StructureBench: A Unified Benchmark Suite for Multi-Scenario Structured Generation Tasks with On-Device Models

Xiaokun Xiong[1], **Zhengjie Xu**[1], **Junyi Chen**[2], **Shihao Bai**[3], **Ruihao Gong**[1*] and **Xianglong Liu**[1]
[1]Beihang University
[2]Shanghai Jiao Tong University
[3]SenseTime Research
{22373167, witt2537, gongruihao, xlliu}@buaa.edu.cn, junyi.chen@sjtu.edu.cn,
baishihao@sensetime.com

Abstract

Structured output generation is increasingly critical for real-world AI systems, particularly in on-device settings where small language models (0.5B–8B parameters) must produce machine-executable outputs under strict latency and privacy constraints. Although constrained decoding provides formal guarantees of structural validity without retraining, its effectiveness across different tasks, models, and constraint formalisms remains insufficiently understood. We introduce **StructureBench**, a comprehensive benchmark for structured generation on edge devices, covering JSON and tool-call generation, code synthesis, mathematics, science, and domain-specific languages, and evaluating over 11 on-device language and vision–language models spanning 0.5B–8B parameters. StructureBench systematically compares prompt-based generation with three widely used constrained decoding frameworks—Outlines, XGrammar, and Guidance—and adopts decoupled metrics to separately assess structural validity and semantic correctness. Our experiments show that constrained decoding consistently enforces syntactic validity, but does not reliably improve semantic accuracy and may even degrade performance for smaller models or complex grammars. These findings reveal clear task- and model-dependent boundaries for effective constrained decoding and highlight the need for scenario-aware adaptation in on-device structured generation. We release StructureBench at https://github.com/Str-Ben/StructureBench.

1 Introduction

Recent advances in large language models (LLMs) have enabled impressive natural language capabilities, yet real-world applications increasingly require structured outputs such as JSON, SQL, or tool-call specifications that [Cai *et al.*, 2023; Li *et al.*, 2024; Zhuo *et al.*, 2024; Chen *et al.*, 2025c] are both syntactically valid and machine-executable. This shift toward structured generation is especially critical in on-device settings. On-device models are typically small (0.5B–8B parameters) and exhibit weaker instruction-following abilities than their cloud-based counterparts, making them prone to format violations without explicit guidance. At the same time, dominant edge use cases—like information extraction, API invocation, and local task automation—naturally demand structured responses. In privacy-sensitive or offline scenarios where cloud fallback is unavailable, correctness must be guaranteed locally on the first attempt, turning structured generation from a desirable feature into a functional necessity.

To fulfill this requirement, the research community has converged on two practical, training-free strategies: prompt engineering and constrained decoding. The latter—enforcing grammatical rules during token-by-token generation without modifying model parameters—has become particularly appealing for on-device use due to its zero-training cost, strong correctness guarantees, and compatibility with quantized models. Frameworks like Outlines [Willard and Louf, 2023], XGrammar [Dong *et al.*, 2024], and Guidance [Guidance AI, 2023] now offer robust support for grammar-guided generation across diverse formats.

Despite increasing interest in structured generation, existing benchmarks exhibit three fundamental limitations. First, they are narrowly scoped to single formats—for example, JSONSchemaBench [Geng *et al.*, 2025] focuses exclusively on JSON validity—largely ignoring other structured outputs such as code, DSLs, or tool-calling protocols. Second, they emphasize decoder efficiency metrics (e.g., latency or memory overhead) rather than task-level effectiveness, offering limited insight into whether constrained decoding improves semantic correctness beyond enforcing syntax. Third, and most critically, evaluations are almost exclusively conducted on large cloud-scale models, overlooking the resource-constrained on-device setting where constrained decoding is often most necessary due to limited instruction-following capacity.

As a result, three key questions remain unresolved: (i) how different constrained decoding frameworks compare in terms of syntactic reliability and semantic quality on edge devices; (ii) how constrained decoding interacts with diverse on-device model characteristics, including scale (0.5B–8B), architecture, and modality; and (iii) whether constrained decoding generalizes robustly across heterogeneous structured

*Corresponding author.

tasks, rather than isolated single-format scenarios.

To address these gaps, we introduce StructureBench, a comprehensive benchmark tailored for structured generation in on-device settings. StructureBench jointly spans three practical axes: diverse structured scenarios, broad on-device model coverage, and systematic comparison of constrained decoding tools. It covers representative tasks including JSON and tool-call generation, code completion, mathematical reasoning, scientific outputs, and DSL synthesis, and evaluates over ten state-of-the-art on-device models across scales, architectures, and modalities. Moreover, StructureBench integrates three widely used constrained decoding frameworks—Outlines, XGrammar, and Guidance—together with prompt-based baselines, enabling fine-grained analysis of both structural validity and task correctness. Our results show that while constrained decoding consistently guarantees syntactic validity, it does not necessarily improve semantic accuracy; smaller models may degenerate under complex grammars, whereas for well-defined tasks such as tool calling or schema-constrained JSON generation, constrained decoding already enables reliable on-device deployment.

The main contributions of this work are:

- StructureBench: A scalable, extensible benchmark that supports plug-and-play integration of new models, tasks, and constrained decoding backends, enabling holistic evaluation of structured generation in resource-constrained environments.
- Decoupled evaluation metrics: We propose novel metrics that separately quantify (a) syntactic compliance, (b) semantic accuracy under constraints, and (c) the marginal gain in semantic correctness attributable to constrained decoding within the space of valid outputs.
- Grammar analysis and refinement: We correlate task difficulty with formal grammar and propose methods for refining grammar constraints by investigating the peculiar phenomena induced by constrained decoding.
- Practical deployment guidelines: Based on extensive empirical results, we map application scenarios to effective model–decoder combinations, offering actionable recommendations for developers targeting on-device structured generation.

2 Related Work

2.1 On-device LLM

Recent work on on-device large language models has focused on compact architectures with fewer than 8B parameters, enabling efficient, real-time, and privacy-preserving inference on edge devices. Representative examples include Qwen3 family [Yang *et al.*, 2025], which offers open-weight models from 0.6B to 8B parameters with strong multilingual capabilities and optimized deployability, as well as lightweight series such as Google's Gemma [Team, 2025a], Huawei's openPangu [Ascend-Tribe, 2025; Chen *et al.*, 2025a] and Microsoft's Phi-3 Mini [Abdin *et al.*, 2024], which demonstrate that careful scaling and optimization can support local conversational, summarization, and translation tasks. Open-source models such as TinyLlama [Zhang *et al.*, 2024] and StableLM-Zephyr [Tow *et al.*, 2024] further expand this ecosystem. In parallel, MiniCPM4 [Team, 2025b] advances on-device efficiency through architectural and training optimizations, providing 0.5B and 8B variants with improved inference speed and long-context capability. These efforts reflect a maturing landscape in which compact LLMs are increasingly practical for diverse edge AI applications despite inherent capacity limitations.

2.2 Structured Generation

Structured output generation has become a central research direction in the LLM era, aiming to produce outputs that are both semantically correct and strictly compliant with predefined formats required by downstream systems, such as schema-constrained JSON, regex-validated strings, or grammar-based SQL queries [Chen *et al.*, 2025b]. A widely adopted solution is *constrained decoding*, which enforces structural correctness by dynamically restricting each decoding step to legal next tokens derived from the target specification and the partial output [Hu *et al.*, 2019; Scholak *et al.*, 2021]. By masking invalid tokens and sampling only from valid candidates, constrained decoding provides formal guarantees of structural validity beyond prompt-based guidance. This paradigm is supported by several open-source frameworks, including Guidance, XGrammar [Dong *et al.*, 2024], and Outlines [Willard and Louf, 2023], which collectively enable reliable format-valid generation across diverse structured tasks.

2.3 Structured Generation Benchmarks

Several benchmarks have been proposed to evaluate LLMs' structured generation capabilities, yet they leave important gaps unaddressed. JSONSchemaBench [Geng *et al.*, 2025] focuses on constrained decoding for JSON generation using large-scale real-world schemas, but is limited to a single output format and primarily emphasizes constraint compliance and efficiency. Other benchmarks examine structured generation without decoding-time constraints, such as BenchCLAMP for CFG-based parsing [Roy *et al.*, 2023], Struc-Bench for tabular and complex structured data [Tang *et al.*, 2024], and Struct-Bench for synthetic structured data under differential privacy [Wang *et al.*, 2025]. However, these benchmarks are typically restricted to specific task types, do not systematically compare constrained decoding against prompt-based generation, and are evaluated almost exclusively on large cloud-scale models. As a result, the effectiveness and limitations of structured generation methods—particularly constrained decoding—remain underexplored in realistic on-device settings across diverse structured tasks.

3 StructureBench Details

3.1 Models

StructureBench evaluates a total of 11 on-device models developed by five institutions, covering a wide range of architectures and parameter scales from 0.5B to 8B. The evaluated models include general-purpose large language models (LLMs), vision–language models (VLMs), as well as

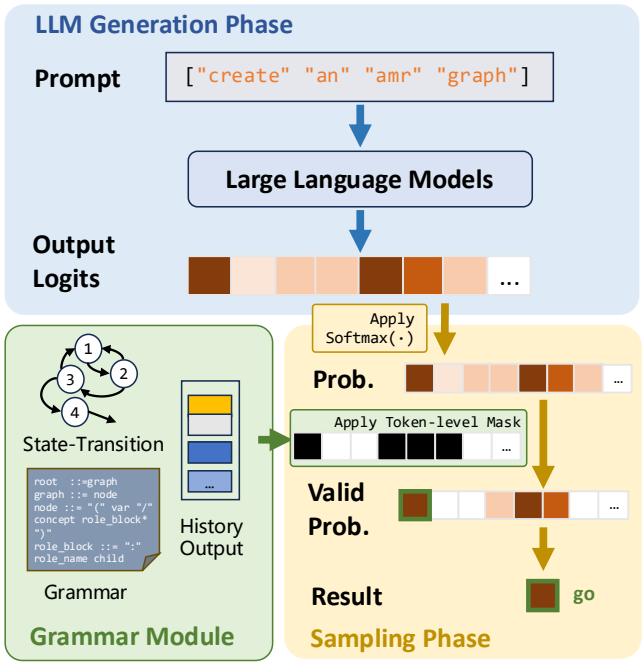

Figure 1: The mechanism of constrained decoding.

Model Family	Modality/Task	Specific Models	Sizes
MiniCPM	Multi-modal	MiniCPM-o-2.6, MiniCPM-V-2.6	2.8B, 8B
	Language	MiniCPM4-0.5B	0.5B
Qwen	Multi-modal	Qwen3-VL-2B-Instruct, Qwen3-VL-8B-Instruct	2B, 8B
	Language	Qwen3-0.6B, Qwen3-8B	0.6B, 8B
	Code	Qwen2.5-Coder-0.5B-Instruct	0.5B
Llama	Language	Llama-3.2-1B-Instruct	1B
Pangu	Language	openPangu-Embedded-1B	1B
DeepSeek	Code	Deepseek-Coder-1.3B-Instruct	1.3B

Table 1: On-device models included in the benchmark.

code-oriented models specifically designed for programmatic tasks. This diverse selection enables a comprehensive assessment of model capabilities under realistic on-device constraints. Detailed model configurations and characteristics are summarized in Table 1.

3.2 Datasets

To comprehensively evaluate on-device models for structured generation, StructureBench spans six representative application domains, ranging from general-purpose data formats to specialized formal languages. The benchmark comprises 13 structured datasets with both text and vision–language inputs, and requires outputs in diverse structured forms, including JSON, Python, SQL, LaTeX, SMILES, PDDL, PENMAN, and mathematical formulas. Structural validity is enforced via JSON Schema, EBNF grammars, or regular expressions. An overview of all datasets is provided in Table 2.

Tool Calling. This domain evaluates function invocation and API interaction via schema-constrained JSON generation, including Berkeley Function Calling Leaderboard (BFCL) [Patil *et al.*, 2025] and hermes-function-calling-v1 (Hermes-FC) [interstellarninja and Teknium, 2025].

Domain	Dataset	Text	Image	Output	Constraint
Tool Call	BFCL	✓	×	JSON	JSON
	Hermes-FC	✓	×	JSON	JSON
Info. Extract	JSONSchemaBench	✓	×	JSON	JSON
	Cheque	✓	✓	JSON	JSON
Math	Ape210k	✓	×	Formula	Regex
	LaTeX-OCR	✓	✓	LaTeX	EBNF
Code	HumanEval	✓	×	Python	EBNF
	Text2SQL	✓	×	SQL	EBNF
Science	SMILES	✓	×	SMILES	EBNF
DSL	AMR-3	✓	×	PENMAN	EBNF
	Planetarium	✓	×	PDDL	EBNF

Table 2: Datasets included in StructureBench.

Information Extraction. Information extraction assesses mapping from unstructured inputs to structured outputs. This domain includes JSONSchemaBench (JSBench) [Geng *et al.*, 2025] and Cheque [Singh, 2023], the latter incorporating image inputs for vision–language evaluation.

Code Generation. Code generation requires strict syntactic and semantic correctness, and includes HumanEval (Python) [Chen *et al.*, 2021] and Text2SQL (SQL) [Meyer *et al.*, 2024], both evaluated under formal grammar constraints.

Mathematics. Mathematical reasoning under symbolic constraints is evaluated using Calc-ape210k (Ape210k) [Kadlčík *et al.*, 2023], and LaTeX-OCR [linxy, 2018], with LaTeX-OCR additionally targeting vision–language models.

Science. This domain focuses on specialized symbolic generation, with smiles-eval (SMILES) [Weininger, 1988] evaluating grammatical validity of chemical representations.

Domain-Specific Languages. DSL tasks test generation in rarely seen formal languages, including AMR-3 (PENMAN) [Banarescu *et al.*, 2013] and Planetarium (PDDL) [Zuo *et al.*, 2024], and are particularly suited for studying the effectiveness of constrained decoding in unfamiliar formal systems.

3.3 Structured Generation Methods under Evaluation

StructureBench compares two representative classes of structured generation approaches: prompt-based generation, which relies solely on model capabilities, and constrained decoding, which enforces structural correctness during generation. This design enables a systematic assessment of how explicit constraints interact with on-device model capacity across diverse structured tasks.

Prompt-based Generation. Prompt-based structured generation serves as the baseline approach in our benchmark. For each task, we construct prompt templates that specify the application scenario, input modality, and required output format, optionally including formatting instructions or few-shot examples. No structural constraints are applied during generation, and the model is free to sample from its full vocabulary. As a result, the generated output depends entirely on the model's instruction-following ability, making this approach a direct reflection of the inherent structured generation capability of on-device models.

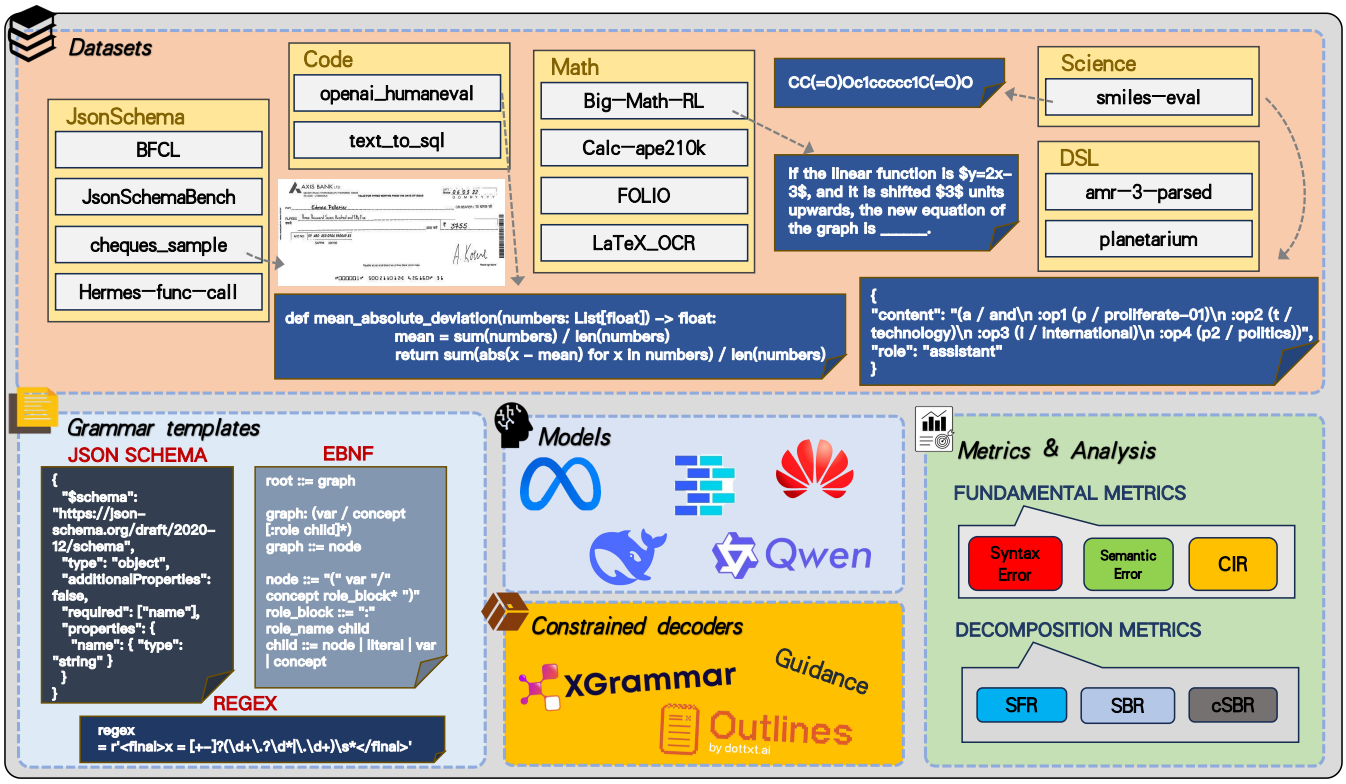

Figure 2: The overview of StructureBench.

Constrained Decoding. To evaluate structure-constrained generation, StructureBench integrates three representative open-source constrained decoding frameworks: Outlines, XGrammar, and Guidance. These frameworks differ in interface design, supported constraint formalisms (e.g., JSON Schema, regular expressions, and EBNF/CFG grammars), and execution workflows. Because on-device deployment is sensitive to latency and memory, we further report an efficiency check for constrained decoding in the supplementary material, showing negligible per-token latency change and substantially reduced token-mask cache memory.

3.4 Benchmark Methodology

Two-Dimensional Evaluation Formulation. We evaluate each output along two orthogonal dimensions: *Structural Validity*, whether it satisfies the executable grammar/schema, and *Task Correctness*, whether a structurally valid output fulfills the task objective. For each sample with input x and domain d, we select the corresponding output type (*e.g.*, JSON, SQL, Python), instantiate a matched constraint formalism (JSON Schema, regex, or EBNF/CFG), validate grammar-based constraints with Lark [Erez, 2016], and apply the same domain-specific prompt template. Task-specific validation procedures are summarized in Table 3.

Controlled Comparison Protocol. We standardize model configurations, decoding hyperparameters, maximum generation length, task instructions, and output specifications across prompt-based and constrained-decoding runs. All constrained decoders use the same constraint inputs and generation protocol; fallback, retry, and error-recovery mechanisms are disabled, and any constraint compilation or decoding failure is recorded as a generation failure.

Output Type	Task Correctness Checker
JSON	jsonschema
Python	Python interpreter
PDDL	Tarski
SQL	SQLGlot
AMR	Penman
LaTeX	Sympy
SMILES	RDKit
FOL	Prover9 automated prover
Formula	Sympy

Table 3: Python libraries as parsers for each output type.

3.5 Evaluation Metrics

We consider an evaluation set $\mathcal{D} = \{1, \ldots, N\}$ and compare two decoding methods: **prompt-only** p and **constrained decoding** c. For each sample $i \in \mathcal{D}$ and method $m \in \{p, c\}$, we define binary indicators to characterize failure modes in structured generation.

Fundamental Evaluation Metrics

Syntax and Semantic Errors. We distinguish two types of errors. A **syntax error** $\mathrm{SynErr}_m(i) \in \{0, 1\}$ occurs when the output violates the target grammar or schema and cannot be parsed or validated, corresponding to structural invalidity. A **semantic error** $\mathrm{SemErr}_m(i) \in \{0, 1\}$ occurs when the output is structurally valid but task-incorrect (e.g., functionally or logically wrong). By definition, semantic errors are only defined over structurally valid outputs.

Constraint-Induced Repetition (CIR). We further identify *Constraint-Induced Repetition*, a degenerate failure mode in which constrained decoding suppresses high-probability tokens and forces repetitive selection among low-probability but valid tokens. CIR preserves syntactic validity but yields meaningless outputs and is recorded separately from syntax errors.

Decomposition Metrics for Constraint Effect

Structural Fix Rate (SFR). To quantify structural correction, we define

$$\mathrm{SFR} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}!\left(\mathrm{SynErr}_p(i) \ \wedge \ \neg\mathrm{SynErr}_c(i)\right). \quad (1)$$

measuring the fraction of samples where constrained decoding repairs syntax errors from prompt-only generation.

Semantic Bonus Rate (SBR). Overall semantic improvement is measured by

$$\mathrm{SBR} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}!\left(\mathrm{SemErr}_p(i) \ \wedge \ \neg\mathrm{SemErr}_c(i)\right). \quad (2)$$

capturing the total reduction in semantic errors.

Conditioned Semantic Bonus Rate (cSBR). While SBR reflects global semantic gains, it does not isolate cases where semantic improvement is directly attributable to successful structural correction. To isolate semantic gains attributable to structural correction, we define

$$\mathrm{cSBR} = \frac{\sum_{i=1}^{N} \mathbf{1}\begin{pmatrix}\mathrm{SynErr}_p(i) \ \wedge \ \neg\mathrm{SynErr}_c(i) \wedge \\ \mathrm{SemErr}_p(i) \ \wedge \ \neg\mathrm{SemErr}_c(i)\end{pmatrix}}{\sum_{i=1}^{N} \mathbf{1}\left(\mathrm{SynErr}_p(i) \ \wedge \ \neg\mathrm{SynErr}_c(i)\right)}. \quad (3)$$

SR–SE taxonomy. Finally, we define **Structural Reliability** (SR) as the fraction of structurally valid outputs and **Semantic Efficiency** (SE) as conditional correctness given validity:

$$\begin{aligned}\mathrm{SR} &= 1 - \mathrm{SynErr},\\ \mathrm{SE} &= \frac{1 - \mathrm{SemErr} - \mathrm{SynErr}}{\mathrm{SR}}.\end{aligned} \quad (4)$$

SR captures whether valid structures can be produced, while SE measures semantic quality within the valid output space.

4 Experiment Result and Analysis

Across tasks, larger models generally improve *structural validity* more consistently than *semantic correctness*, while constrained decoding provides strong gains mainly when failures are dominated by format violations. Importantly, constrained decoding is not universally beneficial: on some open-ended, high-entropy outputs it can induce degeneration (e.g., repetition), motivating a deeper, pattern-based analysis. Accordingly, the remainder of this section decomposes performance from three angles: (i) the impact of **task patterns** (§4.1), (ii) the sensitivity to **constraint/grammar complexity** (§4.2), and (iii) **failure modes** and their practical implications (§4.3). Complete experimental results and supporting materials are available at https://github.com/Str-Ben/StructureBench/blob/main/appendix.md.

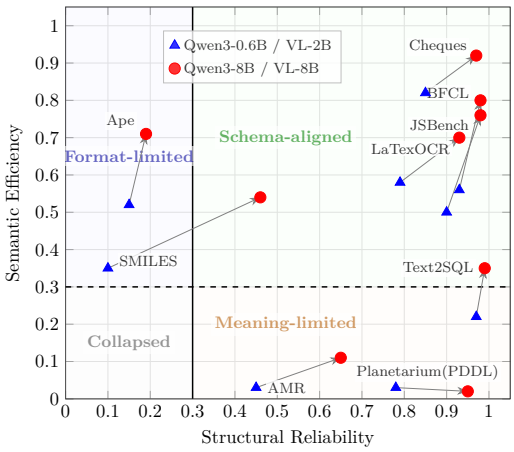

Figure 3: **SR–SE task taxonomy (prompt-only).** Arrows indicate the shift under scaling.

4.1 Impact of Task Patterns

We first summarize tasks by a lightweight SR–SE taxonomy under the prompt-only baseline, and then connect each regime to constrained-decoding *signatures* using the decomposition metrics in §3.5. Figure 3 visualizes this taxonomy for four on-device scales (Qwen3 models family) on representative StructureBench tasks, together with four regimes for readability.

Constrained-decoding signatures. We connect the above regimes to decomposed indicators in §3.5. In practice, we summarize the *repair* effect using a conditional structural fix rate $\mathrm{SFR}_{\mathrm{cond}} = P(\mathrm{Valid}_c \mid \mathrm{Invalid}_p)$, which is easier to compare across tasks with different baseline SR. We then use cSBR to measure whether structural repairs translate into *semantic* gains, and CIR to quantify constraint-induced degeneration. Table 4 reports representative signatures.

Key findings. We summarize the key findings as follows:

- **Schema-aligned tasks benefit most from constrained decoding.** For closed, interface-aligned outputs (e.g., tool calling and schema-based JSON), constraints reliably fix structural errors and often translate them into semantic gains (high cSBR, negligible CIR). As shown in Figure 3, these tasks already exhibit strong SR and moderate-to-high SE under prompt-only generation, making constrained decoding an effective reliability amplifier for on-device deployment.
- **Meaning-bound tasks remain semantically limited despite higher validity.** For symbolic and compositional tasks such as PDDL, SQL, and AMR, constrained decoding improves parseability but does not address missing global reasoning. As a result, semantic gains remain minimal ($\mathrm{cSBR} \approx 0$), even when structural validity increases.
- **Scaling improves structural reliability, but semantic gains are pattern-dependent.** Model scaling consistently increases SR across tasks, while improvements in SE depend on task type: schema-aligned tasks benefit from scaling, whereas meaning-bound tasks show little semantic improvement. This explains why constrained decoding

Pattern	cSFR ↑	cSBR ↑	CIR ↓	Impact of task pattern
Schema-aligned (JSON, Latex)	94.8%	70.5%	0.05%	Structural repairs often convert to semantic gains.
Meaning-bound (AMR, PDDL, SQL)	16.0%	1.5%	7.2%	Validity improves: semantics remains the bottleneck.
Format-limited (SMILES, Formula)	34.0%	33.3%	1.2%	Validity improves; semantic gains require domain knowledge.
Exception (Python)	0.6%	0.0%	97.6%	Hard constraints can induce degeneration and wipe out gains.

Table 4: **Constrained-decoding signatures across task patterns with Qwen3-0.6B + XGrammar.** $\mathrm{cSFR} = P(\mathrm{Valid}_c \mid \mathrm{Invalid}_p)$ (conditional repair success). cSBR and CIR follow §3.5.

Model	Loose Constraints						Moderate Constraints					
	Outlines			XGrammar			Outlines			XGrammar		
	ACC	SV	CIR	ACC	SV	CIR	ACC	SV	CIR	ACC	SV	CIR
Llama-3.2 1B	0.2	64.8	0.0	0.0	0.2	7.2	0.2	78.0	0.0	0.0	55.4	1.0
MiniCPM4 0.5B	0.4	36.4	58.4	0.4	56.0	24.7	0.4	42.2	46.2	0.0	25.8	73.4
Qwen3 0.6B	0.8	81.4	7.0	0.2	14.8	21.2	0.6	72.2	11.8	0.0	32.4	1.2
Qwen3 8B	0.4	68.2	6.4	0.2	99.2	8.0	0.4	92.0	1.0	0.0	91.0	0.0

Table 5: Performance on the Planetarium task under different constraint strengths. *ACC*: accuracy; *SV*: structural validity.

is effective in some scenarios but saturates—or even degrades—performance in others.

- **Open-ended code generation is a notable exception.** For tasks such as Python generation, strict grammars can overconstrain decoding and induce repetition (high CIR), eliminating conditioned semantic gains ($\mathrm{cSBR} \approx 0$). This suggests that hard constraints should be paired with stronger models or replaced by softer or partial constraints.

Takeaway #1

Schema-aligned tasks benefit most: constraints fix structure and often yield real semantic gains (high cSBR, low CIR). Meaning-limited tasks only become *parseable*, remain semantic mismatch. Python generation may *degrade* under hard constraints, requiring softer constraints or extra supervision.

4.2 Sensitivity to Grammar Complexity

Results in Table 4 show that the effectiveness of constrained decoding varies widely across scenarios. For JSON-schema tasks, it consistently improves both structural validity and task accuracy, while on SMILES-Eval it yields only moderate structural gains with limited semantic improvement. In most other scenarios, constrained decoding introduces negative side effects. Grammar complexity also reflects task complexity within a scenario, motivating an analysis of how grammar design drives these divergent outcomes.

A natural hypothesis is that grammatical complexity correlates with grammar size, measured by the number of production rules. However, this is not supported empirically. The penman grammar for AMR contains fewer than 15 productions yet substantially degrades structural validity, whereas the SMILES grammar has around 40 productions and improves it. This indicates that grammar size alone does not determine constrained-decoding behavior.

Instead, structural properties of the grammar are decisive. Grammars that improve structural validity, such as those for JSON Schema and SMILES, are largely tree-structured and avoid recursive rules, preventing repetitive generation. In contrast, in complex or data-scarce scenarios, grammars with recursion frequently induce looping behavior during constrained decoding, resulting in repeated tokens and structural degeneration. These findings suggest that grammar structure—rather than grammar size—is the primary factor governing the success or failure of constrained decoding.

4.3 Failure Mode: Constraint-Induced Token Repetition

LLMs exhibit high Content CIR on several tasks in Table 4, indicating a pronounced tendency toward repetitive outputs under constrained decoding. We analyze this issue by characterizing the repetition phenomenon, explaining its underlying cause, and examining two key influencing factors: grammar complexity and model capacity.

Repetition Phenomenon. Repetition under constrained decoding appears in two forms: *sentence-level repetition*, where similar structural fragments repeat until the token limit is reached, and *character-level repetition*, where the same token is emitted consecutively. Since sentence-level repetition also occurs under unconstrained decoding, we focus on character-level repetition, which is specific to constrained decoding and directly captured by Content CIR.

Underlying Cause: Constraint–Uncertainty Interaction. Character-level repetition arises from the interaction between limited model capability and hard structural constraints. When the model cannot infer the required structure, its high-probability tokens are masked, leaving only low-logit but syntactically valid options. If the grammar contains recursion or regex-like patterns, decoding may repeatedly select the same token, producing character-level loops (Figure 4). Tightening constraints can reduce repetition, but overly strict grammars may mask all tokens and stall decoding (§4.2). Thus, constrained decoding amplifies model uncertainty into a visible degeneration mode rather than creating repetition by itself. To validate this mechanism, we inspect a Planetarium (PDDL) sample generated by Qwen3-0.6B with constrained decoding. After the model emits `(:` with high confidence (0.878), the probabilities of legal continuations drop sharply (`me`: 2.63×10^{-5}, `tri`: 1.97×10^{-5}, `c`: 1.31×10^{-5}), after which repeated whitespace/tab tokens are selected with low probability (≈ 0.003), producing the observed character-level loop.

Impact of Grammar Complexity. To assess the role of grammar complexity, we compare constraint templates with different tightness on complex tasks. Because fully specified grammars are often impractical, relaxed templates (e.g., regex-based identifiers) are commonly used, but we find they induce substantial repetition. Strengthening the grammar by reducing regex usage significantly lowers CIR and improves

Task (Output)	Model + Decoder	SFR ↑	cSBR ↑	CIR ↓
HumanEval (Python)	MiniCPM4-0.5B + XGrammar	0.3871	0.0	0.9833
HumanEval (Python)	Qwen3-0.6B + XGrammar	0.76	0.0526	0.0526
AMR (PENMAN)	Qwen3-8B + Outlines	0.6154	0.0536	0.372
AMR (PENMAN)	Qwen3-8B + XGrammar	0.5385	0.0	0.002

Table 6: Representative constraint-induced degeneration. High SFR alone is insufficient; strong constraints can still yield repetitive outputs (high CIR) with diminishing semantic gains (low cSBR) despite valid syntax.

structural validity, while task accuracy remains largely unchanged (Table 5). Only models that already perform well under loose constraints show minor degradation. Overall, moving from loose to moderate constraints reduces repetition and improves structural reliability, highlighting grammar complexity as a key factor governing CIR.

Impact of Model Capacity. Repetition under constrained decoding ultimately stems from limited model capacity, which constraints exacerbate. As shown in Table 4, repetition is concentrated in code and DSL tasks, where CIR can be extremely high. Within the same model family, scale partially mitigates repetition: for instance, Qwen3-8B exhibits much lower CIR than Qwen3-0.6B, suggesting stronger reasoning ability can compensate for missing task-specific training. However, scale alone is insufficient. As shown in Table 6, even large models can degenerate under strong constraints, achieving high SFR while collapsing into repetitive outputs with near-zero cSBR. Additional scaling results in the supplementary material show the same trend beyond the on-device range: moving from 8B to 32B/72B substantially increases SBR while reducing CIR under the same setting.

Consistency across Model Types. This behavior is not limited to text-only LLMs. Vision–Language Models show repetition patterns consistent with the same task taxonomy in Table 4, indicating that constraint-induced repetition is driven by task structure and model capacity rather than modality.

Takeaway #2

CIR and repetition stem from limited model capacity and are amplified by loose constraints; hence, constrained decoding should employ the strictest grammar the model can support, using repetition analysis to refine those constraints.

4.4 Discussion: Best Practices for Constrained Decoding

Our findings suggest that constrained decoding should be treated as a **reliability layer** rather than a universal enhancement. Its utility depends on the interplay between structural necessity and the risk of model degeneration.

Decision Framework. Deployment should be calibrated based on the **bottleneck type** (structure vs. meaning) and **grammar risk** (complexity). As shown in Table 7, *Hard Constraints* suit structure-limited, low-risk tasks (*e.g.*, acyclic JSON). Conversely, meaning-limited tasks or recursive grammars require *Soft Validation* or partial grammars to prevent Constraint-Induced Regression (CIR).

Locating the "Sweet Spot". To optimize complex constraints, we propose an incremental **constraint-ladder** procedure: (i) enforce top-level skeletons for feasibility; (ii) impose depth caps on recursive elements; and (iii) select the tightest grammar that maximizes structural validity before semantic gains (cSBR) plateau.

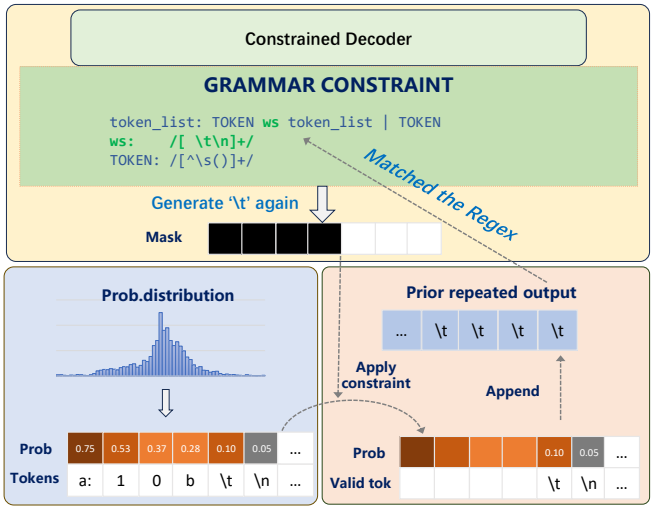

Figure 4: The Mechanism for Character-level Repetition

Grammar Risk	Structure-Limited (*Low SR*)	Meaning-Limited (*Low SE*)
Low *Acyclic low CIR*	**Hard Constraints** Maximize semantic validity; high cSBR expected (e.g., JSON).	**Guardrails Only** Ensure parseability; rely on RAG/SFT for correctness.
High *Recursive/cyclic high CIR*	**Sweet-Spot** Use partial grammars; trade strictness for feasibility (§4.4).	**Avoid Hard Constraints** Prefer post-hoc repair; upgrade base capability first.

Table 7: Decision–deployment matrix.

Takeaway #3

Best Practices for Constrained Decoding: Pre-analyze tasks via Figure 3 to determine appropriateness; during inference, monitor for low-confidence sequences over 10 steps as triggers to tighten constraints; post-hoc, evaluate via SFR, cSBR, and CIR.

5 Conclusion

On-device constrained decoding always keeps the output structurally correct, but whether it helps meaning depends on the task and the model: it excels in schema-bound JSON and tool-call scenarios but can degrade reasoning-heavy or grammar-rich tasks, highlighting the need for scenario-aware adaptation; we further distill practical insights into how to select and tune constrained decoding strategies for resource-constrained devices.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Nos. 62525601, 62476018), and the Postdoctoral Fellowship Program of CPSF (No. BX20250487).

Contribution Statement

Xiaokun Xiong and Zhengjie Xu contributed equally to this work.

References

[Abdin *et al.*, 2024] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, et al. Phi-3 technical report: A highly capable language model locally on your phone, 2024.

[Ascend-Tribe, 2025] Ascend-Tribe. Openpangu-embedded-1b model. https://ai.gitcode.com/ascend-tribe/openpangu-embedded-1b-model, 2025. GitCode repository, accessed 2026-01-20.

[Banarescu *et al.*, 2013] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, et al. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[Cai *et al.*, 2023] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*, 2023.

[Chen *et al.*, 2021] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, et al. Evaluating large language models trained on code, 2021.

[Chen *et al.*, 2025a] Hanting Chen, Yasheng Wang, Kai Han, Dong Li, Lin Li, Zhenni Bi, Jinpeng Li, Haoyu Wang, Fei Mi, Mingjian Zhu, Bin Wang, Kaikai Song, Yifei Fu, Xu He, Yu Luo, Chong Zhu, Quan He, Xueyu Wu, Wei He, Hailin Hu, Yehui Tang, Dacheng Tao, Xinghao Chen, Yunhe Wang, et al. Pangu embedded: An efficient dual-system llm reasoner with metacognition, 2025.

[Chen *et al.*, 2025b] Junyi Chen, Shihao Bai, Zaijun Wang, Siyu Wu, Chuheng Du, Hailong Yang, Ruihao Gong, Shengzhong Liu, Fan Wu, and Guihai Chen. Pre^3: Enabling deterministic pushdown automata for faster structured LLM generation. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11253–11267, Vienna, Austria, July 2025. Association for Computational Linguistics.

[Chen *et al.*, 2025c] Junyi Chen, Chuheng Du, Renyuan Liu, Shuochao Yao, Dingtian Yan, Jiang Liao, Shengzhong Liu, Fan Wu, and Guihai Chen. Tokenflow: Responsive llm text streaming serving under request burst via preemptive scheduling. *arXiv preprint arXiv:2510.02758*, 2025.

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

[Erez, 2016] Shinan Erez. Lark: A parsing toolkit for python. https://github.com/lark-parser/lark, 2016. GitHub repository, accessed 2026-01-20.

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

[Guidance AI, 2023] Guidance AI. Guidance: A language model programming framework. https://github.com/guidance-ai/guidance, 2023. Accessed: 2024-12-18.

[Hu *et al.*, 2019] J. Edward Hu, Huda Khayrallah, Ryan Culkin, Patrick Xia, Tongfei Chen, Matt Post, and Benjamin Van Durme. Improved lexically constrained decoding for translation and monolingual rewriting. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 839–850, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[interstellarninja and Teknium, 2025] interstellarninja and Teknium. Hermes function calling dataset v1. https://huggingface.co/NousResearch/hermes-function-calling-v1, 2025.

[Kadlčík *et al.*, 2023] Marek Kadlčík, Michal Štefánik, Ondřej Sotolář, and Vlastimil Martinek. Calc-x and calcformers: Empowering arithmetical chain-of-thought through interaction with symbolic systems. In *Proceedings of the The 2023 Conference on Empirical Methods in Natural Language Processing: Main track*, Singapore, Singapore, December 2023. Association for Computational Linguistics.

[Li *et al.*, 2024] Zekun Li, Zhiyu Zoey Chen, Mike Ross, Patrick Huber, Seungwhan Moon, Zhaojiang Lin, Xin Luna Dong, Adithya Sagar, Xifeng Yan, and Paul A Crook. Large language models as zero-shot dialogue state tracker through function calling. *arXiv preprint arXiv:2402.10466*, 2024.

[linxy, 2018] linxy. Latex_ocr (hugging face dataset). https://huggingface.co/datasets/linxy/LaTeX_OCR, 2018. Hugging Face Datasets, accessed 2026-01-20.

[Meyer *et al.*, 2024] Yev Meyer, Marjan Emadi, Dhruv Nathawani, Lipika Ramaswamy, Kendrick Boyd, Maarten Van Segbroeck, Matthew Grossman, Piotr Mlocek, and Drew Newberry. Synthetic-Text-To-SQL: A synthetic dataset for training language models to generate sql queries from natural language prompts, April 2024.

[Patil *et al.*, 2025] Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and

Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

[Roy *et al.*, 2023] Subhro Roy, Samuel Thomson, Tongfei Chen, Richard Shin, Adam Pauls, Jason Eisner, and Benjamin Van Durme. Benchclamp: A benchmark for evaluating language models on syntactic and semantic parsing. *Advances in Neural Information Processing Systems*, 36:49814–49829, 2023.

[Scholak *et al.*, 2021] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*, 2021.

[Singh, 2023] Shivalika Singh. Cheques sample data (hugging face dataset). https://huggingface.co/datasets/shivalikasingh/cheques_sample_data, 2023.

[Tang *et al.*, 2024] Xiangru Tang, Yiming Zong, Jason Phang, Yilun Zhao, Wangchunshu Zhou, Arman Cohan, and Mark Gerstein. Struc-bench: Are large language models good at generating complex structured tabular data? In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 12–34, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

[Team, 2025a] Gemma Team. Gemma 3 technical report, 2025.

[Team, 2025b] MiniCPM Team. Minicpm4: Ultra-efficient llms on end devices, 2025.

[Tow *et al.*, 2024] Jonathan Tow, Marco Bellagente, Dakota Mahan, and Carlos Riquelme. Stablelm 3b 4e1t, 2024.

[Wang *et al.*, 2025] Shuaiqi Wang, Vikas Raunak, Arturs Backurs, Victor Reis, Pei Zhou, Sihao Chen, Longqi Yang, Zinan Lin, Sergey Yekhanin, and Giulia Fanti. Structbench: A benchmark for differentially private structured text generation, 2025.

[Weininger, 1988] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988.

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

[Yang *et al.*, 2025] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, et al. Qwen3 technical report, 2025.

[Zhang *et al.*, 2024] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model, 2024.

[Zhuo *et al.*, 2024] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

[Zuo *et al.*, 2024] Max Zuo, Francisco Piedrahita Velez, Xiaochen Li, Michael L. Littman, and Stephen H. Bach. Planetarium: A rigorous benchmark for translating text to structured planning languages, 2024.