

XGrammar: Flexible And Efficient Structured Generation Engine for Large Language Models

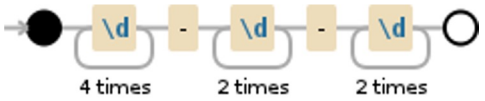
Yixin Dong
Carnegie Mellon University
Oct 16, 2024

Background: Structured Generation for LLMs

Structured Outputs

- Language Code — GitHub Copilot
- JSON Schema — LangChain
- DSL (e.g. PDDL)

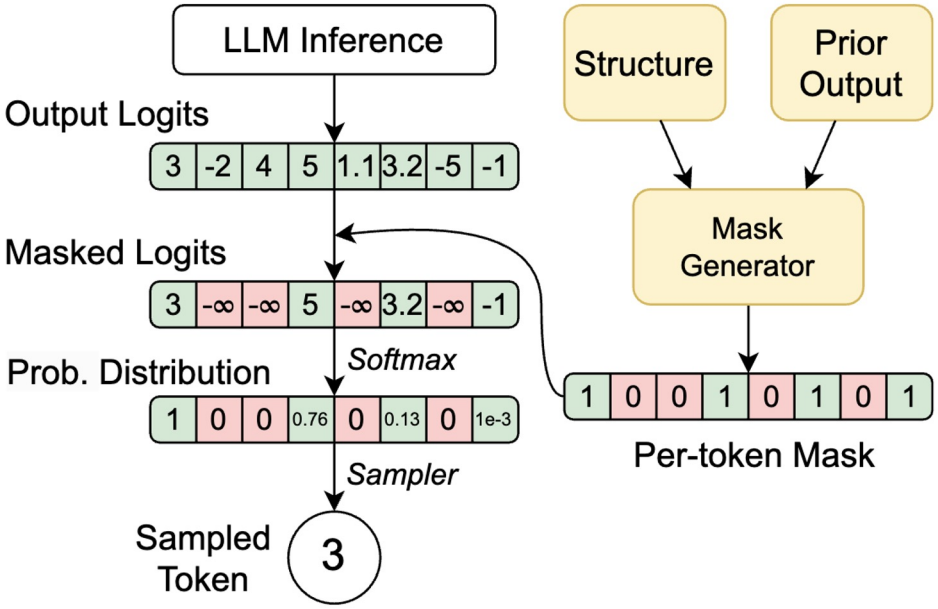
- RegEx



Most of them
can be described as
Context-free Grammar

* Some programming languages like Python can be described as CFG plus maintaining a state

Background: Constrained Decoding



Apply a per-token mask to prevent generating invalid tokens according to the structure

The overhead of the mask generator is crucial!

XGrammar: Flexible and Efficient Structured Generation Engine

XGrammar is a structured generation library that features

Flexibility: Full support for context-free grammar

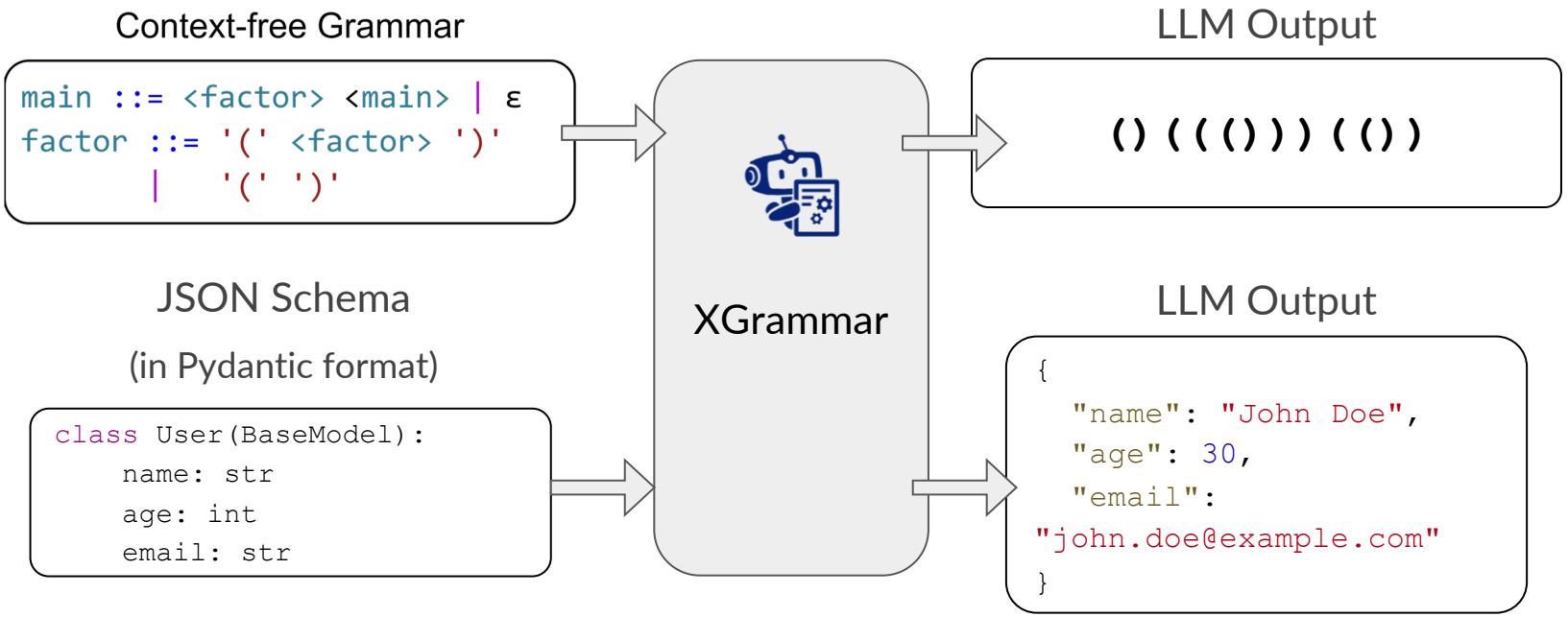
Efficiency: SOTA performance in constraint decoding

Zero-overhead JSON Schema generation

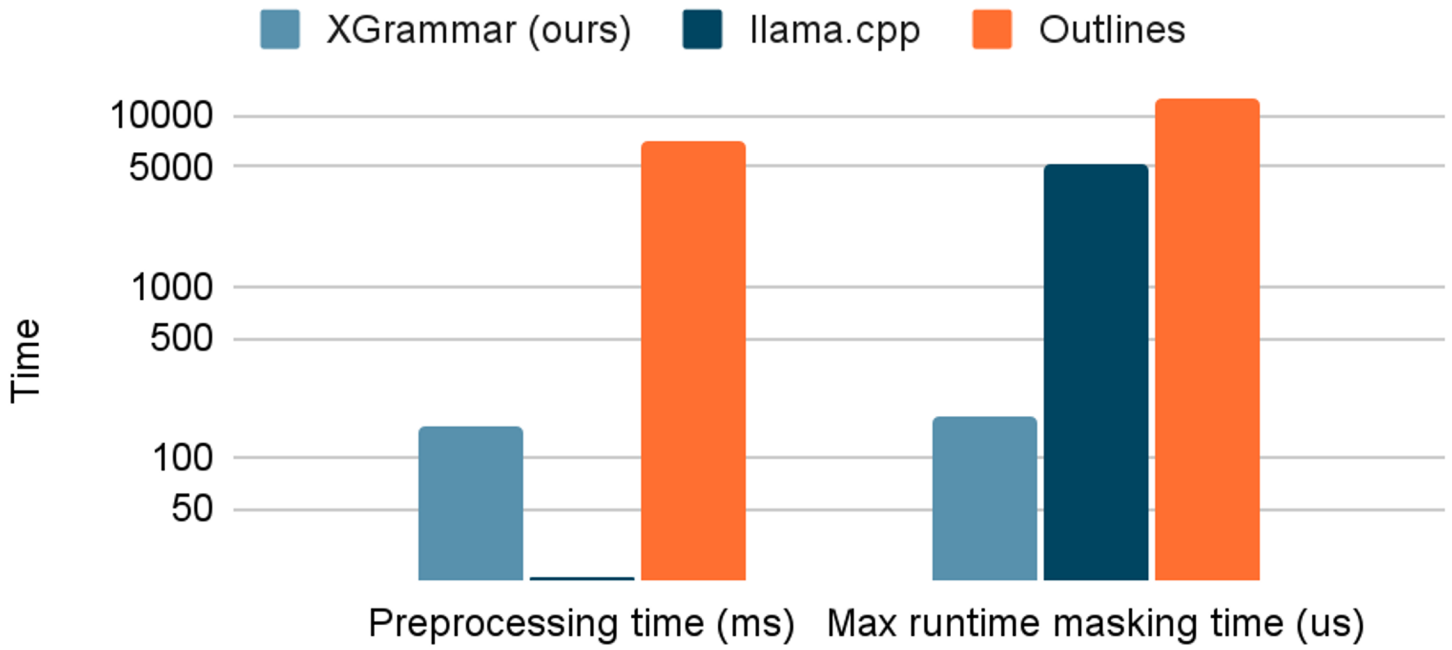
Integration: Easy to integrate with existing LLM serving frameworks

MLC-LLM, SGLang, Web-LLM, etc

Demonstration: CFG or JSON Schema with XGrammar

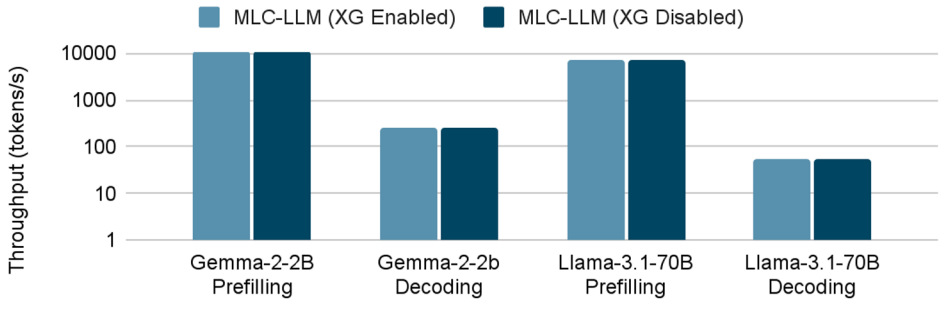


Unit benchmarks



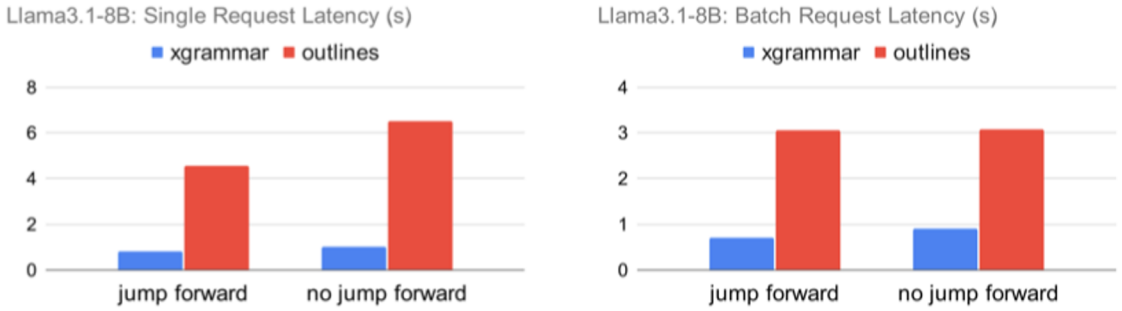
End-to-end benchmarks

MLC-LLM E2E demonstrates zero-overhead constrained decoding



(H100, batch=1)

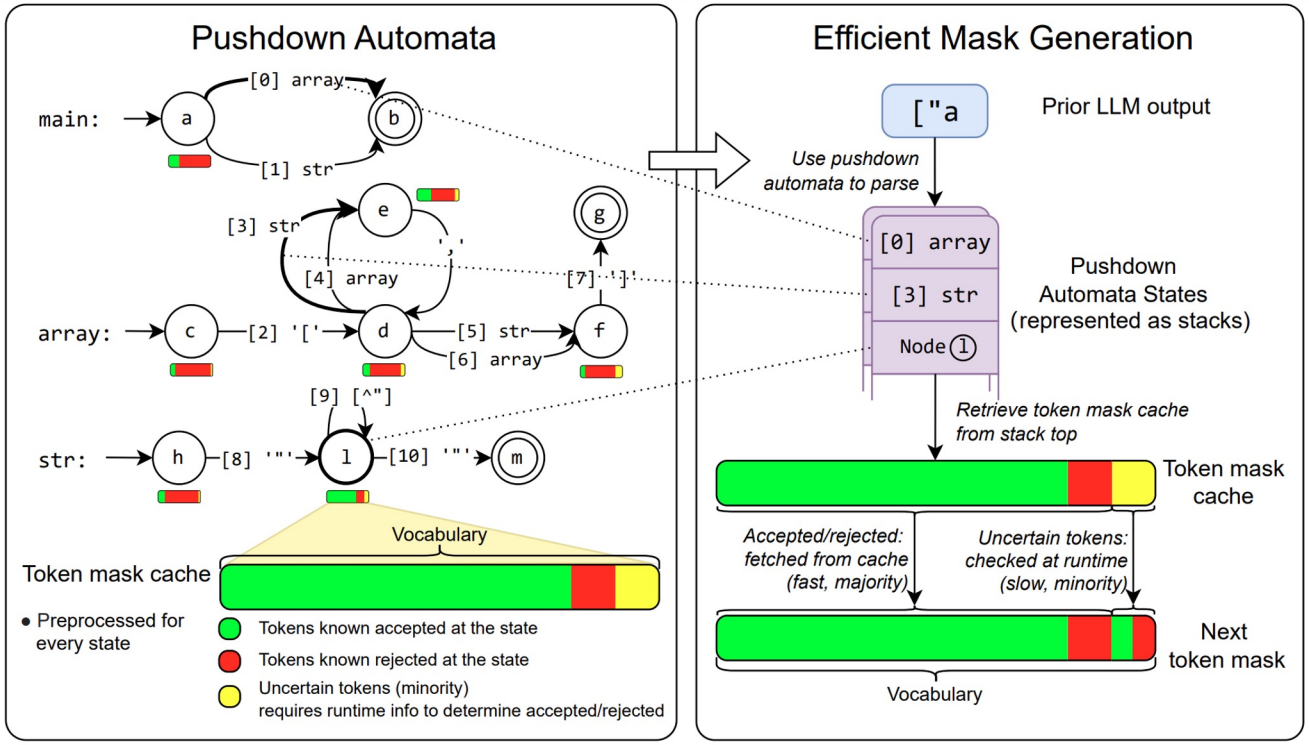
SGLang E2E demonstrates the superiority of XGrammar over other libraries



(A100)

Key optimization: token mask cache

Most part of the mask can be precomputed, although a few remaining ones need to be detected at runtime



Usage: BNFGrammar

```
bnf_grammar = BNFGrammar(
    """main ::= factor main | ""
    factor ::= "(" factor ")" | "()" """,
    main_rule="main",
)
```

Provide the grammar specification in the extended Backus-Naur form (EBNF)

And specify the name of the main rule ("main" by default)

Usage: BuiltinGrammar

```
BuiltinGrammar.json()
BuiltinGrammar.json_schema(schema: str | type[Pydantic.BaseModel])
BuiltinGrammar.str(s: str)
BuiltinGrammar.multiple_choices(choices: List[str])
BuiltinGrammar.concat(grammars: List[BNFGrammar])
BuiltinGrammar.regex(pattern: str)
```

XGrammar provides multiple helper methods to conveniently generate grammar

Usage: GrammarMatcher

```
matcher = GrammarMatcher(bnf_grammar, TokenizerInfo(hf_tokenizer))
while True:
    logits = model.inference(input_tokens)
    bitmask: torch.Tensor = matcher.find_next_token_bitmask()
    GrammarMatcher.apply_bitmask_inplace(logits, bitmask)
    token_id = sampler.sample(logits)
    matcher.accept_token(token_id)
```

At every step, XGrammar provides a token mask to apply to logits, masking invalid tokens.

Each generated token updates the matcher state.

Usage: Caching the preprocessing result

```
init_context = GrammarMatcherInitContext(grammar, tokenizer_info) # 150 ms

matcher = GrammarMatcher(init_context) # 20us
bitmask = matcher.find_next_token_bitmask() # 30us

------------------
init_context_cache = GrammarMatcherInitContextCache(tokenizer_info) # 20us

init_context_1 = init_context_cache.get_init_ctx_for_grammar(grammar) # 150ms
init_context_2 = init_context_cache.get_init_ctx_for_grammar(grammar) # 10us
```

The preprocessing can be slow (100-200 ms)

GrammarMatcherInitContext:

The preprocessing result, can be used to construct GrammarMatcher fast

GrammarMatcherInitContextCache:

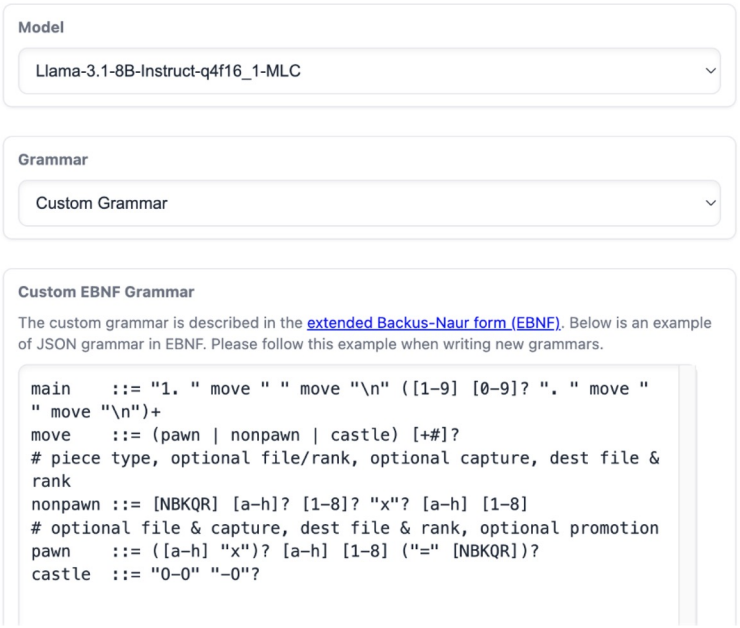
Quickly retrieve the init context of the same grammar multiple times.

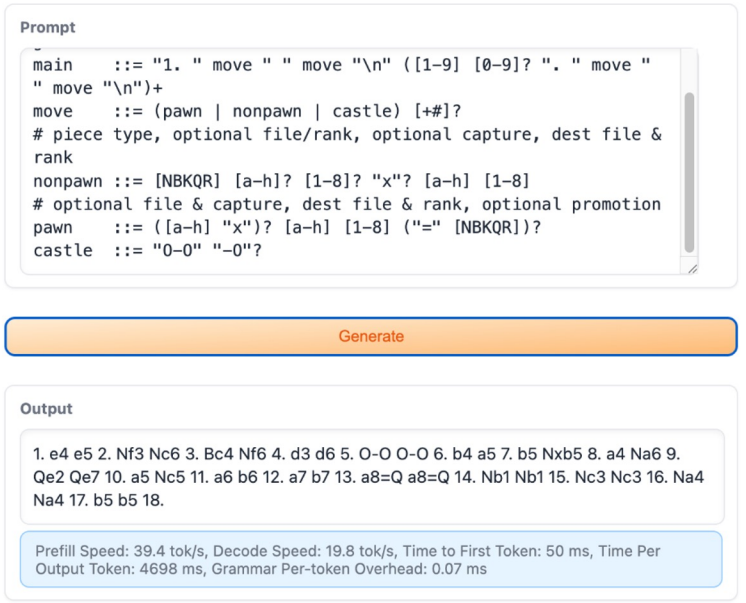
Integration with LLM serving frameworks

- XGrammar is designed for easy integration and cross-platform support (with C++, Python, and TypeScript APIs)
- XGrammar has already been integrated with SGLang, MLC-LLM and WebLLM and is currently being integrated with deepseek
- XGrammar will be open-sourced and released later this month

Try it out on Web-LLM

Run structured generation completely on your web browser with great efficiency!





With help from Charlie Ruan and Nestor Qin from CMU Catalyst Lab



Thanks

Questions are welcome!

Yixin Dong

Oct 16, 2024