Constrained Decoding of Diffusion LLMs with Context-Free Grammars

Niels Mündler, Jasper Dekoninck, Martin Vechev

ICLR 2026

LLMs are used to write code extensively

Completion

Chat

Agent

Microsoft has over a million paying Github Copilot users: CEO Nadella

Written by Tiernan Ray

Oct. 25, 2023 at 5:52 a.m. PT

AI means everyone can now be a programmer, Nvidia chief says

By Reuters

May 29, 2023 3:40 PM PDT

Over 25% of Google's code is now written by AI—and CEO Sundar Pichai says it's just the start

BY GREG MCKENNA

October 30, 2024 at 11:14 AM EDT

ANTHROP\C

37.2% of queries sent to Claude were in the "computer and mathematical" category, which in large part covers software engineering roles.

Feb 10, 2025

Diffusion LLMs are emerging

Diffusion Language Models are Super Data Le

Jinjie Ni[1†], Qian Liu, Longxu Dou[2], Chao Du[2], Zili Wang[3], Hang Yan[4], Tianyu Pang[2], Michael Qizhe Shieh[1]
[1]National University of Singapore, [2]Sea AI Lab, [3]StepFun, [4]Shanghai Qiji Zhifeng Co., Ltd.

Diffusion Beats Autoregressive in Data-Constrained Settings

Mihir Prabhudesai* Carnegie Mellon University
Menging Wu* Carnegie Mellon University
Amir Zadeh Lambda
Katerina Fragkiadaki Carnegie Mellon University
Deepak Pathak Carnegie Mellon University

Performance
Gemini Diffusion's external benchmark performance is comparable to much larger models, whilst also being faster.

Benchmark	Gemini Diffusion	Gemini 2.0 Flash-Lite
Code LiveCodeBench (v6)	30.9%	28.5%

A.I. Is Getting More Powerful, but Its Hallucinations Are Getting Worse

A new wave of "reasoning" systems from companies like OpenAI is producing incorrect information more often. Even the companies don't kno

Forbes

INNOVATION

How AI-Generated Code Is Unleashing A Tsunami Of Security Risks

WIRED SECURITY POLITICS THE BIG

DAN GOODIN, ARS TECHNICA SECURITY APR 30, 2025 3:00 PM

AI Code Hallucinations Increase the Risk of 'Package Confusion' Attacks

A new study found that code generated by AI is more likely to contain made-up information that can be used to trick software into interacting with malicious code.

(Diffusion) LLMs don't leverage formal language rules

Our Work

Leveraging formal grammars for Diffusion LLM generation

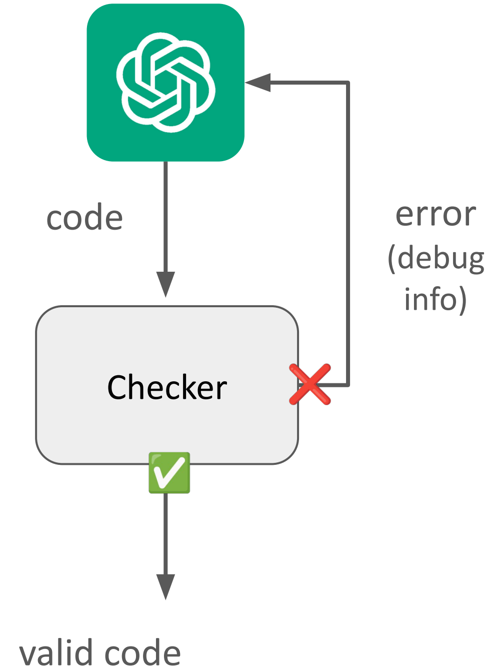
Direct approach: Feedback loop with type checker

Treat syntax checker as a black box:

- No control during LLM decoding

- LLM needs to figure out a valid alternative from the error messages

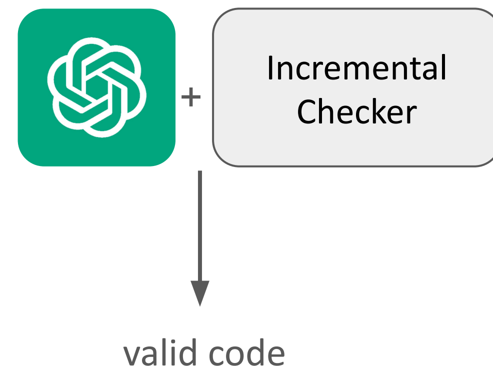
- Unpredictable number of iterations



Our work: Constraining LLMs' decoding

Potential Benefits:

- Full control during decoding
- Generated code is guaranteed to be valid
- Only one iteration



Wanted: new incremental checkers for LLM decoding

Background: dLLM generates code in any order

```
int fib (int n) {
```

Background: dLLM generates code in any order

```
int fib (int n) {

}
```

Background: dLLM generates code in any order

```
int fib (int n) {


            fib(n-1)
}
```

Background: dLLM generates code in any order

```
int fib (int n) {


  return fib(n-1)
}
```

Background: dLLM generates code in any order

```
int fib (int n) {
  if ( n

  return fib(n-1)
}
```

Background: Incremental Constraint Checker detects violations

```
int fib (int n) {
  if ( n == 0

  return fib(n-1)
}
```

Background: Incremental Constraint Checker detects violations

```
int fib (int n) {
  if ( n == 0

  return fib(n-1) )
}
```

Background: Incremental Constraint Checker detects violations

```
int fib (int n) {
  if ( n == 0

  return fib(n-1) )
}
```

Background: Incremental Constraint Checker detects violations

```
int fib (int n) {
  if ( n == 0

  return fib(n-1) +
}
```

Background: Incremental Constraint Checker detects violations

```
int fib (int n) {
  if ( n == 0

  return fib(n-1) +
}
```

How to determine violations?

Is there a completion that is syntactically valid?

```
int fib (int n) {
  if ( n == 0
  
  return fib(n-1) +
}
```

```
int fib (int n) {
  if ( n == 0 )
    return 1
  return fib(n-1) + fib(n-2);
}
```

```
int fib (int n) {
  if ( n == 0
  
  return fib(n-1) )
}
```

Abstracting away…

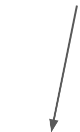
```
int fib (int n) {
  if ( n == 0

  return fib(n-1) )
}
```

Is there a completion that is syntactically valid?

snappify.com

Abstracting away…

Is there a completion that is syntactically valid?



$$L_c$$

All possible completions

```
int fib (int n) {
  if ( n == 0

  return fib(n-1) )
}
```

Abstracting away…

Is there a completion that is syntactically valid?

L_c

All possible completions

```
int fib (int n) {
  if ( n == 0
  
  return fib(n-1) )
}
```

L_s

All C++ programs

```
S → Def() {Lines}
Lines → Line ; Lines | ε
Def → <type> <name>
```

Abstracting away…

Is there a completion that is syntactically valid?

$L_c \cap L_s$

Step 1: Compute the Intersection

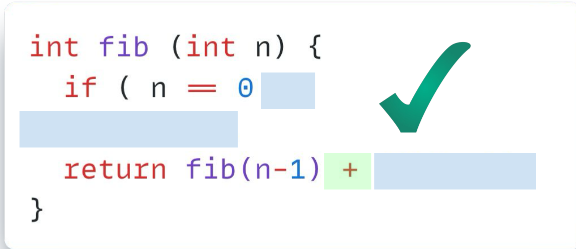
$= \varnothing$?

Step 2: Check if it is empty

How to determine violations?

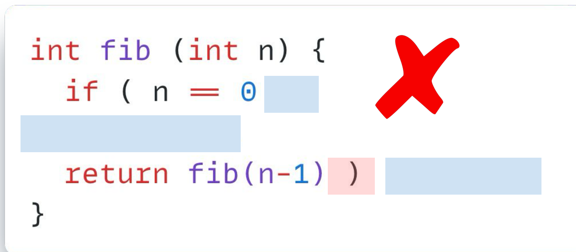
Is there a completion that is syntactically valid?

$$L_c \cap L_s$$



```
int fib (int n) {
  if ( n == 0 )
    return 1
  return fib(n-1) + fib(n-2);
}
```

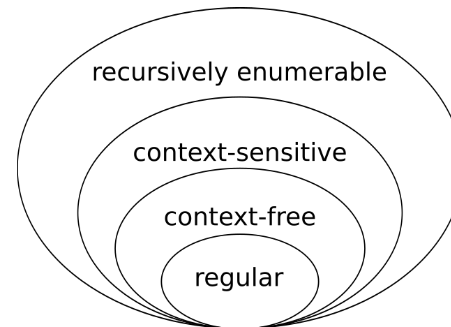
$\{ \ldots, \ldots \}$



$\varnothing$

Abstracting away…

Is there a completion that is syntactically valid?



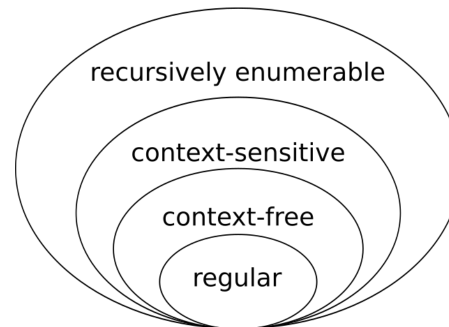
$L_c \cap L_s$ $= \varnothing$?

Step 1: Compute the Intersection

Step 2: Check if it is empty

Abstracting away…

Is there a completion that is syntactically valid?



$L_c \cap L_s$

Step 1: Compute the Intersection

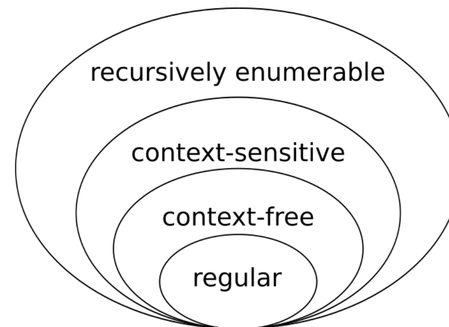
$= \varnothing$?

Step 2: Check if it is empty

Regular	CF	CS	RE
✓	✓	✗	✗

Abstracting away…

Is there a completion that is syntactically valid?



$L_c \cap L_s$

Step 1: Compute the Intersection

$= \varnothing$?

Step 2: Check if it is empty

	Regular	CF
Regular	Regular	CF
CF	CF	CS

Regular	CF	CS	RE
✓	✓	✗	✗

Step 1: Compute the Intersection

Is there a completion that is syntactically valid?

L_c

```
int fib (int n) {
  if ( n == 0 
  
  return fib(n-1) 
}
```

L_s

S ⟶ Def() {Lines}
Lines ⟶ Line ; Lines | ε
Def ⟶ <type> <name>

Step 1: Compute the Intersection

Is there a completion that is syntactically valid?

L_c

Regular Language

```
int fib (int n) {
  if ( n == 0 .*
        .*
  return fib(n-1)      .*
}
```

L_s

```
S → Def() {Lines}
Lines → Line ; Lines | ε
Def → <type> <name>
```

Step 1: Compute the Intersection

Is there a completion that is syntactically valid?

L_c

Regular Language

```
int fib (int n) {
  if ( n == 0 .*
      .*
  return fib(n-1)      .*
}
```

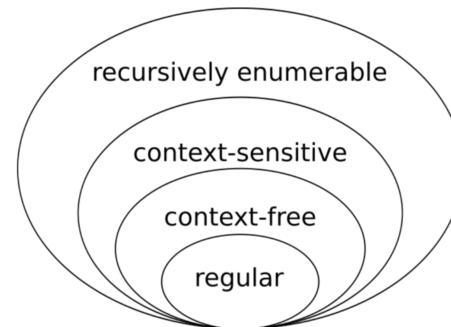
L_s

Context-Free Language

```
S → Def() {Lines}
Lines → Line ; Lines | ε
Def → <type> <name>
```

Abstracting away…

Is there a completion that is syntactically valid?



$L_c \cap L_s$

Step 1: Compute the Intersection

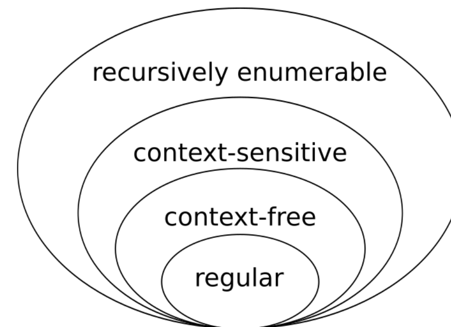
$= \varnothing$?

Step 2: Check if it is empty

Good news: possible

Bad news: expensive! $O(n^4)$ (CFG x Regular x Regular x Regular)

Abstracting away…



Is there a completion that is syntactically valid?

$L_c \cap L_s$

Step 1: Compute the Intersection

$= \varnothing$?

Step 2: Check if it is empty

… so we apply a number of optimizations (ask us about them!)

Size reduction >99%

Experimental Evaluation

Setup

4 Diffusion LLMs

3 Methods

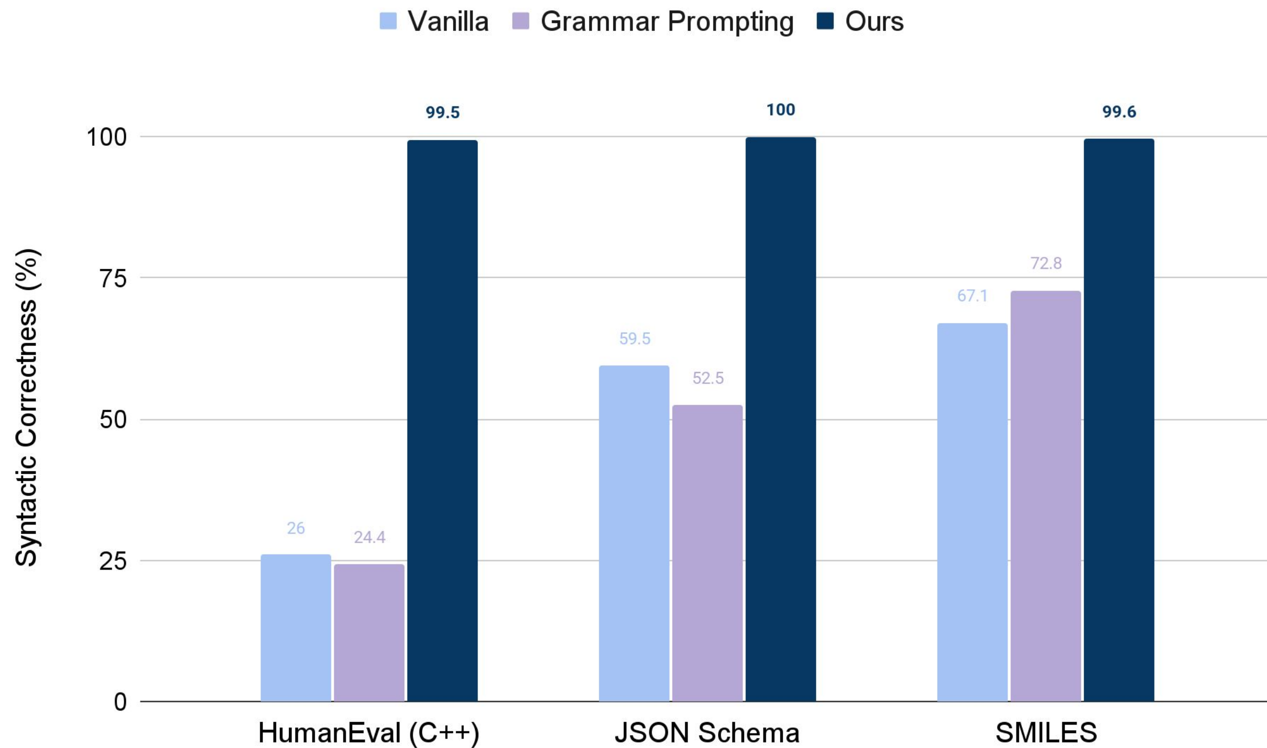
- Vanilla
- Grammar Prompting
- Our Constrained Decoding

3 Tasks

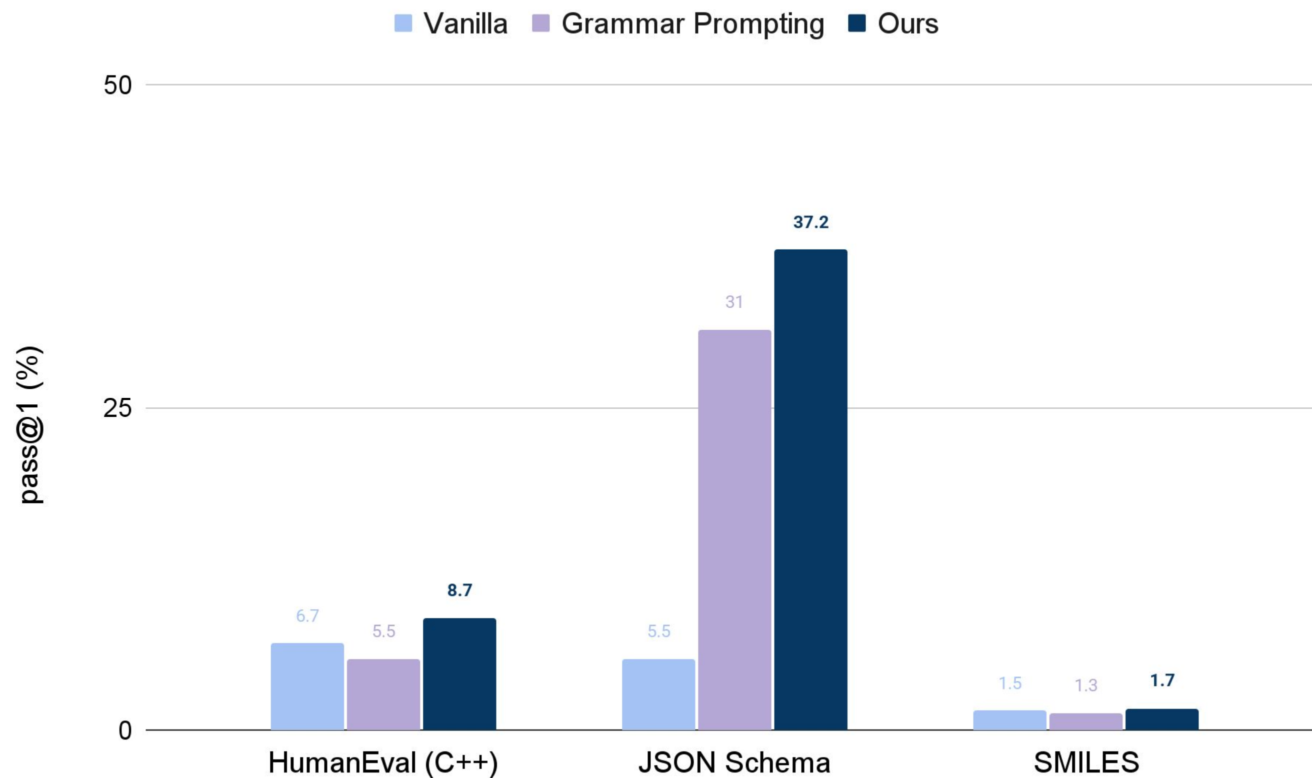
- HumanEval (C++)
- JSON Schema
- SMILES

```
...
The answer must adhere to the following grammar:

S -> Line
Line -> Atom OptComboChainBranchList
OptComboChainBranchList -> ComboChainBranchList | ε
ComboChainBranchList -> ComboChainBranchElement | ComboChainBranchElement
    ComboChainBranchList
ComboChainBranchElement -> Chain | Branch
Chain -> . Atom | OptBond ComboAtomRnumList
OptBond -> Bond | ε
ComboAtomRnumList -> ComboAtomRnumElement | ComboAtomRnumElement ComboAtomRnumList
ComboAtomRnumElement -> Atom | Rnum
Bond -> - | bond
Branch -> ( OptBondOrDotLineList )
OptBondOrDotLineList -> OptBondOrDotLineElement | OptBondOrDotLineElement
```

Near perfect syntactic adherence



Significant functionality improvements



Discussion

Very versatile (any Context Free Grammar)

No precomputation needed (vs. masking approaches [1,2])

Less cost than sampling twice

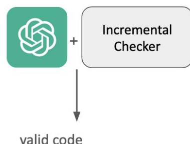
[1] Beurer-Kellner et. al., *Guiding LLMs the right way*, ICML 24
[2] Ugare et. al., *SynCode: LLM Generation with Grammar Augmentation*, TMLR 25

Conclusion

Our work: Constraining LLMs' decoding

Potential Benefits:

- Full control during decoding
- Generated code is guaranteed to be valid
- Only one iteration

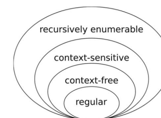


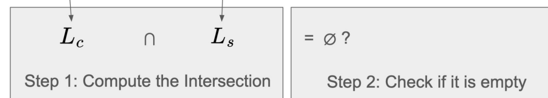
Wanted: new incremental checkers for LLM decoding

12

Abstracting away…

Is there a completion that is syntactically valid?



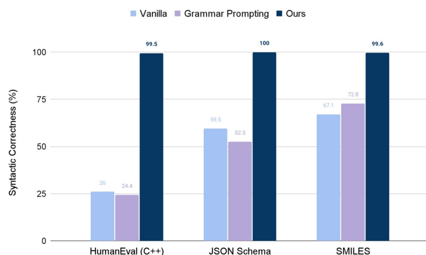


	Regular	CF
Regular	Regular	CF
CF	CF	CS

Regular	CF	CS	RE
O(n)	O(n)	✗	✗

42

Near perfect syntactic adherence



56

Discussion

Less cost than sampling twice

No precomputation needed (vs masking approaches)

Very versatile (any Context Free Grammar)

54

http://constrained-diffusion.ai