-Ctrl (Yidou-Weng et al., 2026). Tab. 6 reports task success.

- *Exact reproduction.* With the empty stack, our budgeted lookahead is their backward recursion. Over 320 million table entries on three production-scale machines the largest disagreement is $4.3 \cdot 10^{-14}$ nats with no support mismatch, and decoded inside their untouched decoder all 342 plans are byte-identical to their published trained arm.
- *Horizonless weights with the budget mask.* Replacing their length-indexed table by the horizonless weights under the budget mask of Section 3.2 (Corollary 3.3) again gives all 342 plans byte-identical, while storing one matrix instead of 21, which is 1.2 MB against 24.8 MB per prompt at the median and 65 MB against 1.34 GB at the largest.

Each horizonless value was solved to a relative change below 10^{-9} (11 to 34 rounds), and on the three audited machines no entry falls below the exact 20-action budgeted table, which a converged horizonless value never can.

	Task SR	Exec SR
reachability mask only (Meta-Ctrl)	79.0 ± 0.4	84.6
uniform guide (Meta-Ctrl)	87.5 ± 0.5	93.8
trained guide, budgeted table (Meta-Ctrl)	89.7 ± 0.5	96.1
ours, budgeted lookahead	89.7 ± 0.5	96.1
ours, horizonless weights + budget mask	89.7 ± 0.5	96.1

Table 6: EAI, VirtualHome action sequencing, 342 prompts, greedy decoding, official evaluator over six seeds.

D.4 COMMONGEN

Each CommonGen instance (Lin et al., 2020) gives three to five concepts, and GPT2-large (Radford et al., 2019) must write a sentence that uses all of them. The constraint is Ctrl-G's finite automaton over the concepts' token patterns, followed by a final period, and the guide is Ctrl-G's published HMM with 32,768 hidden states. We decode the 993 development concept sets with Ctrl-G's beam decoder, replacing only its length-indexed lookahead by our horizonless weights under the budget mask of Section 3.2, and score with the benchmark's text metrics and with Ctrl-G's own acceptance check (Tab. 7). All arms, including our run of Ctrl-G's decoder, forbid the end-of-sequence token while the automaton is in a non-accepting state. Without this gate a sentence can end before the constraint is met.

- *Nearly the same quality, no length index.* Our run of Ctrl-G's decoder matches its published row. With the horizonless weights the selected sentence is identical on 901 of 993 instances, and BLEU is 0.7 lower (paired bootstrap, 95% interval $[-1.2, -0.2]$). The lookahead needs 33 times less storage, one matrix instead of one per remaining length.
- *The budget mask guarantees termination.* Ctrl-G's decoder leaves one sentence unterminated at the 32-token cap, a repetition loop. Under the budget mask none is.

Each horizonless value converged to a relative change below 10^{-9} in 536 to 674 rounds. The mask's distances equal an independent breadth-first search on every row. Ctrl-G's check counts a concept as present when its token pattern appears, so "pin" is satisfied by "pinched". A stricter whole-word check gives 95.2% for Ctrl-G's decoder and 95.3% for ours.

	BLEU-4	ROUGE-L	CIDEr	SPICE	satisfied
Ctrl-G, published	32.1	45.2	16.0	30.8	100%
Ctrl-G decoder, our run	32.4	45.2	16.4	31.9	99.9%
budget mask only	1.0	16.7	0.2	9.9	100%
horizonless weights + budget mask (ours)	31.7	44.9	16.3	31.9	100%

Table 7: CommonGen development set, 993 concept sets, GPT2-large with Ctrl-G's 32,768-state guide. Satisfied: accepted by Ctrl-G's automaton and ended with the final period.

D.5 SMILES ACRYLATE GENERATION

Scope. We run one benchmark task, SMILES generation for the *acrylates* monomer class.

The task, from Wang et al. (2023) and used again by CARS (Parys et al., 2025), asks an 8B model for a molecule of a named class in SMILES notation, given every gold exemplar of that class in the prompt and reshuffled on every sample. A run is 100 unique non-exemplar completions. CARS's grammar is not released, and no released decoder for this task can apply a constraint to a local model's next-token scores, so we implement both. The decoding setup follows CARS's paper, the prompt is their standard template with its trailing space removed, and every rule of the constraint comes from the prompt or public chemistry. We use Llama-3.1-8B-Instruct at $T = 1.0$ with a 256-token budget, three seeds, and the benchmark's own scorer. The model and checkpoint commit match the published row exactly (`meta-llama/Llama-3.1-8B-Instruct` at `0e9e39f`), as do temperature, top-p, top-k, the token budget, the stop rule, the 100-unique target, the sample cap and the three metric definitions.

The constraint is nested, and the budget makes termination structural. The machine is a nested-word automaton over characters, in which parentheses are the call and return tokens and ring-bond digits are obligations that must be discharged. The arm reweights each token by the horizonless lookahead with exponent $c = 0.1$, chosen on development prompts, and the budget mask of Section 3.2 admits a token only when an accepting completion still fits in the tokens that remain, so *the arm cannot run out of budget mid-molecule*. That matters for the metric rather than for elegance, because the benchmark's sampling loop scores any non-empty, non-duplicate completion with no termination check, so a decoder that can be cut off mid-string is scored on those fragments. Across 986 draws, none was cut off.

acrylates, 100 unique molecules $\times$ 3 seeds	validity	in class	diversity
CARS, published	0.93	0.92	0.81
budget mask only	1.000	0.997	0.82
horizonless weights + budget mask (ours)	**1.000**	**1.000**	0.80
restricted to gold-length molecules ($\leq$ 40 chars)	1.000	1.000	0.72

Table 8: Every draw is scored and none is discarded. The mask-only row is one run. For our row, in two identical runs, validity and membership are 1.000 on every seed, and diversity moves by up to .022 between runs. Validity is RDKit (Landrum et al., 2006) parseability and *in class* is the benchmark's membership predicate.

What the machine is built from. Every rule comes from one of two places, the prompt the model is shown or public chemistry. The motif the machine requires, an ester of acrylic acid written `C=CC(=O)O` and its three equivalent spellings, is the definition of an acrylate and follows from the class name in the prompt. The SMILES grammar, ring-closure pairing and the permitted-valence table are standard. The remaining parameters are read off the exemplars in the prompt and are in every case no tighter than what those exemplars exhibit:

parameter	machine	prompt exemplars
atom alphabet	N, O, P, S, F, I, B, C, Br, Cl	use only C, O, F; machine is a superset
ring labels	$\{1, 2\}$	use exactly $\{1, 2\}$
minimum ring span	2 atoms	span 4–5 atoms; machine is looser
simultaneously open rings	5	reach 2; machine is looser

Nothing in the constraint encodes a fact unavailable at generation time, since each parameter is either taken from the prompt or strictly more permissive than the prompt requires.

Checking that the constraint covers validity. A constraint built this way need not be exact. It can admit strings that are not molecules. We therefore iterate the machine against RDKit until no accepted string is rejected, and treat that as the coverage test for the construction rather than as a source of rules. The rules it surfaces are themselves public chemistry. A ring bond that duplicates a bond the chain has already made is not a molecule. So what the iteration provides is confidence that the enumeration is complete, not new information about the target.

Validity is measured, not derived. The machine's language is not contained in the scorer's, so nothing in the construction forces a parseable molecule. The entry in Tab. 8 is an empirical result. Pooled over every acrylate molecule this implementation has generated, $1{,}614/1{,}614$ distinct molecules are valid (Wilson 95% lower bound 0.9976). The machine also accepts $32/32$ of the class's gold molecules, so the language being scored is not narrower than the class.

Diversity is not a claim. 9.4% of draws reach the cap and are closed there. These longer strings raise fingerprint diversity, which restricted to gold-length molecules is 0.72, below the gold set's 0.754. We therefore claim validity and membership only.

The published grammar is not available. The grammar behind the published row is not the released grammar file. Its authors describe theirs as enforcing the required functional group, the

released file does not require it, and their own reported sampling rates imply an acceptance rate an order of magnitude below the released file's. Our machine sits between the two, accepting 72.3% of unconstrained draws. The comparison in Tab. 8 is therefore between two methods under different constraints on a matched model and matched decoding parameters, and is bounded by an artefact that is not publicly available.

E EXPERIMENTAL DETAILS

Setup shared by the two benchmarks. The language model is Llama-3.1-8B-Instruct in bf16 on DeepJSONEval and a 38.1M-parameter character-level GPT-2 (12 layers, 8 heads, width 512, vocabulary 11, context 604) trained on the task on RNA, both with greedy decoding. On DeepJSONEval the guide is a per-prompt HMM with $h = 32$ hidden states fitted by EM on 16 unconstrained samples of the model for the same prompt, at a median of 42 s per instance, almost all of it sampling. Per-prompt guides reach 1.84 nats per token on held-out continuations, against 7.76 for one guide trained on the whole corpus (24 development instances). On RNA it follows Yidou-Weng et al. (2025) and is one HMM with $h = 32$ shared across molecules, whose initial distribution at each position is produced by a small network reading the base model's hidden state. The HMM and the network are trained jointly by gradient descent on the likelihood of the model's continuations, from 2,340 unconstrained samples over 780 molecules with invalid samples kept. Unlike that work, it also trains the HMM parameters, since a frozen HMM caps the fit. The exponent c is fitted on a disjoint development split. On DeepJSONEval it is fitted as the slope of the model's log-lookahead, estimated by sampling, on the guide's over ten development instances ($c^* = .123$, 95% interval $[.100, .146]$ clustered by instance, about three hours on one GPU, once per benchmark). On RNA it is fitted by a grid on 20 development molecules ($c = .1$). On DeepJSONEval, rejection sampling draws eight samples at the model's shipped sampling settings ($T = .6$, top-$p = .9$), keeps the most likely valid one and falls back to greedy if none is valid. On RNA it draws up to eight samples at temperature 1 and keeps the first valid one. If none is valid it returns the last draw, so its validity is not bounded below by greedy decoding. Every draw counts toward the LM calls.

Solving for $\mathbf{W}$. All machines were solved in fp64 by iterating equation 5 from zero, $\mathbf{W} \leftarrow I + T_\iota \mathbf{W} + \sum_\gamma C_\gamma \mathbf{W} R_\gamma \mathbf{W}$, with absolute tolerance 10^{-12} and relative tolerance 10^{-9}, the relative test stopping the loop, plus a support-stability check. Over the 500 DeepJSONEval machines the median was 101 rounds (51 to 182), none unconverged. This is the iteration of Theorem 3.1. It uses only additions and multiplications of nonnegative numbers, so every entry of $\mathbf{W}$ is computed to a small relative error, which matters because the mask reads exact zeros and the weights read logarithms. With small ε the a priori bound is loose at these round counts, so convergence is established by the stopping test and by the references below. Every reported solve was also checked, against an exact or finite reference or at two tolerances. RNA was checked against the exact span recursion, to $3.5 \cdot 10^{-15}$, EAI against the finite budgeted table, and DeepJSONEval, CommonGen and SMILES by solving at two tolerances.

DeepJSONEval. The 500 instances are a seeded random subset of the 2,100 and contain 344 distinct problems (156 rows repeat another row's input). Standard errors are clustered by problem. The schema is compiled per instance to a nested-word automaton whose stack symbol is the open container. The machine enforces every schema keyword present in the benchmark except non-string `enum`, which occurs on 21 of the 500 instances, all with the same numeric set. The keywords `format`, `const`, `uniqueItems`, `patternProperties` and all numeric and count bounds do not occur in the 500. Format 1.000 is therefore guaranteed by construction on 479 of the 500 (95.8%) and measured on the other 21, where a machine-accepted document can still violate the enumeration. The machine also imposes "no additional properties" at every object, which JSON Schema does not require by default, so it can reject documents the scorer would accept. We ran XGrammar 0.2.7 on the same model and prompt. Paired against it, our row differs by Format $+.026$ $[+.011, +.041]$, Detailed $+.015$ $[+.004, +.025]$ and Strict $+.004$ $[-.013, +.021]$. Eleven of XGrammar's thirteen Format failures are schemas that list a required key absent from `properties`. XGrammar builds its grammar from `properties` and cannot emit the key, whereas our machine requires every required key before the object closes. The other two reach the cap unclosed. $\mathbf{W}$ occupies $8.2 \cdot 10^{-6} s^2$ GB at $h = 32$, which is 2.4 GB at the median machine and 38 GB at $s = 2{,}154$. Measured on 20 machines stratified in $\log s$ across the 500, three replicates each on an idle H200, the offline solve

takes 0.3 s at $s = 45$, about 4 s at the median machine and 162 s at $s = 2{,}154$. Gold documents have median 143, p90 225 and maximum 336 tokens. Our row emits median 201, p90 307. Under the 2,048-token cap, the unconstrained arm reaches the cap on 6 documents, XGrammar on 2 and the weights-only arm on 5, all left unclosed. Ours reaches it on none. On the 397 documents the unconstrained model already produced validly, our row minus unconstrained is Detailed $+.006$ $[-.002, +.013]$ and Strict $+.010$ $[-.010, +.030]$, paired and clustered.

Tokens and storage on DeepJSONEval. Tokens are lifted to the automaton by composing each token's character walk from each state, which classifies it as internal, call or return. A token whose walk crosses two or more stack actions, such as `}]`, is banned at that state. Every bracket remains available as a one-character token, so the language is unchanged, and the banned tokens carry a median of 0.14 nats of the model's mass per document. $\mathbf{W}$ is formed once per instance, and $\mathbf{B}_\gamma = \mathbf{W}R_\gamma$ is never materialised. Each cached vector is one sparse application of R_γ followed by one product with $\mathbf{W}$, memoised per distinct stack, so the online cost beyond $\mathbf{W}$ is a handful of vectors: ten, 5.5 MB, on the largest instances, where $\mathbf{W}$ is 38 GB.

RNA. Held-out set of 215 sequences (length up to 300, first 20 skipped for development). The base model is trained on the bpRNA training split (Danaee et al., 2018) and evaluated on ArchiveII (Sloma & Mathews, 2016). Training keeps structures of length 20 to 300 over A, C, G, U without pseudoknots, and removes every sequence that also appears exactly in ArchiveII (1,681 of 47,158, 3.6%), which leaves 44,477 for training and 1,000 for validation. This removes exact overlaps only. bpRNA and ArchiveII share RNA families, and we do not hold out by family. The machine accepts Watson–Crick pairs and GU/UG with minimum loop size 3, pushing the position of each open bracket, so that a closing bracket is legal according to the nucleotide at that position. It was brute-force verified on GGGAAAUCCCAU (12 nt, with two negative controls) and UAAAAUU-GAACCGCCAGGAACC (22 nt, 1,535 structures). Because acceptance fixes the output length at the sequence length, the horizonless sum is finite by construction. The lookahead weight uses the same solve for $\mathbf{W}$ as the other benchmarks. The mask is the budget mask of Section 3.2, which here coincides with exact-length viability because $|y| = |x|$ is one of the machine's acceptance conditions (checked on 3.3 million reachable configurations). An independent fixed-length span recursion sharing no code reproduces all 215 outputs byte for byte (worst relative lookahead discrepancy $7 \cdot 10^{-16}$ to $3.5 \cdot 10^{-15}$ per molecule along the decoded trajectories, and $3.2 \cdot 10^{-16}$ over 1,804 reachable stacks on four molecules). An exact recursion cannot agree with a solve that stopped early, so this also certifies the fixed point was reached. The guide is fitted on 2,340 unconstrained samples (5.4% valid, invalid kept). Machines have $L + 1$ linear states and L stack symbols for a sequence of length L (median 116, maximum 295, the maximum set by the base model's context window). The horizonless solve of $\mathbf{W}$ on an idle GPU takes 1.7 s at $L = 77$, 4.2 s at $L = 118$, 21.8 s at $L = 236$ and 43.6 s at $L = 295$ (three replicates each, spread below 9%). Per token, the machine's own work, the viability test and the lookahead over all candidates, is 19 μs on CPU and the guide's belief update is 0.10 ms. End to end, decoding with mask and weights costs 3.94 ms per token against 3.50 ms unconstrained on the 38M model.

Dyck-2. The language has two bracket types with a content rule per type (only `a` inside square brackets, only `b` inside parentheses) and two obligations (every level must contain a token before it closes, and the string must contain an `a`). The linear state records whether an `a` has been seen, whether the current level has a token, and the current level's type, so $s = 12$. The stack symbol records the opened bracket's type together with the type of the level outside it, which the return restores, so $p = 6$. The machine is deterministic and trimmed. The language model is a hand-built memory-3 HMM that knows the content rule. Every hidden state ends the string with probability at least .02. The hidden state records the types of the innermost three open brackets, an overflow flag, and the kind of the last token, so it is exact to depth 3 and must guess the level type beyond. It has $h = 87$ states ($hs = 1{,}044$) and accepts 47% of its own output. The surrogate guides are HMMs with $h = 16$ fitted by EM on $2 \cdot 10^4$ samples of the model, with end probability floored at 10^{-3}. The exponent $c = .45$ is the slope of the true on the surrogate log-lookahead after centring both within each position, which removes an offset shared by all candidates. The uncentred slope plotted in Tab. 2 is .40. The exact lookahead agrees with the length series to $3 \cdot 10^{-11}$ and with brute force over the 18,909 accepted strings of length at most 10 to $6 \cdot 10^{-16}$. We score 10^5 samples from the exact conditional under each reweighting, with a 1,000-token cap.

Budgeted against horizonless. The two variants of Section 3.2 answer different questions and each is exact on its own. Decoding with the horizonless weights samples the conditional $P(\cdot \mid L)$ to floating point (KL $3 \cdot 10^{-16}$), and the budgeted decoder samples the truncated conditional $P(\cdot \mid L, |y| \leq r)$ with the same precision at every cap r. Measured from the truncated conditional, the budgeted decoder is therefore at KL zero and the horizonless weights at exactly $-\log P(|y| \leq r \mid L)$, the log-probability that the cap binds. The reverse direction is undefined, since the horizonless law puts mass beyond the cap. Tab. 9 gives the value measured along 10^5 samples of the truncated conditional, scoring both decoders' per-step conditionals. It matches the closed form from the summary to $8 \cdot 10^{-16}$ and the budgeted decoder sits at zero to the same precision. Because both decoders are exact samplers of their own law, the per-sample log-ratio is constant on the support and the measurement verifies that the implemented conditionals telescope to the closed form. It carries no statistical error. The value decays geometrically in r, to .0015 nats at $r = 64$, about 6.5 times the mean length. At a generous cap the two variants therefore nearly coincide, and at a binding one they differ by exactly $-\log P(|y| \leq r \mid L)$. At a batch of 10^5 particles the mask costs 4.1 μs per token and the horizonless weights 8.3 μs (one gather and one memoised $(hs)^2$ matrix–vector product per never-seen stack content).

cap r	8	12	16	24	32	64
KL, nats (mean length 9.8)	.521	.307	.192	.082	.036	.0015

Table 9: KL from the truncated conditional $P(\cdot \mid L, |y| \leq r)$ to decoding with the horizonless weights on Dyck-2, in nats, measured on 10^5 samples per cell. It equals $-\log P(|y| \leq r \mid L)$ to $8 \cdot 10^{-16}$, and the budgeted decoder of App. B is at zero to the same precision at every r. The reverse KL is undefined because the horizonless law puts mass beyond the cap.

Overhead. Per-token costs are paired measurements in one process on an idle GPU for DeepJSONEval, and batched CPU timings for Dyck-2. Offline solve times are for the median and the largest automaton.

	states (med. / max)	solve $\mathbf{W}$ (med. / max)	per token
Dyck-2	12	23 s	8.3 μs (mask 4.1 μs)
DeepJSONEval	539 / 2,154	4 s / 162 s	+1.67 ms
RNA	117 / 296	4.2 s / 44 s	19 μs + 0.10 ms guide

Table 10: Overhead: offline solve of $\mathbf{W}$ and cost per token. DeepJSONEval: milliseconds added by the mask and weights to 16.21 ms per unconstrained token, median of two instances ($s = 539$ and 2,154), three repetitions each. Dyck-2 uses batched CPU timings.