# Good-Enough Structured Generation

## A Case Study on JSON Schema

**Ivan Lee** • Loris D'Antoni • Taylor Berg-Kirkpatrick

## 1 Key Finding

A simple **verify-retry loop** achieves **95%+ coverage** on real-world JSON schemas, while grammar-constrained decoding (GCD) achieves only **~50%**.

## 2 What is Grammar-Constrained Decoding?

LLMs are probabilistic—they can produce any token at each step. When you need output conforming to a JSON Schema, there's no guarantee the model follows it.

**GCD solves this** by masking invalid tokens during generation. At each step, only tokens that keep the output valid against the schema are allowed.

**The promise:** Guaranteed schema-compliant output, every time.

*JSON Schema is GCD's primary application—the format powering APIs, configs, and LLM tool-calling. We evaluate on JSONSchemaBench: 9,558 real-world schemas.*

## 3 The Problem

**The reality:** Popular GCD implementations fail on 50% of real-world schemas.

The guarantee sounds great—but it only works if GCD can actually handle your schema.

## 4 Two Paradigms

**Same guarantee, different enforcement point.**

### Constrained Decoding

Guarantee *in the decoder*

*Model cannot produce invalid output*

+ Single-shot, predictable latency
+ Guaranteed valid (when supported)
− Limited by CFG expressiveness
− ~50% schema coverage

### Verify-Retry Loop

Guarantee *from the verifier*

*You don't accept invalid output*

+ Full spec coverage (95%+)
+ Scales with model capability
− Higher latency
− May timeout without valid output

## 5 Key Terms

**Coverage** *(metric)*
Percentage of schemas for which a language model + decoding strategy produces output that validates against the schema

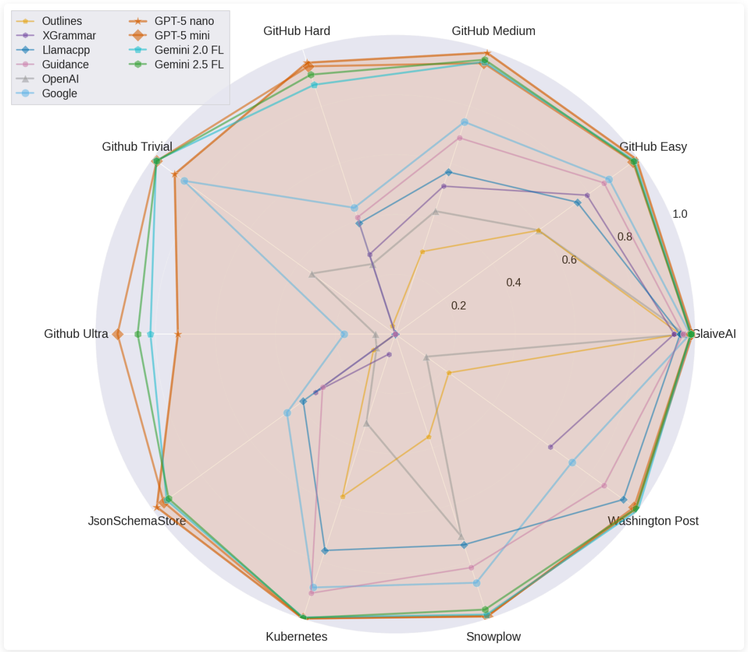
**CFG** *(formalism)*
Context-Free Grammar; the class of formal languages GCD can enforce. JSON Schema exceeds CFG expressiveness.

**GCD** *(decoding strategy)*
Grammar-Constrained Decoding—masks invalid tokens at each generation step to guarantee syntactic validity

**Rejection Sampling** *(decoding strategy)*
Generate, validate, discard and retry from scratch on failure

**Verifier Loop** *(decoding strategy)*
Generate, validate, retry with validation errors as feedback

## 6 Coverage Comparison




*Coverage across JSONSchemaBench datasets. Verify-retry (orange/green) achieves near-perfect coverage. GCD methods (gray/blue) show brittle, dataset-specific failures.*

## 7 Cost vs Coverage

| Model | Strategy | Coverage | Latency |
|---|---|---|---|
| GPT-5 nano | Structured Output | 50.2% | 1.7s |
| | Rejection Sampling | 86→94% | 4.1s |
| | Verifier Loop | 86→94% | 3.6s |
| Gemini 2.5 Flash | Structured Output | 77.9% | 1.9s |
| | Rejection Sampling | 91→97% | 3.2s |
| | Verifier Loop | 91→98% | 3.1s |

Coverage shows first→final attempt with up to 5 retries.

## 8 Why Verifier Loop Over Rejection Sampling?

| Dataset | Structured | Rejection | Verifier | Δ |
|---|---|---|---|---|
| Github_trivial | 60.9% | 98.2% | 99.1% | +0.9 |
| Github_easy | 73.9% | 98.9% | 99.1% | +0.3 |
| Github_medium | 59.2% | 97.8% | 98.3% | +0.6 |
| Github_hard | 33.4% | 91.7% | 95.5% | +3.9 |
| Github_ultra | 9.5% | 86.6% | 92.1% | +5.5 |
| Glaiveai2K | 97.0% | 98.8% | 98.8% | -0.0 |
| JsonSchemaStore | 25.5% | 97.0% | 96.9% | -0.0 |
| Kubernetes | 59.9% | 99.9% | 99.9% | -0.0 |
| Snowplow | 76.9% | 97.9% | 99.5% | +1.6 |
| WashingtonPost | 43.2% | 99.6% | 100.0% | +0.4 |

GPT-5 nano with reasoning tokens.

| Dataset | Structured | Rejection | Verifier | Δ |
|---|---|---|---|---|
| Github_trivial | 89.0% | 98.9% | 98.9% | +0.0 |
| Github_easy | 89.0% | 98.3% | 98.8% | +0.5 |
| Github_medium | 75.7% | 96.7% | 97.4% | +0.7 |
| Github_hard | 44.5% | 87.9% | 94.2% | +6.2 |
| Github_ultra | 16.5% | 87.2% | 93.3% | +6.1 |
| Glaiveai2K | 97.1% | 98.5% | 98.5% | 0.0 |
| JsonSchemaStore | 43.3% | 95.5% | 95.3% | -0.2 |
| Kubernetes | 88.5% | 99.9% | 99.8% | -0.1 |
| Snowplow | 85.1% | 96.8% | 99.3% | +2.5 |
| WashingtonPost | 73.6% | 100.0% | 100.0% | +0.0 |

Gemini 2.5 Flash with thinking. Δ = verifier − rejection.

Feedback helps most on **hard schemas**.

## 9 Why Does GCD Fail?

### 1. Theoretical Ceiling

GCD is limited to **context-free grammars**, but JSON Schema requires:

- **Cross-field dependencies:** `if/then/else`, `dependentSchemas`
- **Value constraints:** `minimum`, `maximum`, `multipleOf`, `minLength`

### 2. Implementation Challenges

- **Token-boundary misalignment:** LLM vocabulary doesn't map to grammar terminals
- **State explosion:** Complex regex patterns create unmanageable automata
- **Incomplete support:** Some libraries have incomplete support for CFG-expressible features like `anyOf` and recursion

## 10 The Bigger Picture

JSON Schema is our case study, but the pattern may generalize.

Structured domains already have mature verifiers—compilers, type checkers, schema validators—that handle full specifications.

Rather than expanding constrained decoding to capture more expressive languages, an alternative: leverage existing verification tools as feedback in an agentic loop.

The research opportunity shifts from *decoder complexity* to *feedback design*: how should verifiers expose information to maximize model correction?

## 11 Scope & Limitations

Our case study focuses on **JSON Schema**, a high-resource format well-represented in training data. GCD may remain valuable for:

- Novel/out-of-distribution DSLs
- Latency-critical applications
- Strict single-shot requirements

**Unweighted benchmark:** JSONSchemaBench treats all schemas equally. Real-world usage may concentrate on schemas where GCD performs better (or worse).

### 🔥 New Preprint 🔥

**Has DeepSeek-OCR solved the long-context problem?**

***Optical Context Compression Is Just (Bad) Autoencoding***

arxiv.org/abs/2512.03643

BERG LAB

UC San Diego

DL4C NeurIPS 2025

NEURAL INFORMATION PROCESSING SYSTEMS

Paper

Contact