IterGen: Iterative Semantic-aware Structured LLM Generation with Backtracking

Shubham Ugare, Rohan Gumaste, Tarun Suresh, Gagandeep Singh, Sasa Misailovic
Department of Computer Science, University of Illinois Urbana-Champaign, USA

Introduction

Grammar-constrained LLM Generation

- Context-free grammars (CFGs) define syntax through terminal symbols, fundamental building blocks of the language (e.g., words) and non-terminal symbols (placeholders e.g. sentence)
- Recent techniques [1] use CFGs to guide LLM generation and ensure outputs adhere to specific syntax rules
- Constrained generation typically uses parsers to analyze partial outputs and filter out invalid tokens

English text EBNF grammar

```
paragraph: sentence+
sentence: word+ sentence_end
word: /[a-zA-Z0-9]+/ | other_punctuations
sentence_end: "." | "!" | "?"
other_punctuations: "," | ";" | ":" | "'" | "\""
% ignore " "
```

IterGen: Main Contributions

- First framework to use grammar symbols as navigational abstractions for both forward and backward LLM generation control
- Enables semantic-aware generation through grammar-guided navigation, allowing targeted corrections
- Significantly improves output quality across multiple domains: SQL, Privacy Leakage, Vega-Lite

Key Interface Functions

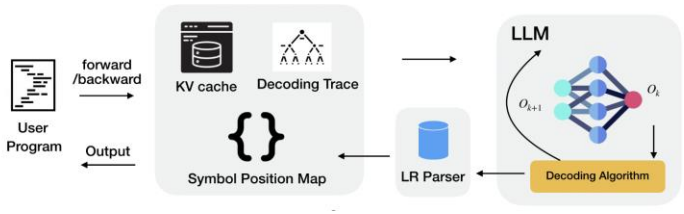
- **forward(symbol, count)**: Generates until specified number of grammar symbols is reached
- **backward(symbol, count)**: Backtracks generation by removing specified symbols
- **view(symbol)**: Inspects all parts of the generation corresponding to a given grammar symbol

Enforcing Semantic Properties over LLMs Structured Output

⬇

Grammar-guided Navigation

Workflow



A user program utilizing IterGen manages LLM generation through **forward** and **backward** calls.

- **Symbol Position Map:** Tracks exact positions of grammar symbols in generated text, enabling precise navigation.
- **Reduce Operation Tracking:** Updates symbol positions during parser reductions.
- **Token Misalignment Solution:** Generates an extra lookahead token to confirm symbol completion, then removes it.
- **Tree-Based Decoding History:** Maintains generation paths as a token tree with a recurrence penalty (γ) to encourage exploration.

Results

IterGen code for SQL generation

```
def generate_sql_with_itergen(iter_gen, problem):
    iter_gen.start(problem['prompt'])
    schema = parse_sql_schema(problem)
    attempts = 0

    while not iter_gen.finished() and attempts < max_iter:
        out = iter_gen.forward(stop_symbols=['column_name', 'table_name'])
        attempts += 1

        if not exists_column(schema, iter_gen.view('column_name')[-1]):
            iter_gen.backward('column_name')
            continue

        if not exists_table(schema, iter_gen.view('table_name')[-1]):
            iter_gen.backward('table_name')
            continue

    return out
```

SQL:

- 41.63% average accuracy vs. 28.9% for standard
- Execution rate of 75.84% vs. 50.28% for standard generation

Model	STD Leaks	Our leaks	STD Perplexity	IterGen Perplexity
Qwen2.5-0.5B-Instruct	46	0	6.87	7.0
Qwen2.5-1.5B-Instruct	57	0	6.17	6.28
Llama-3.2-1B	62	0	6.14	6.25
Llama-3.2-3B	61	0	5.91	6.0
Llama-2-7b	59	0	5.97	6.07
Llama-3-8B	67	0	5.66	5.76

Model	Method	Overall Accuracy (%)	Execute (%)	Tokens
Qwen2.5-0.5B-Instruct	Standard	1.0	2.3	53.27
	SynCode	9.2	28.3	66.79
	IterGen	**26.0**	**64.7**	**39.02**
Qwen2.5-1.5B-Instruct	Standard	0.0	0.0	44.51
	SynCode	32.7	60.7	54.50
	IterGen	**50.7**	**80.0**	**38.44**
Llama-3.2-1B	Standard	25.5	50.6	37.28
	SynCode	28.7	58.7	40.33
	IterGen	**30.9**	**67.6**	**38.66**
Llama-3.2-3B	Standard	28.5	65.3	40.42
	SynCode	34.8	78.8	39.96
	IterGen	**35.6**	**81.4**	**39.08**
Llama-2-7b	Standard	20.3	31.9	42.58
	SynCode	24.4	40.8	46.16
	IterGen	**35.1**	**64.5**	**51.36**
Llama-3-8B	Standard	46.2	87.7	32.95
	SynCode	46.4	87.6	33.02
	IterGen	**47.6**	**89.5**	**32.68**

Privacy:

- 51.4% reduction (from 51.4% to 0%)
- Minimal overhead (4-7 additional generated tokens)
- Minimal impact on response quality (perplexity from 5.66-6.87 to 5.76-7.0)

IterGen code is available at https://github.com/structuredllm/itergen.

[1] SynCode: LLM Generation with Grammar Augmentation
Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, Gagandeep Singh