Stay Within Your Bounds: Distance-Guided Decoding for Guaranteed Context-Free Grammar Compliance

Anonymous ACL submission

Abstract

Grammar-constrained decoding helps large language models produce syntactically valid structured outputs, such as code, JSON, and SQL. For context-free grammars, many practical decoders enforce local prefix feasibility: each token must keep the current prefix extendable to some valid completion. Yet, under tokenizer–grammar mismatch and finite token budgets, feasible prefixes may still fail to reach acceptance. We propose a lookahead-guided decoding framework for context-free grammars based on pushdown automata. Offline, we compute bounded pushdown summaries with reachability labels and upper-bound distances to acceptance. Online, these estimates guide horizon-aware pruning and beam search. The resulting decoder is syntactically sound: every output is accepted by the target grammar. Experiments on JSON, SQL, and Linear Temporal Logic (LTL) show both consistent syntactic validity and improved completion quality over existing baselines.

1 Introduction

Large language models (LLMs) are increasingly used to generate structured objects such as programs, queries, and data records. In these settings, syntactic well-formedness is not merely a formatting preference: invalid outputs may be rejected by downstream parsers, violate interface contracts, or fail to execute. This has motivated a growing line of work on constrained and guided decoding, including grammar-constrained generation, type-constrained code generation, and structured generation engines for LLMs (Willard and Louf, 2023; Beurer-Kellner et al., 2024; Ugare et al., 2024; Dong et al., 2025; Park et al., 2025).

For context-free grammar constraints, many practical decoders enforce validity through *local prefix feasibility*. At each step, they compute which tokens keep the current prefix extendable to at least one valid completion and mask the remaining tokens. This mechanism is effective for preventing immediate syntax violations and has been implemented in several recent systems for structured generation (Willard and Louf, 2023; Beurer-Kellner et al., 2024; Ugare et al., 2024; Dong et al., 2025; Park et al., 2025; Chen et al., 2025). However, prefix feasibility is weaker than completed-output validity under a finite token budget. A prefix may remain locally admissible while moving toward configurations that require too many remaining tokens, additional structural commitments, or low-probability continuations. Thus, local masking preserves prefix validity, but it does not quantify progress toward acceptance.

A complementary line of work studies global control of autoregressive generation using automata, dynamic programming, probabilistic steering, or sampling-based inference (Zhang et al., 2023, 2024; Albinhassan et al., 2026; Loula et al., 2025; Lipkin et al., 2025). These methods show that constrained generation can benefit from reasoning about future completions. For context-free grammar constraints, however, such lookahead is difficult: the natural operational model is a pushdown automaton (PDA), whose configurations include an unbounded stack. This yields an infinite configuration space, making direct finite-state value computation lookahead inapplicable.

In this paper, we introduce SWYB, a lookahead-guided decoding framework for LLM generation under context-free grammar constraints. SWYB augments local grammar masking with distance-to-acceptance estimates computed over a pushdown representation. Offline, we construct bounded stack summaries using ideas from the reachability analysis of pushdown systems (Bouajjani et al., 1997) and weighted pushdown systems (Reps et al., 2003). These summaries associate abstract PDA configurations with reachability labels and token-level upper-bound distances to acceptance. Online, the

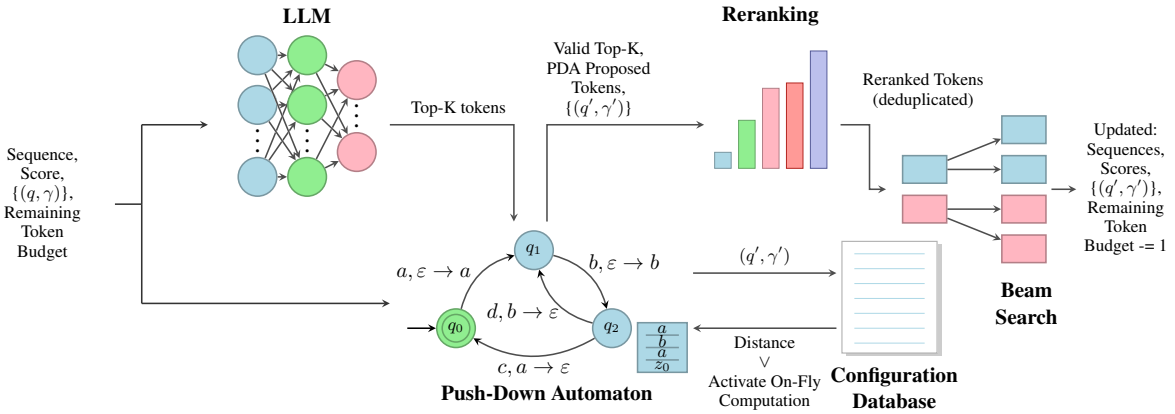

Figure 1: Overview of SWYB: Model top-K tokens are validated by tokenizer-aware PDA execution, filtered using reachability and distance-to-acceptance estimates, and combined with PDA-proposed candidates. Surviving tokens are reranked and used to update the beam state.

decoder uses these estimates for pruning and reranking in the beam search.

A practical difficulty is that LLM vocabularies are defined over subword tokens, whereas grammars are usually specified over characters, terminals, or lexical units. SWYB therefore includes a tokenizer-aware consumption procedure for matching model tokens against PDA transitions. At each decoding step, it combines model top-k tokens with automaton-proposed backup tokens, validates the resulting candidates through budgeted successor exploration, and uses distance information to favor continuations that make structural progress. The resulting decoder is syntactically sound: every completed output returned by the algorithm is accepted by the target PDA.

In summary, our contributions are:

- A sound pushdown lookahead mechanism based on bounded reachability and token-distances to acceptance.
- A tokenizer-aware decoding algorithm combining model top-k, automaton proposals, and budgeted PDA exploration.
- Experiments on JSON, SQL and LTL grammars showing consistent syntactic correctness and improved completion quality.

2 Related Work

Constrained decoding methods differ along three axes: the constraint class they support, whether they enforce prefix feasibility, and whether they use lookahead or remaining-budget information. Table 1 summarizes this landscape. SWYB targets a missing combination in practical systems: CFG/PDA constraints with prefix safety and budget-aware acceptance guidance.

Grammar masking. Grammar-constrained decoding restricts the output tokens of the language model so that the completion belongs to a given formal language. Practical systems such as Guidance and Outlines-style guided generation support structured generation through regular expressions, templates, and grammar-like specifications (Lundberg et al., 2024; Willard and Louf, 2023). Recent work focuses on efficient CFG enforcement under subword tokenization: DOMINO addresses tokenizer-grammar alignment (Beurer-Kellner et al., 2024), SynCode builds grammar-aware lookup structures (Ugare et al., 2024), XGrammar optimizes grammar execution and token-mask construction (Dong et al., 2025), GreatGramma studies tokenizer-grammar preprocessing (Park et al., 2025), and Pre^3 exploits deterministic pushdown automata for LR(1) grammars, which constitute a strict subset of CFGs (Chen et al., 2025).

These methods primarily enforce *local prefix feasibility*: at each step, the decoder checks whether a token keeps the current prefix extendable to at least one valid completion. This provides prefix safety, but not necessarily completed-output validity under a finite token budget. If decoding stops because the maximum number of tokens is reached before an accepting configuration, the returned output may still be syntactically invalid, even though every generated prefix was locally feasible (Ugare et al., 2024). Thus, local masking answers whether a prefix can still be completed, but not whether it

is close enough to acceptance within the remaining budget.

Global control. A complementary line of work studies controlled generation through global objectives, automata-based dynamic programming, or probabilistic inference. GeLaTo formulates tractable control for autoregressive generation with automata and dynamic programming (Zhang et al., 2023). Ctrl-G extends this view to adaptable logical control (Zhang et al., 2024). SEM-CTRL steers generation toward semantic constraints (Albinhassan et al., 2026). ABS enforces constraints that can be represented as a deterministic finite automaton (DFA) using the distance from the accepting states (Collura et al., 2025). Sequential Monte Carlo and adaptive weighted rejection sampling provide sampling-based control mechanisms for syntactic, semantic, or black-box constraints (Loula et al., 2025; Lipkin et al., 2025). These methods show the value of lookahead, but global control is much harder for CFGs than for regular constraints because pushdown automata have unbounded stacks.

Reachability analysis. The formal-methods literature provides symbolic tools for infinite-state pushdown systems. Reachability for pushdown automata can be represented using finite stack summaries and saturation procedures (Bouajjani et al., 1997). Weighted pushdown systems extend this idea by attaching weights to transitions and computing path quantities, with applications to interprocedural dataflow analysis (Reps et al., 2003). SWYB adapts this perspective to constrained decoding by computing bounded pushdown summaries and upper-bound distances to acceptance, then using them to guide token pruning and re-ranking.

Program constraints. Several methods specialize in constrained decoding for code or programming-language structure. Type-constrained decoding restricts generation using type information (Mündler et al., 2025), while correctness-guaranteed code generation studies decoding under stronger program-level specifications (Li et al., 2025). CFG-constrained decoding has also been extended to diffusion language models (Mündler et al., 2026). These approaches are complementary to ours: they enrich the constraint language or model class, whereas SWYB adds *sound* distance-guided lookahead to CFG/PDA decoding.

Method	Constraint	Prefix	Budget
Guidance (Lundberg et al., 2024)	CFG	partial	no
Outlines (Willard and Louf, 2023)	CFG	yes	no
DOMINO (Beurer-Kellner et al., 2024)	CFG	yes	no
SynCode (Ugare et al., 2024)	CFG	yes	no
XGrammar (Dong et al., 2025)	CFG	yes	no
GreatGramma (Park et al., 2025)	CFG	yes	no
Pre^3 (Chen et al., 2025)	DCFG	yes	no
GeLaTo (Zhang et al., 2023)	REG	yes	yes
Ctrl-G (Zhang et al., 2024)	REG	yes	yes
ABS (Collura et al., 2025)	REG	yes	yes
GenLM.control (Loula et al., 2025; Lipkin et al., 2025)	CFG/BB	yes	yes
SEM-CTRL (Albinhassan et al., 2026)	sem.	no	partial
Type-constr. (Mündler et al., 2025)	program	yes	no
SWYB	CFG	yes	yes

Table 1: Positioning of SWYB relative to constrained and controlled decoding methods. "Prefix" indicates whether generation is kept within the prefix closure of the constraint language. "Budget" indicates whether decoding uses a remaining-horizon, distance-to-acceptance, or related lookahead signal.

3 CFG-Constrained Decoding as a PDA Reachability Problem

We consider LLM decoding under a context-free grammar (CFG) constraint. Since CFG and pushdown automata (PDA) define the same class of languages, any CFG can be represented by an equivalent PDA, and conversely, any PDA recognizes a context-free language (Hopcroft et al., 2006). We therefore formulate the constraint through a PDA

$$\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_0, z_0, F), \tag{1}$$

where Q is a finite set of control states, Σ is the input alphabet whose elements correspond to CFG terminals, Γ is the stack alphabet, $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Q \times \Gamma^*$ is the transition relation, $q_0 \in Q$ is the initial state, $z_0 \in \Gamma$ is the initial stack symbol, and $F \subseteq Q$ is the set of accepting states. A PDA configuration is a pair $(q, \gamma) \in Q \times \Gamma^*$, where γ is the stack, written with its top on the left. We write

$$(q, \gamma) \xrightarrow{a} (q', \gamma') \tag{2}$$

when the PDA can consume $a \in \Sigma \cup \{\varepsilon\}$ and move from (q, γ) to (q', γ'). The language on Σ accepted by $\mathcal{P}$ is denoted as $\mathcal{L}(\mathcal{P})$.

Token-level configurations. LLMs generate tokens from a vocabulary V, whereas the PDA consumes terminals from Σ. A token may represent

a full terminal, several terminals, a strict prefix of a terminal, or a string crossing terminal boundaries. We therefore track an *extended configuration* $(q, \gamma, \rho) \in Q \times \Gamma^* \times \Sigma^*$, where ρ is the open prefix of a terminal currently being matched; $\rho = \varepsilon$ means that no terminal is open. Let $\mathcal{E} = Q \times \Gamma^* \times \Sigma^*$. Token consumption is represented as a transition function

$$\mathrm{Consume} : \mathcal{E} \times V \to 2^{\mathcal{E}}, \tag{3}$$

where $\mathrm{Consume}((q, \gamma, \rho), t)$ contains all extended configurations reachable by matching the decoded token string against CFG terminals and applying the induced PDA transitions.

Reachability and token distance. For a stack bound H, let $\mathcal{S}_H \subseteq Q \times \Gamma^{\leq H}$ be a bounded pushdown summary. Reachability in this summary follows the symbolic view of pushdown systems, in which predecessor sets can be computed by Pre^*-style saturation procedures (Bouajjani et al., 1997). We write $\mathrm{Reach}_H(q, \gamma)$ when (q, γ) can reach an accepting configuration within $\mathcal{S}_H$.

We also associate summarized configurations with a token-level distance estimate

$$d_H : \mathcal{S}_H \to \mathbb{N} \cup \{\infty\}. \tag{4}$$

This is inspired by weighted pushdown systems, which attach weights to pushdown transitions and compute path quantities (Reps et al., 2003). Here the cost unit is a *model token*. If $d^*_H(q, \gamma)$ denotes the true minimum token cost to acceptance within the bounded summary, we require

$$d_H(q, \gamma) \geq d^*_H(q, \gamma). \tag{5}$$

Thus, if the remaining token budget is K and $d_H(q, \gamma) > K$, then no accepting continuation of length at most K exists within the bounded summary.

Objective. We formulate CFG-constrained decoding as the problem of producing the highest-probability sequence accepted by the PDA within a finite token budget T:

$$y^\star \in \arg \max_{\substack{y \in \mathcal{L}(\mathcal{P}) \\ |y| \leq T}} \sum_{i=1}^{|y|} \log p_\theta(y_i \mid y_{<i}). \tag{6}$$

Exact optimization of this sequence-level objective is intractable in general: for autoregressive models, exact maximum-a-posteriori decoding is NP-hard, even under simple unary constraints (Pachet and Roy, 2026). Accordingly, SWYB treats Eq. (6) as an approximate search objective, guided by reachability and distance estimates.

Algorithm 1 SWYB decoding

Input: LLM p_θ, PDA $\mathcal{P}$, summary $\mathcal{S}_H$, distance d_H, max length T, beam width M
Output: Best completed output y found

```
B ← {(ε, 0, q_0, z_0, ε)}
for k = 0 to T − 1 do
    K ← T − k
    B' ← ∅
    foreach (y, ℓ, q, γ, ρ) ∈ B do
        if Accept(q, γ, ρ) then
            add (y, ℓ, q, γ, ρ) to B'
            continue
        form T_k^cand by Eq. (8)
        foreach t ∈ T_k^cand do
            compute C_t by Eq. (9),
            foreach (q', γ', ρ') ∈ C_t do
                score using Eq. (16)
                add (yt, s', q', γ', ρ') to B'
    B ← top-M beams in B'
return best accepting beam in B
```

4 Proposed Method

4.1 Overview

SWYB is a two-phase decoder for LLM generation under CFG constraints represented as PDAs. Offline, it builds the bounded summary $\mathcal{S}_H$ with reachability labels Reach_H and token-distance estimates d_H. Online, our beam search, with a width of M, maintains extended configurations (q, γ, ρ) and uses these labels to validate, prune, and rank token candidates. Algorithm 1 describes the online decoding loop, while Figure 1 summarizes the full pipeline. Each beam stores a prefix y, a score ℓ, and an extended configuration (q, γ, ρ). Given a totel token budget T, at step k, with the remaining horizon $K = T - k$, the decoder builds a candidate set, executes candidates through the PDA, filters out unreachable or over-budget successors, and keeps the top M scored extensions. Here $\mathrm{Accept}(q, \gamma, \rho)$ means that the PDA acceptance condition holds and no terminal is open, i.e., $\rho = \varepsilon$.

Offline and on-the-fly computation. Offline, before decoding begins, the grammar is translated to a PDA and SWYB caches $\mathcal{S}_H$ and d_H in a configuration database reused across prompts. Online, most queries are answered by lookup, while extended configurations absent from the configuration database are computed on the fly and stored in the configuration database itself, whilst those out of bounds or unreachable are discarded. Thus, repeated use of the same grammar amortizes the cost of the summary, while on-the-fly computation covers configurations not encountered during precomputation.

4.2 Candidate Tokens

At each step, SWYB combines model-led and automaton-led proposals. Let $\mathrm{TopK}_k(y)$ be the model top-k tokens for the current prefix y. The PDA also proposes tokens compatible with locally admissible continuations: Let $T_{\mathrm{prop}}(q, \gamma, \rho)$ be the set of tokens whose decoded string is compatible with at least one admissible continuation from the extended configuration (q, γ, ρ):

$$T_{\mathrm{prop}}(q, \gamma, \rho) \subseteq V. \tag{7}$$

For a given configuration (q, γ, ρ), the automaton-proposed tokens are obtained by considering every transition $(q, a, \gamma_{\text{top}}, q', \gamma') \in \Delta$ for which ρ is a prefix of a, where γ_{top} is the top element of γ. Let $a[|\rho| :]$ be the remaining suffix after removing ρ; if non-empty, its first token (according to the model's tokenizer) is included in $T_{\mathrm{prop}}(q, \gamma, \rho)$.

$$T_{\mathrm{prop}}(q, \gamma, \rho) \;=$$

$$\left\{ \mathrm{First}\big(\mathrm{tok}(a[|\rho| :])\big) \;\middle|\; \begin{array}{l} (q, a, \gamma_{\text{top}}, q', \gamma') \in \Delta, \\ \rho \text{ is a prefix of } a, \\ a[|\rho| :] \neq \varepsilon \end{array} \right\}$$

where $\mathrm{First}(t_1 t_2 \ldots) = t_1$ and $\mathrm{tok}(\cdot)$ are the model's tokenization functions. The resulting candidate set is

$$T_k^{\mathrm{cand}} = \mathrm{TopK}_k(y) \cup T_{\mathrm{prop}}(q, \gamma, \rho). \tag{8}$$

Thus, high-probability model tokens are checked first, while automaton proposals recover structurally useful tokens that may fall outside the model top-M.

4.3 Successor Filtering

For each candidate token t, we compute a viable successor set

$$C_t = \mathcal{C}_{B,H}(t, e, K) \subseteq \mathrm{Consume}(e, t), \tag{9}$$

where $e = (q, \gamma, \rho)$ and $\mathcal{C}_{B,H}$ explore at most B successors and retain only those whose current distance estimate witnesses a completion within the remaining token horizon:

$$d_H(q', \gamma') \leq K - 1. \tag{10}$$

Equivalently, successors with $d_H(q', \gamma') > K - 1$ are discarded. For every token with $C_t \neq \emptyset$, its lookahead distance is

$$D_{B,H}(t, e) = \min_{(q', \gamma', \rho') \in C_t} d_H(q', \gamma'). \tag{11}$$

Otherwise, the token t is discarded.

4.4 Distance-Guided Scoring

In addition to pruning, distance estimates are used to promote tokens that provide progress towards acceptance. Let $z_k(t)$ be the model logit for token t at step k, and define

$$T_k^{\mathrm{viab}} = \{u \in T_k^{\mathrm{cand}} : C_u \neq \emptyset\}. \tag{12}$$

Let $z_k^{\max} = \max_{u \in T_k^{\mathrm{viab}}} z_k(u)$. For a viable token t, we compute

$$r_k(t) = \frac{D_{B,H}(t)}{\max(1, K - 1)} \tag{13}$$

and the ramping coefficient

$$\alpha_k(t) = \alpha_{\min} + (1 - \alpha_{\min}) \min\{1, r_k(t)\}. \tag{14}$$

Tokens selected for promotion are pushed toward the best viable logit:

$$\widetilde{z}_k(t) = (1 - \alpha_k(t)) z_k(t) + \alpha_k(t) z_k^{\max}. \tag{15}$$

The successor beam score is then

$$s' = \ell + \log \mathrm{softmax}(\widetilde{z}_k)(t). \tag{16}$$

4.5 Soundness and Approximation

We distinguish syntactic soundness from search completeness. Soundness means that every completed output returned by SWYB is accepted by the target PDA. Search completeness would require exploring all valid completions within the token budget, including those needed to recover the highest-scoring accepted output. SWYB is sound but not complete, because it uses a bounded summary, a bounded successor exploration budget B, model top-k candidate truncation, distance filtering, and finite beam width M.

Theorem 4.1 (Syntactic soundness)**.** *Given a token budget T, assume that the PDA initial configuration (q_0, z_0) satisfies $d_H(q_0, z_0) \leq T$. Then every completed output returned by* SWYB *is accepted by $\mathcal{P}$ and a least one valid output is returned by* SWYB.

Proof. We prove by induction over decoding steps that every retained beam $(y, \ell, q, \gamma, \rho)$ at every step k is consistent with a PDA run from the initial extended configuration (q_0, z_0, ε) after consuming the decoded token sequence y and at a distance of at most $T - k$, where T is the total token budget.

Initially, the only beam is $(\varepsilon, 0, q_0, z_0, \varepsilon)$, so the invariant holds. Assume it holds at step k. A successor beam is added only after a token t is selected and at least one valid successor (q', γ', ρ') is taken from $C_t \subseteq \mathrm{Consume}((q, \gamma, \rho), t)$. This successor corresponds to at least one valid PDA execution of $\mathrm{Tok}(t)$ from (q, γ, ρ), and by the distance filtering (Eq. 10), the resulting successor beam is at a distance of at most $T - k - 1$. This shows heredity.

Thus, after T steps, $\mathrm{Accept}(q, \gamma, \rho)$ holds all completed beams, as there is at most a distance $T - T = 0$ from an accepting configuration. □

Theorem 4.1 is independent of the quality of the distance estimate: distances affect which valid prefixes are explored, but they cannot make an invalid token sequence valid. The converse need not hold. Since d_H is an upper-bound estimate, a successor with $d_H(q', \gamma') > K - 1$ may still admit a shorter completion not captured by the estimate. Thus, distance filtering may reduce completeness but not syntactic soundness.

5 Experiments

Models. We evaluate SWYB on structured generation tasks requiring syntactic validity under CFG using three instruction-tuned large language models that differ in size, family, and architecture: LLAMA-3.1-8B-INSTRUCT (Grattafiori et al., 2024), LLAMA-3.2-3B-INSTRUCT (Meta, 2024), and QWEN2.5-7B-INSTRUCT (Qwen et al., 2025)[1]. Reporting results on all three let us assess robustness across base architectures rather than tuning to a single model.

Benchmarks. We consider three structured generation domains: JSON, SQL, and Linear Temporal Logic (LTL). For JSON, we use the `json-mode-eval` dataset (NousResearch, 2024), which consists of 100 examples, and requires generating valid JSON objects adhering to a given schema, testing the model's ability to produce properly nested structures and correct data types. For SQL, we employ the Spider dataset (Yu et al., 2018), a cross-domain text-to-SQL benchmark where each example pairs a natural language question with a database schema, and the model must generate a syntactically valid SQL query that correctly retrieves the answer. We use the official validation split, which contains 1,034 examples. For LTL, we adopt the drone planning task from Pan et al. (2023), using their golden dataset comprising 6,185 examples. The model must generate LTL formulas that specify planning goals for drones, ensuring that each output strictly follows LTL syntax.

Baselines. We include two simple baselines: BASE LM and SAMPLE-VERIFY. BASE LM runs the unconstrained base model with a temperature of 0.8 for 10 independent generations per input (different random seeds). SAMPLE-VERIFY uses the same 10 raw generations, discards syntactically invalid ones, and then, for each of the 10 fixed seeds, randomly selects one surviving generation per seed (if any) to form 10 final runs. This follows the protocol of (Lipkin et al., 2025).

For comparison, we consider all available open-source constrained decoding methods whose public implementations support by default the grammar required by the task. Concretely:

JSON: Guidance (Lundberg et al., 2024), Outlines (Willard and Louf, 2023), XGrammar (Dong et al., 2025), SynCode (Ugare et al., 2024), and GenLM (Loula et al., 2025; Lipkin et al., 2025).

SQL: SynCode and GenLM. While XGrammar technically supports SQL it took up to 20 seconds per generated token and was therefore discarded.

LTL: While LTL is a CFG, currently, none of the public implementations natively support LTL grammar.

The following methods are excluded due to lack of runnable public code: DOMINO[2] (Beurer-Kellner et al., 2024), GreatGramma (Park et al., 2025), and SEM-CTRL (Albinhassan et al., 2026). Additional approaches (GeLaTo (Zhang et al., 2023), Ctrl-G (Zhang et al., 2024), ABS (Collura et al., 2025)) are restricted to regular grammars; Pre[3](Chen et al., 2025) handles only deterministic CFGs; and Type-

[1]Code will be released upon acceptance.

[2]A public repository exists but contains no executable implementation at the time of writing.

Method	Llama-3.1-8B-Instruct				Llama-3.2-3B-Instruct				Qwen2.5-7B-Instruct			
	Syn.	Schema	Time	Perp.	Syn.	Schema	Time	Perp.	Syn.	Schema	Time	Perp.
Base LM	63.7±15.9	57.1±14.4	1.74±0.0	1.05	56.4±9.8	49.4±8.7	**0.96**±0.1	**1.06**	93.6±0.8	87.4±0.8	**1.47**±0.0	**1.01**
Sample-Verify	92.0±0.0	80.9±0.9	17.38±0.0	1.08	80.0±0.0	67.1±1.9	9.65±0.0	1.22	96.0±0.0	88.3±0.5	14.71±0.0	1.04
Guidance	96.0	89.0	2.88	1.08	93.0	84.0	2.76	1.12	92.0	87.0	2.85	1.02
Outlines	94.0	90.0	3.49	1.07	93.0	90.0	2.04	1.12	95.0	89.0	3.62	1.02
XGrammar	93.0	85.0	**1.57**	1.45	95.0	88.0	**0.96**	1.38	93.0	85.0	1.60	1.80
SynCode	88.0	79.0	3.05	**1.02**	88.0	77.0	2.12	1.13	94.0	88.0	1.50	**1.01**
GenLM (NP=2)	90.8±1.9	86.6±1.8	3.77±1.9	1.05	91.4±1.3	87.5±1.4	3.93±1.1	1.09	93.6±0.5	89.0±0.0	2.93±1.9	**1.01**
GenLM (NP=5)	91.6±1.1	87.2±1.3	5.59±0.3	1.03	93.0±1.0	88.4±0.9	7.36±0.4	1.09	93.8±0.4	89.0±0.0	4.07±0.2	**1.01**
▷ SWYB	**100.0**	**100.0**	3.96	1.13	**100.0**	**100.0**	3.46	1.14	**100.0**	**100.0**	4.05	1.11

Table 2: JSON generation results across base models. Syn. denotes syntactic validity and Schema denotes schema validity. Best results within each model are shown in bold and second-best results are underlined. SWYB is ran with $\alpha = 0.25$ and 2 beams, and GenLM is ran with default configuration using 2 and 5 number of particles (NP). Syncode has two modes: Strict and Mask, here we only report Strict as it outperforms its counterpart all metrics. For

Method	Llama-3.1-8B-Instruct				Llama-3.2-3B-Instruct				Qwen2.5-7B-Instruct			
	Ex.	Syn.	Time	Perp.	Ex.	Syn.	Time	Perp.	Ex.	Syn.	Time	Perp.
Base LM	53.8±4.7	92.9±7.9	**0.87**±0.0	1.61	46.3±4.0	96.2±6.7	**0.53**±0.0	2.16	61.4±0.6	99.2±0.2	**0.88**±0.0	3.48
Sample-Verify	57.5±1.1	**100.0**±0.0	8.90±0.0	1.56	47.8±0.7	99.1±0.3	5.29±0.0	3.50	61.5±0.4	99.7±0.1	8.83±0.0	10.38
SynCode	43.2	90.7	3.31	3.70	39.6	98.1	2.01	2.47	43.2	90.7	3.31	3.70
GenLM (NP=5,5,4)	59.2±0.9	99.9±0.1	10.74±8.4	1.67	49.4±0.7	99.9±0.1	4.17±0.3	**1.38**	51.9±0.4	86.1±0.6	6.77±0.1	3.78
▷ SWYB	**61.1**	**100.0**	9.92	**1.22**	**51.5**	**100.0**	6.25	1.43	**62.8**	**100.0**	13.03	**2.10**

Table 3: SQL generation results across base models. Ex. denotes execution accuracy and Syn. denotes syntactic correctness. Best results within each model are shown in bold and second-best results are underlined. GenLM uses NP=5 for Llama models and NP=4 for Qwen. SWYB uses beam width 4; α is set to 0.25, 0.75, and 0.50 for Llama-3.1-8B, Llama-3.2-3B, and Qwen2.5-7B, respectively.

constr. (Mündler et al., 2025) works only for TypeScript. For each baseline, we report the mean and standard deviation over multiple seeds; deterministic methods are reported with a single run.

Metrics. For each domain, we measure both constraint satisfaction and generation quality.

JSON. We report *syntactic validity* (fraction of outputs parsable as JSON), *schema validity* (fraction conforming to the required schema, e.g., correct field names and types), *perplexity* (mean token-level perplexity of the generated sequences under the base LM, measuring fluency), and *average decoding time* (seconds per output, capturing computational overhead).

SQL. We report *syntactic correctness* (percentage of queries that parse according to the SQL grammar), *execution accuracy* (percentage that return the expected result when run on the target database), *perplexity*, and *decoding time*.

LTL. We report *syntactic correctness* (parser acceptance for LTL formulas), *task accuracy* (the percentage of generated LTL formulas whose corresponding Büchi automaton is equivalent to that of the ground-truth formula, verifying semantic correctness), *perplexity*, and *decoding time*.

Main findings. Across all three domains and base models, SWYB is the only method that achieves 100% syntactic correctness in every setting. On JSON, this also corresponds to 100% schema validity across all models. On SQL and LTL, SWYB improves task-specific accuracy while preserving syntactic validity. The cost is a moderate decoding-time overhead relative to lightweight local masking methods, but it is comparable to that of repeated sample-and-verify, which achieves lower performance in all domains.

5.1 JSON Generation

Table 2 reports results on `json-mode-eval`. SWYB is the only method that achieves perfect syntactic and schema validity across all three base models. Other constrained decoders generally improve over the Base LM on the two Llama models, but their gains are less consistent on Qwen2.5-7B, where the unconstrained model is already strong. SWYB maintains 100% validity in all settings, at a

	Llama-3.1-8B-Instruct				Llama-3.2-3B-Instruct				Qwen2.5-7B-Instruct			
Method	Acc.	Syn.	Time	Perp.	Acc.	Syn.	Time	Perp.	Acc.	Syn.	Time	Perp.
Base LM	22.1±3.3	98.8±0.4	**0.29**±0.0	1.31	9.6±3.3	94.6±3.1	**0.17**±0.0	1.57	30.9±0.5	98.4±0.4	**0.29**±0.0	1.61
Sample-Verify	22.2 ±0.3	**100.0** ±0.0	2.86 ±0.0	1.31	9.3 ±0.2	99.9±0.0	1.64±0.0	1.56	30.6 ±0.2	99.7 ±0.0	2.89 ±0.0	**1.07**
▷ SWYB	**29.6**	**100.0**	1.69	**1.23**	**22.1**	**100.0**	0.78	**1.27**	**31.6**	**100.0**	1.53	1.09

Table 4: LTL generation results across base models. Acc. denotes task accuracy and Syn. denotes syntactic correctness. Best results within each model are shown in bold. SWYB uses beam width 4; α is set to 0.75, 0.50, and 0.50 for Llama-3.1-8B, Llama-3.2-3B, and Qwen2.5-7B, respectively.

moderate runtime cost: it is slower than lightweight masking-based methods, but faster than Sample-Verify and competitive with GenLM at higher particle counts. Its perplexity remains close to the best baseline, suggesting that the guarantee does not come at the cost of substantially lower model likelihood.

5.2 SQL Generation

Table 3 reports results on Spider. SWYB achieves perfect syntactic correctness across all three base models and the highest execution accuracy in every setting. Sample-Verify also reaches perfect or near-perfect syntax but does not improve execution accuracy to the same extent, showing that syntactic filtering alone is insufficient for this task. SynCode variants underperform the Base LM in execution accuracy, while GenLM is competitive on the Llama models but degrades on Qwen2.5-7B-Instruct. In contrast, SWYB consistently improves both syntactic correctness and execution accuracy across architectures. Its runtime is comparable to Sample-Verify and GenLM on the Llama models, though it is slower on Qwen, and its perplexity remains the best or second-best in all settings. Overall, SWYB provides the strongest trade-off between syntactic guarantees, execution accuracy, and decoding cost.

5.3 LTL Generation

Table 4 reports results on the LTL dataset. From the table, we observe that SWYB consistently achieves perfect syntactic correctness (100%) across all three models, while also attaining the highest task accuracy in every setting. In comparison, Sample-Verify reaches perfect or near-perfect syntax but delivers substantially lower accuracy and often performs no better than the Base LM. Moreover, SWYB is faster than Sample-Verify and achieves better or comparable perplexity. These results highlight that SWYB not only guarantees syntactic correctness but also improves task accuracy and efficiency, offering a clear advantage over both the Base LM and Sample-Verify in the LTL setting.

6 Conclusion

We have introduced SWYB, a distance-guided decoding framework for guaranteed context-free grammar compliance in large language models.

Unlike prior grammar-constrained decoders that rely solely on local prefix feasibility, SWYB augments token-level validation with bounded pushdown summaries and upper-bound distance estimates to acceptance. This enables budget-aware pruning and reranking: configurations whose estimated completion cost exceeds the remaining token horizon are discarded; conversely, our beam search softly promotes tokens that make measurable structural progress toward acceptance.

Experiments on JSON, SQL, and LTL generation demonstrate that SWYB consistently achieves perfect syntactic validity across three diverse base models while also improving task-specific accuracy over existing methods, including Sample-Verify, SynCode, XGrammar, and GenLM. The method maintains competitive perplexity and decoding time, offering a favorable trade-off between guarantee and efficiency.

7 Limitations

Among the limitations are the stack height bound H and the exploration budget B. The parameter H determines the maximum stack height considered during offline precomputation. If H is set too low, some reachable configurations may be missing from the bounded summary; in that case, the decoder may need to compute distances online for configurations not present in the precomputed table. Although such online computation is not prohibitively expensive, it is slower than a simple lookup. The exploration budget B controls how many intermediate PDA configurations are explored. A large B increases runtime but enables more detailed lookahead. The impact of B depends

on the complexity of the PDA: for small grammars such as LTL, exhaustive exploration is perfectly feasible and incurs little overhead. Importantly, even with a minimal budget $B = 1$ and a sufficiently large H, the algorithm remains sound because soundness only requires knowing that at least one valid completion exists within the remaining token budget; detailed enumeration of all possible successors is not needed. The distance estimate d_H is a constructive upper-bound estimate within the bounded summary. A small value certifies that a completion within the remaining token budget is known, but a large value does not prove that no shorter completion exists. Finally, while SWYB empirically improves generation quality, it does not incorporate semantic control (e.g., domain-specific constraints). Adding such semantic guidance remains an interesting direction for future work.

8 Ethics Statement

Large Language Models (LLMs) were used in a limited manner for two specific purposes: (1) suggesting alternative phrasings for a few sentences (simple copy-editing), and (2) generating LaTeX code for tables and formatting. LLMs were not used for research design, analysis, or interpretation of results. All generated output was reviewed and verified by the authors, who assume full responsibility for the final content.

Acknowledgments

References

Mohammad Albinhassan, Pranava Madhyastha, and Alessandra Russo. 2026. Sem-ctrl: Semantically controlled decoding. *Transactions on Machine Learning Research*. J2C Certification.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2024. Guiding llms the right way: fast, non-invasive constrained generation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Ahmed Bouajjani, Javier Esparza, and Oded Maler. 1997. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of the 8th International Conference on Concurrency Theory*, CONCUR '97, page 135–150, Berlin, Heidelberg. Springer-Verlag.

Junyi Chen, Shihao Bai, Zaijun Wang, Siyu Wu, Chuheng Du, Hailong Yang, Ruihao Gong, Shengzhong Liu, Fan Wu, and Guihai Chen. 2025. Pre3: Enabling deterministic pushdown automata for faster structured llm generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 11253–11267. Association for Computational Linguistics.

Vincenzo Collura, Karim Tit, Laura Bussi, Eleonora Giunchiglia, and Maxime Cordy. 2025. Abs: Enforcing constraint satisfaction on generated sequences via automata-guided beam search. *Preprint*, arXiv:2506.09701.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. 2025. Xgrammar: Flexible and efficient structured generation engine for large language models. *Preprint*, arXiv:2411.15100.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

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

Lingxiao Li, salar rahili, and Yiwei Zhao. 2025. Correctness-guaranteed code generation via constrained decoding. In *Second Conference on Language Modeling*.

Ben Lipkin, Benjamin LeBrun, Jacob Hoover Vigly, João Loula, David R. MacIver, Li Du, Jason Eisner, Ryan Cotterell, Vikash Mansinghka, Timothy J. O'Donnell, Alexander K. Lew, and Tim Vieira. 2025. Fast controlled generation from language models with adaptive weighted rejection sampling. In *Second Conference on Language Modeling*.

João Loula, Benjamin LeBrun, Li Du, Ben Lipkin, Clemente Pasti, Gabriel Grand, Tianyu Liu, Yahya Emara, Marjorie Freedman, Jason Eisner, Ryan Cotterell, Vikash Mansinghka, Alexander K. Lew, Tim Vieira, and Timothy J. O'Donnell. 2025. Syntactic and semantic control of large language models via sequential monte carlo. In *The Thirteenth International Conference on Learning Representations*.

Scott Lundberg, Marco Ribeiro, and 1 others. 2024. Guidance. `https://github.com/guidance-ai/guidance`. Accessed: 2025-09-17.

Meta. 2024. Llama-3.2-3b-instruct. `https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct`.

Niels Mündler, Jasper Dekoninck, and Martin Vechev. 2026. Constrained decoding of diffusion LLMs with context-free grammars. In *The Fourteenth International Conference on Learning Representations*.

Niels Mündler, Jingxuan He, Hao Wang, Koushik Sen, Dawn Song, and Martin Vechev. 2025. Type-constrained code generation with language models. *Proc. ACM Program. Lang.*, 9(PLDI).

NousResearch. 2024. json-mode-eval. https://huggingface.co/datasets/NousResearch/json-mode-eval.

François Pachet and Pierre Roy. 2026. Hidden biases in conditioning autoregressive models. *Preprint*, arXiv:2604.07855.

Jiayi Pan, Glen Chou, and Dmitry Berenson. 2023. Data-efficient learning of natural language to linear temporal logic translators for robot task specification. *Preprint*, arXiv:2303.08006.

Kanghee Park, Timothy Zhou, and Loris D'Antoni. 2025. Flexible and efficient grammar-constrained decoding. *Preprint*, arXiv:2502.05111.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Thomas Reps, Stefan Schwoon, and Somesh Jha. 2003. Weighted pushdown systems and their application to interprocedural dataflow analysis. In *Proceedings of the 10th International Conference on Static Analysis*, SAS'03, page 189–213, Berlin, Heidelberg. Springer-Verlag.

Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. 2024. Syncode: Llm generation with grammar augmentation. *Preprint*, arXiv:2403.01632.

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

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *CoRR*, abs/1809.08887.

Honghua Zhang, Meihua Dang, Nanyun Peng, and Guy Van den Broeck. 2023. Tractable control for autoregressive language generation. *Preprint*, arXiv:2304.07438.

Honghua Zhang, Po-Nien Kung, Masahiro Yoshida, Guy Van den Broeck, and Nanyun Peng. 2024. Adaptable logical control for large language models. *Preprint*, arXiv:2406.13892.

A System specifications

All the experiments were run on a machine equipped with the following infrastructure: Intel® Xeon® Silver 4416+ CPU, 20 Cores, 40 Threads, 2.00/3.90 GHz; NVidia L40S GPU, 48 GB GDDR6, 18176 CUDA Cores, 142 RT Cores, 568 Tensor Cores.