SEM-CTRL: Semantically Controlled Decoding

Published in Transactions on Machine Learning Research (03/2026)

Mohammad Albinhassan[1] Pranava Madhyastha[2] Alessandra Russo[1]

[1]Imperial College London [2]City, University of London

Outline

1. Motivation
2. Background
3. Limitations of Existing Approaches
4. SEM-CTRL: Our Approach
5. Experiments
6. Results
7. Conclusion

Motivation

The Promise and Problem of LLMs

Large Language Models are remarkably capable — from language generation to agent control.

But we cannot guarantee what they generate.

In many real-world settings we **know** what valid solutions look like and can evaluate them — yet LLMs routinely violate known constraints.

> **Our goal:** Given a set of rules, how do we make the LLM *strictly* adhere to them while still solving the task correctly?

Two Dimensions of Output Quality

Validity

Does the output satisfy **formal constraints**?

- Syntactic structure
- Context-sensitive rules
- Domain semantics

Correctness

Does the output **solve the task**?

- Achieves the goal state

Existing approaches address one but rarely both. As we are in the language domain, constraints can be naturally expressed using **formal languages** — let's start there.

Background

Answer Set Programming (ASP)

ASP is a declarative logic programming paradigm for knowledge representation and combinatorial problem solving. Used in robotics, bioinformatics, and knowledge representation.

Answer Set Programming (ASP)

ASP is a declarative logic programming paradigm for knowledge representation and combinatorial problem solving. Used in robotics, bioinformatics, and knowledge representation.

Core Syntax

```
% Facts (what is true)
color(red). color(green).
color(blue).

% Rules (derive new knowledge)
h :- b1, b2, ..., bn.
% ``h is true if all bi are true''

% Constraints (reject invalid)
:- b1, b2, ..., bn.
% ``reject if all bi are satisfied''
```

Answer Set Programming (ASP)

ASP is a declarative logic programming paradigm for knowledge representation and combinatorial problem solving. Used in robotics, bioinformatics, and knowledge representation.

Core Syntax

```
% Facts (what is true)
color(red). color(green).
color(blue).
% Rules (derive new knowledge)
h :- b1, b2, ..., bn.
% ``h is true if all bi are true''
% Constraints (reject invalid)
:- b1, b2, ..., bn.
% ``reject if all bi are satisfied''
```

Example: Sudoku in ASP

```
% Fact: cell (1,1) is assigned 4
cell((1,1), 4).
% Rule: cells in the same row
same_row((R,C1),(R,C2)) :- C1 != C2.
% Constraint: no duplicates in a row
:- same_row(C1,C2), cell(C1,N), cell(C2,N).
```

A **symbolic solver** finds all valid assignments (*answer sets*).

Formal Languages

A **formal language** $L \subseteq \Sigma^*$ is a (possibly infinite) set of strings over a finite alphabet Σ, governed by logical rules for syntax and semantics.

Formal Languages

A **formal language** $L \subseteq \Sigma^*$ is a (possibly infinite) set of strings over a finite alphabet Σ, governed by logical rules for syntax and semantics.

Even Binary Numbers

$\Sigma = \{0, 1\}$

$L = \{10, 110, 1010, \ldots\}$

Formal Languages

A **formal language** $L \subseteq \Sigma^*$ is a (possibly infinite) set of strings over a finite alphabet Σ, governed by logical rules for syntax and semantics.

Even Binary Numbers

$\Sigma = \{0, 1\}$

$L = \{10, 110, 1010, \ldots\}$

Python Programs

$\Sigma = \{a, \ldots, z, 0, \ldots, 9, \texttt{for}, \ldots\}$

$L = \{\texttt{print('hello')}, \ldots\}$

Formal Languages

A **formal language** $L \subseteq \Sigma^*$ is a (possibly infinite) set of strings over a finite alphabet Σ, governed by logical rules for syntax and semantics.

Even Binary Numbers

$\Sigma = \{0, 1\}$

$L = \{10, 110, 1010, \ldots\}$

Python Programs

$\Sigma = \{a, \ldots, z, 0, \ldots, 9, \texttt{for}, \ldots\}$

$L = \{\texttt{print('hello')}, \ldots\}$

Chemical Molecules

$\Sigma = \{H, He, Li, \ldots, @, +, -\}$

$L = \{CC, CCO, C1, \ldots\}$

Formal Languages

A **formal language** $L \subseteq \Sigma^*$ is a (possibly infinite) set of strings over a finite alphabet Σ, governed by logical rules for syntax and semantics.

Even Binary Numbers

$\Sigma = \{0, 1\}$

$L = \{10, 110, 1010, \ldots\}$

Python Programs

$\Sigma = \{a, \ldots, z, 0, \ldots, 9, \texttt{for}, \ldots\}$

$L = \{\texttt{print('hello')}, \ldots\}$

Chemical Molecules

$\Sigma = \{H, He, Li, \ldots, @, +, -\}$

$L = \{CC, CCO, C1, \ldots\}$

Languages are generated by **grammars** $G = \langle N, T, P, S \rangle$ with non-terminals N, terminals $T = \Sigma$, production rules P, and start symbol S. Grammars enable building compositional structures from strings.

Grammars: CFG vs CSG

Context-Free Grammar (CFG)

Ensures strings are **syntactically** correct.

Language: $L_1 = \{a^i b^j c^k \mid i, j, k \geq 0\}$

```
start → as bs cs {}
as → "a" as {} | {}
bs → "b" bs {} | {}
cs → "c" cs {} | {}
```

✓ aaabbbccc, abbccc, ac

Any counts allowed — no cross-dependencies.

Context-Sensitive Grammar (CSG)

Captures **context-dependent** patterns that CFGs cannot.

Language: $L_2 = \{a^n b^n c^n \mid n \geq 0\}$

```
start → as bs cs {
:- size(X)@1, not size(X)@2.
:- size(X)@1, not size(X)@3.
}
```

✓ abc, aabbcc

× abbccc (unequal counts)

ASP constraints enforce equal lengths.

Grammars: CFG vs CSG

Context-Free Grammar (CFG)

Ensures strings are **syntactically** correct.

Language: $L_1 = \{a^i b^j c^k \mid i, j, k \geq 0\}$

```
start → as bs cs {}
as → "a" as {} | {}
bs → "b" bs {} | {}
cs → "c" cs {} | {}
```

✓ aaabbbccc, abbccc, ac

Any counts allowed — no cross-dependencies.

Context-Sensitive Grammar (CSG)

Captures **context-dependent** patterns that CFGs cannot.

Language: $L_2 = \{a^n b^n c^n \mid n \geq 0\}$

```
start → as bs cs {
:- size(X)@1, not size(X)@2.
:- size(X)@1, not size(X)@3.
}
```

✓ abc, aabbcc

× abbccc (unequal counts)

ASP constraints enforce equal lengths.

But neither CFGs nor CSGs can express **semantics** — for that, we need Answer Set Grammars.

Answer Set Grammars (ASGs)

ASGs extend CFGs with ASP-based semantic constraints:

$$\text{ASG} = \langle G_{\text{CF}},\ \Psi \rangle \qquad \Psi = \Psi_{PR} \cup \Psi_B$$

- Ψ_{PR}: **Parse-tree constraints** — ASP annotations on production rules (`{...}`)
- Ψ_B: **Background knowledge** — domain rules, state encodings, general facts

Answer Set Grammars (ASGs)

ASGs extend CFGs with ASP-based semantic constraints:

$$\text{ASG} = \langle G_{\text{CF}},\ \Psi \rangle \qquad \Psi = \Psi_{PR} \cup \Psi_B$$

- Ψ_{PR}: **Parse-tree constraints** — ASP annotations on production rules (`{...}`)
- Ψ_B: **Background knowledge** — domain rules, state encodings, general facts

Key property

ASGs express **domain-specific knowledge**, meaningful relationships, and rules that transcend the typical local, positional nature of grammar rules — from syntactic (CFG, $\Psi = \emptyset$) through context-sensitive ($\Psi_{PR} \neq \emptyset$, $\Psi_B = \emptyset$) to full semantic control ($\Psi_B \neq \emptyset$).

Connecting Grammars to LLM Decoding

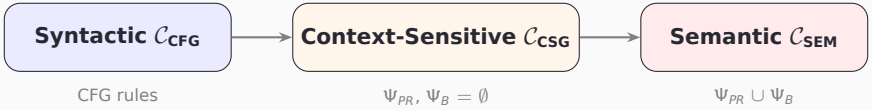
Given a grammar, we compute the **valid next tokens** at each step and mask the LLM accordingly.

Constraint function:

$$\mathcal{C}(y_{<t}) = \{y_t \mid \exists\, w \in L : (y_{<t} \circ y_t) \text{ is a prefix of } w\}$$

Constrained sampling:

$$q_{\mathcal{C}}(y_t \mid x, y_{<t}) \propto p_\theta(y_t \mid x, y_{<t}) \cdot \mathbb{I}[y_t \in \mathcal{C}(y_{<t})]$$



Most prior work stops at $\mathcal{C}_{\text{CFG}}$. SEM-CTRL goes all the way to $\mathcal{C}_{\text{SEM}}$.

Connecting Grammars to LLM Decoding

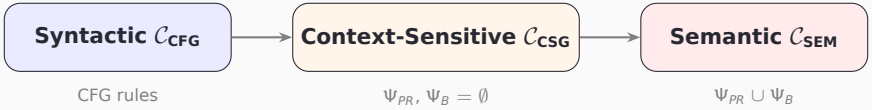
Given a grammar, we compute the **valid next tokens** at each step and mask the LLM accordingly.

Constraint function:

$$\mathcal{C}(y_{<t}) = \{y_t \mid \exists\, w \in L : (y_{<t} \circ y_t) \text{ is a prefix of } w\}$$

Constrained sampling:

$$q_{\mathcal{C}}(y_t \mid x, y_{<t}) \propto p_\theta(y_t \mid x, y_{<t}) \cdot \mathbb{I}[y_t \in \mathcal{C}(y_{<t})]$$



Most prior work stops at $\mathcal{C}_{\text{CFG}}$. SEM-CTRL goes all the way to $\mathcal{C}_{\text{SEM}}$.

Connecting Grammars to LLM Decoding

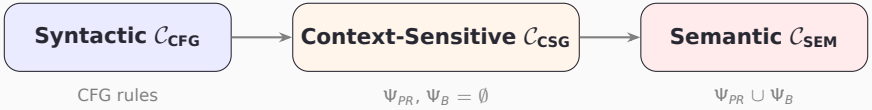
Given a grammar, we compute the **valid next tokens** at each step and mask the LLM accordingly.

Constraint function:

$$\mathcal{C}(y_{<t}) = \{y_t \mid \exists\, w \in L : (y_{<t} \circ y_t) \text{ is a prefix of } w\}$$

Constrained sampling:

$$q_{\mathcal{C}}(y_t \mid x, y_{<t}) \propto p_\theta(y_t \mid x, y_{<t}) \cdot \mathbb{I}[y_t \in \mathcal{C}(y_{<t})]$$



Most prior work stops at $\mathcal{C}_{\text{CFG}}$. `SEM-CTRL` goes all the way to $\mathcal{C}_{\text{SEM}}$.

Connecting Grammars to LLM Decoding

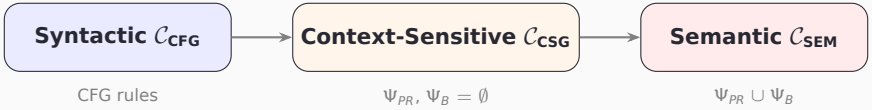
Given a grammar, we compute the **valid next tokens** at each step and mask the LLM accordingly.

Constraint function:

$$\mathcal{C}(y_{<t}) = \{y_t \mid \exists\, w \in L : (y_{<t} \circ y_t) \text{ is a prefix of } w\}$$

Constrained sampling:

$$q_{\mathcal{C}}(y_t \mid x, y_{<t}) \propto p_\theta(y_t \mid x, y_{<t}) \cdot \mathbb{I}[y_t \in \mathcal{C}(y_{<t})]$$



Most prior work stops at $\mathcal{C}_{\text{CFG}}$. SEM-CTRL goes all the way to $\mathcal{C}_{\text{SEM}}$.

Limitations of Existing Approaches

Why Current Methods Fall Short

Controlled Generation

- **Syntactic control** (CFG-constrained decoding)
  Cannot handle context-dependent or semantic correctness
- **Domain-specific solutions**
  CFGs + ad-hoc constraints — lack generalizability
- Focus on *validity* only — no notion of *correctness*

Why Current Methods Fall Short

Controlled Generation

- **Syntactic control** (CFG-constrained decoding)
  Cannot handle context-dependent or semantic correctness
- **Domain-specific solutions**
  CFGs + ad-hoc constraints — lack generalizability
- Focus on *validity* only — no notion of *correctness*

Search-Guided Reasoning

- **Unconstrained MCTS / ToT**
  Huge token space $\Rightarrow$ aggressive pruning risks eliminating valid solutions
- Cannot guarantee semantic validity

Why Current Methods Fall Short

Controlled Generation

- **Syntactic control** (CFG-constrained decoding)
  Cannot handle context-dependent or semantic correctness
- **Domain-specific solutions**
  CFGs + ad-hoc constraints — lack generalizability
- Focus on *validity* only — no notion of *correctness*

Search-Guided Reasoning

- **Unconstrained MCTS / ToT**
  Huge token space $\Rightarrow$ aggressive pruning risks eliminating valid solutions
- Cannot guarantee semantic validity

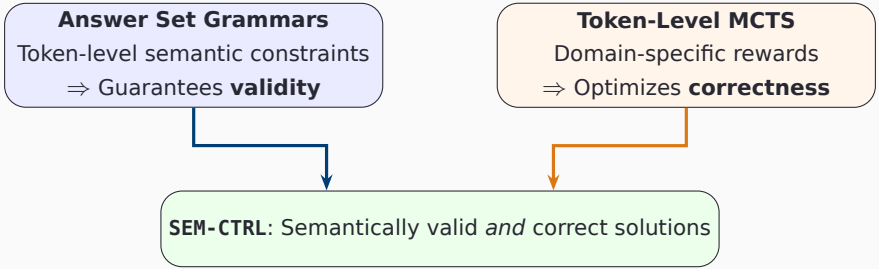
Symbolic Solver Approaches

- Translate problem to formal specification, solve externally
- **Translation errors** are a fundamental bottleneck

SEM-CTRL: Our Approach

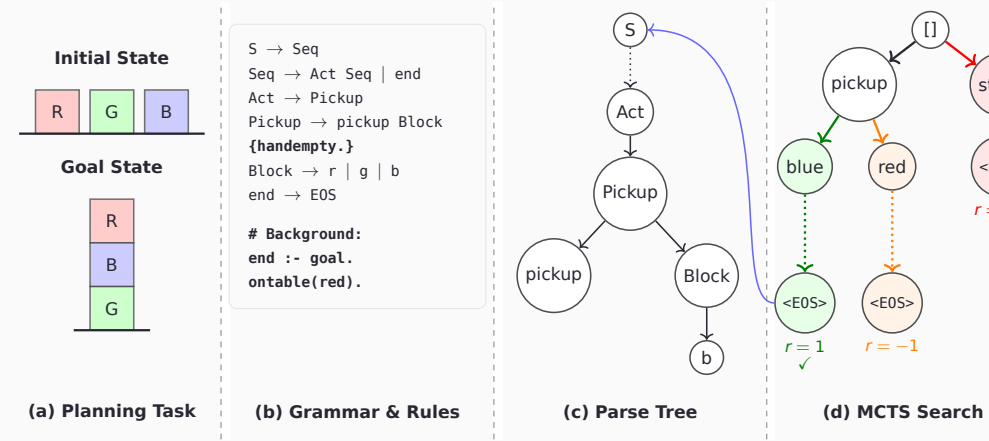
SEM-CTRL: The Big Picture

SEM-CTRL unifies semantic constraints with guided search:



Constraints guarantee validity **by construction**; tree search explores the semantically valid token space to find correct solutions.

Method Overview (Figure 1 from the paper)

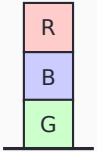


(a) Task with initial/goal states. (b) ASG with syntax and semantic rules. (c) Partial parse tree. (d) MCTS over token space: correct (✓), suboptimal (-1), invalid (×). Blue arrow: MCTS–parse tree correspondence.

Validity vs Correctness: Blocksworld Example

Initial State

Goal State



Syntactically valid:

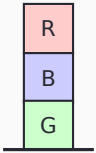
`pickup red, pickup green, stack blue green`

Follows grammar but violates preconditions (can’t hold two blocks)

Validity vs Correctness: Blocksworld Example

Initial State

Goal State



Syntactically valid:

`pickup red, pickup green, stack blue green`

Follows grammar but violates preconditions (can’t hold two blocks)

Semantically valid:

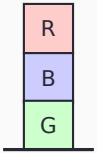
`pickup blue, stack blue red, pickup green, stack green blue`

All preconditions satisfied but doesn’t reach goal

Validity vs Correctness: Blocksworld Example

Initial State

Goal State



Syntactically valid:

`pickup red, pickup green, stack blue green`

Follows grammar but violates preconditions (can't hold two blocks)

Semantically valid:

`pickup blue, stack blue red, pickup green, stack green blue`

All preconditions satisfied but doesn't reach goal

Valid & Correct:

`pickup blue, stack blue green, pickup red, stack red blue, end`

Achieves goal — this is what `SEM-CTRL` finds

`SEM-CTRL` guarantees validity via ASG and efficiently searches for correctness via MCTS.

ASG for Blocksworld

Grammar & Semantic Rules

S → Seq

Seq → Act Seq | end

Act → Pickup

Pickup → pickup Block

{handempty.}

Block → r | g | b

end → EOS

 # Background (Ψ_B):

end :- goal.

ontable(red).

ASG for Blocksworld

Grammar & Semantic Rules

S → Seq

Seq → Act Seq | end

Act → Pickup

Pickup → pickup Block **{handempty.}**

Block → r | g | b

end → EOS

 # Background (Ψ_B):

end :- goal.

ontable(red).

What each component does:

Ψ_{PR}: {handempty.}

Parse-tree constraint — can only pick up if hand is empty

ASG for Blocksworld

Grammar & Semantic Rules

S → Seq

Seq → Act Seq | end

Act → Pickup

Pickup → pickup Block

{handempty.}

Block → r | g | b

end → EOS

 # Background (Ψ_B):

end :- goal.

ontable(red).

What each component does:

Ψ_{PR}: {handempty.}

Parse-tree constraint — can only pick up if hand is empty

Ψ_B: Background knowledge

Domain rules (e.g., `end :- goal.`), state tracking, initial state encoding

ASG for Blocksworld

Grammar & Semantic Rules

S → Seq

Seq → Act Seq | end

Act → Pickup

Pickup → pickup Block **{handempty.}**

Block → r | g | b

end → EOS

 # Background (Ψ_B):

end :- goal.

ontable(red).

What each component does:

Ψ_{PR}: `{handempty.}`

Parse-tree constraint — can only pick up if hand is empty

Ψ_B: Background knowledge

Domain rules (e.g., `end :- goal.`), state tracking, initial state encoding

The ASG is **task-specific** (works for any Blocksworld instance), not instance-specific. MCTS + rewards handle instance goals.

Semantically Valid Completions

At each decoding step, the ASG computes valid next tokens:

$$\mathcal{C}_{\text{ASG}}(y_{<t}) = \{y_t \in T \mid \exists\, \delta \in \Delta(y_{<t}),\ \delta \oplus y_t \in \Delta(y_{<t} \circ y_t)\}$$

- $\Delta(y_{<t})$: partial parse trees consistent with prefix $y_{<t}$
- $\delta \oplus y_t$: extending parse tree δ by one token; checked against all ASP constraints Ψ

Semantically Valid Completions

At each decoding step, the ASG computes valid next tokens:

$$\mathcal{C}_{\text{ASG}}(y_{<t}) = \{y_t \in T \mid \exists\, \delta \in \Delta(y_{<t}),\ \delta \oplus y_t \in \Delta(y_{<t} \circ y_t)\}$$

- $\Delta(y_{<t})$: partial parse trees consistent with prefix $y_{<t}$
- $\delta \oplus y_t$: extending parse tree δ by one token; checked against all ASP constraints Ψ

Guarantee

By restricting to $\mathcal{C}_{\text{ASG}}$ at every step, every prefix remains extendable $\Rightarrow$ final output **guaranteed** in $L(G_{\text{ASG}})$.

Semantically Valid Completions

At each decoding step, the ASG computes valid next tokens:

$$\mathcal{C}_{\text{ASG}}(y_{<t}) = \{y_t \in T \mid \exists\, \delta \in \Delta(y_{<t}),\ \delta \oplus y_t \in \Delta(y_{<t} \circ y_t)\}$$

- $\Delta(y_{<t})$: partial parse trees consistent with prefix $y_{<t}$
- $\delta \oplus y_t$: extending parse tree δ by one token; checked against all ASP constraints Ψ

Guarantee

By restricting to $\mathcal{C}_{\text{ASG}}$ at every step, every prefix remains extendable $\Rightarrow$ final output **guaranteed** in $L(G_{\text{ASG}})$.

Practical impact

Valid children per step: **1–15 tokens**
vs. entire vocabulary: **128k+ tokens**
$\Rightarrow$ Massive search space reduction.

Vocabulary Alignment

ASG terminals T and LLM vocabulary V don't always align — multiple LLM tokens may compose a single terminal and vice versa.

We formalize alignment through bidirectional mappings:

$$\tau : T^* \to V^* \qquad \tau^{-1} : V^* \to T^* \cup \{\bot\}$$

satisfying: $\forall s \in T^* : \tau^{-1}(\tau(s)) = s$ (mapping consistency)

Vocabulary Alignment

ASG terminals T and LLM vocabulary V don't always align — multiple LLM tokens may compose a single terminal and vice versa.

We formalize alignment through bidirectional mappings:

$$\tau : T^* \to V^* \qquad \tau^{-1} : V^* \to T^* \cup \{\bot\}$$

satisfying: $\forall s \in T^* : \tau^{-1}(\tau(s)) = s$ (mapping consistency)

> This alignment ensures that ASG-level constraints translate faithfully to token-level masks, regardless of how the LLM's tokenizer splits terminals.

Token-Level Decoding as an MDP

We formulate sequence generation as a **Markov Decision Process**:

- **States** $s \in \mathcal{S}$: partial generation $(x, y_{<t})$ — input prompt + tokens so far
- **Actions** $a \in \mathcal{A}$: selecting a token from vocabulary V
- **Transitions** $T(s, a) = (x, y_{<t} \circ a)$: deterministically append token
- **Policy** $\pi_\theta(a \mid s_t) = p_\theta(a \mid x, y_{<t})$: the LLM itself

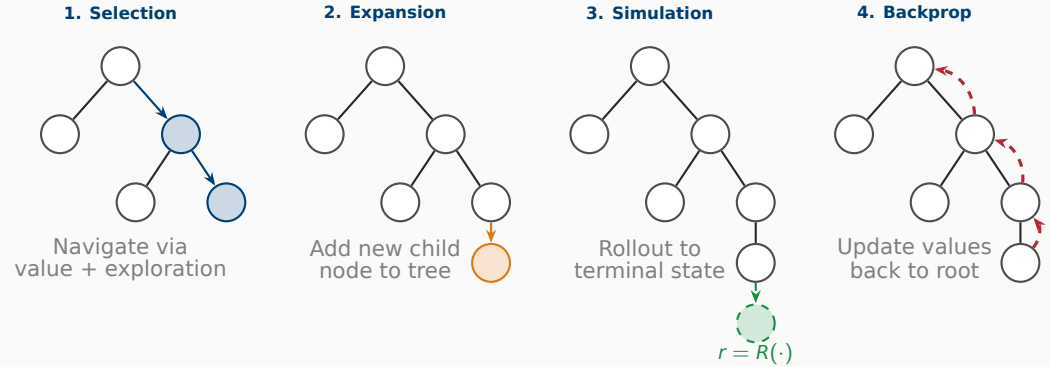
Domain-Specific Reward:

$$R(s_t, a) = \begin{cases} 1 & \text{if } y_{<t} \circ a \in L(G_{\text{ASG}}) \wedge \rho(y_{<t} \circ a) = 0 \\ -\rho(y_{<t} \circ a) & \text{otherwise} \end{cases}$$

where $\rho(\cdot)$ is a task-specific distance-to-goal. We use **explicit** domain-specific rewards, unlike prior work relying on LLM-based approximate rewards.

Monte-Carlo Tree Search (MCTS): A Primer

MCTS is a search algorithm that builds a search tree incrementally through four repeated steps:



In our setting: each **node** is a token sequence, each **edge** is a token choice, and **rollouts** generate complete sequences scored by domain-specific rewards.

Semantically Guided MCTS

Three key modifications to standard MCTS:

1. Constrained Selection — guide via $q_{\mathcal{C}_{\text{ASG}}}$:

$$\arg\max_a \; Q(s_t, a) + U(s_t, a) \qquad U(s_t, a) = \beta(s) \cdot q_{\mathcal{C}_{\text{ASG}}}(a \mid s_{<t}) \cdot \frac{\sqrt{\sum_b N(s_t, b)}}{1 + N(s_t, a)}$$

Semantically Guided MCTS

Three key modifications to standard MCTS:

1. **Constrained Selection** — guide via $q_{\mathcal{C}_{\text{ASG}}}$:

$$\arg\max_a \; Q(s_t, a) + U(s_t, a) \qquad U(s_t, a) = \beta(s) \cdot q_{\mathcal{C}_{\text{ASG}}}(a \mid s_{<t}) \cdot \frac{\sqrt{\sum_b N(s_t, b)}}{1 + N(s_t, a)}$$

2. **Semantic Expansion** — only valid children from $\mathcal{C}_{\text{ASG}}$ (branching factor 1–15)

Semantically Guided MCTS

Three key modifications to standard MCTS:

1. **Constrained Selection** — guide via $q_{\mathcal{C}_{\text{ASG}}}$:

$$\arg\max_a \; Q(s_t, a) + U(s_t, a) \qquad U(s_t, a) = \beta(s) \cdot q_{\mathcal{C}_{\text{ASG}}}(a \mid s_{<t}) \cdot \frac{\sqrt{\sum_b N(s_t, b)}}{1 + N(s_t, a)}$$

2. **Semantic Expansion** — only valid children from $\mathcal{C}_{\text{ASG}}$ (branching factor 1–15)

3. **Controlled Rollouts** — $y \sim \prod_t q_{\mathcal{C}_{\text{ASG}}}(\cdot \mid s_t)$ — every rollout is valid by construction

Semantically Guided MCTS

Three key modifications to standard MCTS:

1. **Constrained Selection** — guide via $q_{\mathcal{C}_{\text{ASG}}}$:

$$\arg\max_a \; Q(s_t, a) + U(s_t, a) \qquad U(s_t, a) = \beta(s) \cdot q_{\mathcal{C}_{\text{ASG}}}(a \mid s_{<t}) \cdot \frac{\sqrt{\sum_b N(s_t, b)}}{1 + N(s_t, a)}$$

2. **Semantic Expansion** — only valid children from $\mathcal{C}_{\text{ASG}}$ (branching factor 1–15)

3. **Controlled Rollouts** — $y \sim \prod_t q_{\mathcal{C}_{\text{ASG}}}(\cdot \mid s_t)$ — every rollout is valid by construction

Why this matters:

Token Pruning

Branching factor reduced from **128k+** to **1–15** tokens per step

Semantically Guided MCTS

Three key modifications to standard MCTS:

1. **Constrained Selection** — guide via $q_{\mathcal{C}_{\text{ASG}}}$:

$$\arg\max_a \; Q(s_t, a) + U(s_t, a) \qquad U(s_t, a) = \beta(s) \cdot q_{\mathcal{C}_{\text{ASG}}}(a \mid s_{<t}) \cdot \frac{\sqrt{\sum_b N(s_t, b)}}{1 + N(s_t, a)}$$

2. **Semantic Expansion** — only valid children from $\mathcal{C}_{\text{ASG}}$ (branching factor 1–15)

3. **Controlled Rollouts** — $y \sim \prod_t q_{\mathcal{C}_{\text{ASG}}}(\cdot \mid s_t)$ — every rollout is valid by construction

Why this matters:

Token Pruning

Branching factor reduced from **128k+** to **1–15** tokens per step

Deep Search

Enables tractable exploration at significant depths

Semantically Guided MCTS

Three key modifications to standard MCTS:

1. **Constrained Selection** — guide via $q_{\mathcal{C}_{\text{ASG}}}$:

$$\arg\max_a \; Q(s_t, a) + U(s_t, a) \qquad U(s_t, a) = \beta(s) \cdot q_{\mathcal{C}_{\text{ASG}}}(a \mid s_{<t}) \cdot \frac{\sqrt{\sum_b N(s_t, b)}}{1 + N(s_t, a)}$$

2. **Semantic Expansion** — only valid children from $\mathcal{C}_{\text{ASG}}$ (branching factor 1–15)

3. **Controlled Rollouts** — $y \sim \prod_t q_{\mathcal{C}_{\text{ASG}}}(\cdot \mid s_t)$ — every rollout is valid by construction

Why this matters:

Token Pruning

Branching factor reduced from **128k+** to **1–15** tokens per step

Deep Search

Enables tractable exploration at significant depths

Reachability

If a correct solution exists at depth d, it remains discoverable

Handling Computational Overheads

Three complementary optimization strategies:

1. Caching Partial ASG Derivations
Cache partial parse trees δ to avoid recomputing $\mathcal{C}_{\text{ASG}}(y_{<t})$ for seen prefixes.

2. Semantic Tree Pruning
Unlike methods relying on arbitrary top-k sampling, our semantic constraints maintain a **small branching factor** (1–15) while preserving solution reachability — if a correct solution exists at depth d, it remains discoverable.

3. Tree-Structure Caching
Reuse stored expansions and rollouts for recurring prefixes $y_{<t}$ across MCTS iterations. Only pay for expansions once per node.

Experiments

Tasks

Class	Task	Constraint	$\rho(\cdot)$
Synthetic Grammar Synthesis (SGS)	$a^n b^n c^n$	Equal count of a, b, c	$n - n_w$
	$a^m b^n c^m d^n$, $m \neq n$	Paired counts	$(m+n) - (m_w+n_w)$
	$w \mid w$ (Copy)	Exact string replication	Dist. to targets
Combinatorial Reasoning (CR)	Sudoku 3×3	Row, column	0 if solved, 1 else
	Sudoku 4×4	Row, column, box	0 if solved, 1 else
	3-Graph Coloring	Valid 3-color assignment	$e - e_w$
Planning	Blocksworld	State validity + preconditions	$h(s, g) + \alpha \cdot \text{len}$
Parsing	JSON	CFG structure	—

Models and Baselines

SEM-CTRL evaluated with: Llama 3.2 1B, Llama 3.1 8B, Llama 3.1 70B

Models and Baselines

SEM-CTRL evaluated with: Llama 3.2 1B, Llama 3.1 8B, Llama 3.1 70B

Baselines:

- **Base (Greedy):** Unconstrained greedy decoding
- **Best-of-N (BoN):** Sample N generations, reject invalid, rank by reward
  N matched to SEM-CTRL's computational budget for fair comparison
- **SOTA Reasoning Models:** o1-preview, DeepSeek-R1, o4-mini
- Full ablation across all algorithm–constraint combinations

Models and Baselines

SEM-CTRL evaluated with: Llama 3.2 1B, Llama 3.1 8B, Llama 3.1 70B

Baselines:

- **Base (Greedy):** Unconstrained greedy decoding
- **Best-of-N (BoN):** Sample N generations, reject invalid, rank by reward
  N matched to SEM-CTRL's computational budget for fair comparison
- **SOTA Reasoning Models:** o1-preview, DeepSeek-R1, o4-mini
- Full ablation across all algorithm–constraint combinations

Evaluation Metrics:

- **A**: Task-specific accuracy (correctness)
- $\mathbf{V}_{\text{CFG}}$: CFG validity $\mathbf{V}_{\text{CSG}}$: CSG validity $\mathbf{V}_{\text{SEM}}$: Semantic validity

Results

How does SEM-CTRL perform overall?

		Synthetic Grammar Synthesis			Combinatorial Reasoning			Planning
Alg.	**Model**	$a^n b^n c^n$	$a^m b^n c^m d^n$	Copy	Graph	Sud-3	Sud-4	Blocks
Base	Llama 1B	3.3	0.0	10.0	0.0	0.0	0.0	0.0
	Llama 70B	30.0	0.0	60.0	37.5	90.0	30.0	23.2
BoN	Llama 1B	7.8	1.1	48.9	0.0	0.0	0.0	4.3
	Llama 70B	71.1	22.2	88.9	**100.0**	96.7	80.0	48.8
API	o1-preview	83.3	80.0	96.7	75.0	**100.0**	**100.0**	94.5
	DeepSeek-R1	83.3	70.0	96.7	75.0	**100.0**	**100.0**	**96.5**
	o4-mini	93.3	93.3	**100.0**	75.0	**100.0**	**100.0**	**98.5**
SEM-CTRL	Llama 1B	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	74.0
	Llama 70B	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**96.8**

SEM-CTRL achieves **100%** on all SGS and CR tasks, even with Llama 1B. On Blocksworld (70B), matches SOTA reasoning models.

How does SEM-CTRL perform overall?

		Synthetic Grammar Synthesis			Combinatorial Reasoning			Planning
Alg.	**Model**	$a^n b^n c^n$	$a^m b^n c^m d^n$	Copy	Graph	Sud-3	Sud-4	Blocks
Base	Llama 1B	3.3	0.0	10.0	0.0	0.0	0.0	0.0
	Llama 70B	30.0	0.0	60.0	37.5	90.0	30.0	23.2
BoN	Llama 1B	7.8	1.1	48.9	0.0	0.0	0.0	4.3
	Llama 70B	71.1	22.2	88.9	**100.0**	96.7	80.0	48.8
API	o1-preview	83.3	80.0	96.7	75.0	**100.0**	**100.0**	94.5
	DeepSeek-R1	83.3	70.0	96.7	75.0	**100.0**	**100.0**	**96.5**
	o4-mini	93.3	93.3	**100.0**	75.0	**100.0**	**100.0**	**98.5**
SEM-CTRL	Llama 1B	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	74.0
	Llama 70B	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**96.8**

SEM-CTRL dramatically lifts model performance — Llama **1B** jumps from near 0% to **100%** on all SGS and CR tasks, surpassing even Base + BoN 70B across the board.

How does SEM-CTRL perform overall?

		Synthetic Grammar Synthesis			Combinatorial Reasoning			Planning
Alg.	**Model**	$a^n b^n c^n$	$a^m b^n c^m d^n$	Copy	Graph	Sud-3	Sud-4	Blocks
Base	Llama 1B	3.3	0.0	10.0	0.0	0.0	0.0	0.0
	Llama 70B	30.0	0.0	60.0	37.5	90.0	30.0	23.2
BoN	Llama 1B	7.8	1.1	48.9	0.0	0.0	0.0	4.3
	Llama 70B	71.1	22.2	88.9	**100.0**	96.7	80.0	48.8
API	o1-preview	83.3	80.0	96.7	75.0	**100.0**	**100.0**	94.5
	DeepSeek-R1	83.3	70.0	96.7	75.0	**100.0**	**100.0**	**96.5**
	o4-mini	93.3	93.3	**100.0**	75.0	**100.0**	**100.0**	**98.5**
SEM-CTRL	Llama 1B	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	74.0
	Llama 70B	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**96.8**

SEM-CTRL **outperforms** SOTA reasoning models on all tasks except Blocksworld, where it **matches** them — while **guaranteeing** semantic validity, which no baseline can.

Can SEM-CTRL guarantee grammatical validity?

Alg.	Model	A	V_{CFG}	V_{CSG}
Base	1B	4.4	79	23
	70B	30.0	99	39
BoN	1B	19.3	65	35
	70B	60.7	97	79
API	o1-prev	86.7	**100**	88
	DS-R1	83.3	**100**	87
	o4-mini	94.7	99	95
SEM-CTRL	1B	**100**	**100**	**100**
	70B	**100**	**100**	**100**

Synthetic Grammar Synthesis task results

Can SEM-CTRL guarantee grammatical validity?

Alg.	Model	A	$\mathbf{V}_{\mathrm{CFG}}$	$\mathbf{V}_{\mathrm{CSG}}$
Base	1B	4.4	79	23
	70B	30.0	99	39
BoN	1B	19.3	65	35
	70B	60.7	97	79
API	o1-prev	86.7	**100**	88
	DS-R1	83.3	**100**	87
	o4-mini	94.7	99	95
SEM-CTRL	1B	**100**	**100**	**100**
	70B	**100**	**100**	**100**

Synthetic Grammar Synthesis task results

All models struggle much more with V_{CSG} than V_{CFG}:

- 1B: 79% V_{CFG} but only 23% V_{CSG}
- Even DS-R1: 100% V_{CFG} but 87% V_{CSG}

Two failure modes: inability to capture constraints *and* failure to ensure correctness.

Can SEM-CTRL guarantee grammatical validity?

Alg.	Model	A	$\mathbf{V}_{\mathrm{CFG}}$	$\mathbf{V}_{\mathrm{CSG}}$
Base	1B	4.4	79	23
	70B	30.0	99	39
BoN	1B	19.3	65	35
	70B	60.7	97	79
API	o1-prev	86.7	**100**	88
	DS-R1	83.3	**100**	87
	o4-mini	94.7	99	95
SEM-CTRL	1B	**100**	**100**	**100**
	70B	**100**	**100**	**100**

Synthetic Grammar Synthesis task results

All models struggle much more with V_{CSG} than V_{CFG}:

- 1B: 79% V_{CFG} but only 23% V_{CSG}
- Even DS-R1: 100% V_{CFG} but 87% V_{CSG}

Two failure modes: inability to capture constraints *and* failure to ensure correctness.

> LLMs can approximate constraints but **lack robustness**. Explicit semantic control is crucial for reliable generation.

How does each component contribute?

Alg.	$\mathcal{C}$	A
Base	—	30%
	$\mathcal{C}_{\text{CFG}}$	30%
	$\mathcal{C}_{\text{SEM}}$	40%
BoN	—	66%
	$\mathcal{C}_{\text{CFG}}$	66%
	$\mathcal{C}_{\text{SEM}}$	90%
MCTS	—	46%
	$\mathcal{C}_{\text{CFG}}$	62%
SEM-CTRL	$\mathcal{C}_{\text{SEM}}$	**98%**

Blocksworld ablation on Llama 70B

- $\mathcal{C}_{\text{CFG}}$ alone: minimal benefit without search
- $\mathcal{C}_{\text{SEM}}$ provides the largest gains across all configurations

How does each component contribute?

Alg.	$\mathcal{C}$	A
Base	—	30%
	$\mathcal{C}_{\text{CFG}}$	30%
	$\mathcal{C}_{\text{SEM}}$	40%
BoN	—	66%
	$\mathcal{C}_{\text{CFG}}$	66%
	$\mathcal{C}_{\text{SEM}}$	90%
MCTS	—	46%
	$\mathcal{C}_{\text{CFG}}$	62%
SEM-CTRL	$\mathcal{C}_{\text{SEM}}$	**98%**

Blocksworld ablation on Llama 70B

- $\mathcal{C}_{\text{CFG}}$ alone: minimal benefit without search
- $\mathcal{C}_{\text{SEM}}$ provides the largest gains across all configurations
- BoN + $\mathcal{C}_{\text{SEM}}$: 70B jumps from 66% $\rightarrow$ **90%**
- MCTS + $\mathcal{C}_{\text{CFG}}$ still underperforms BoN + $\mathcal{C}_{\text{SEM}}$

How does each component contribute?

Alg.	$\mathcal{C}$	A
Base	—	30%
	$\mathcal{C}_{\text{CFG}}$	30%
	$\mathcal{C}_{\text{SEM}}$	40%
BoN	—	66%
	$\mathcal{C}_{\text{CFG}}$	66%
	$\mathcal{C}_{\text{SEM}}$	90%
MCTS	—	46%
	$\mathcal{C}_{\text{CFG}}$	62%
SEM-CTRL	$\mathcal{C}_{\text{SEM}}$	**98%**

Blocksworld ablation on Llama 70B

- $\mathcal{C}_{\text{CFG}}$ alone: minimal benefit without search
- $\mathcal{C}_{\text{SEM}}$ provides the largest gains across all configurations
- BoN + $\mathcal{C}_{\text{SEM}}$: 70B jumps from 66% $\rightarrow$ **90%**
- MCTS + $\mathcal{C}_{\text{CFG}}$ still underperforms BoN + $\mathcal{C}_{\text{SEM}}$

Key insight

Semantic control provides crucial structure for MCTS — improvements **exceed the sum** of individual contributions.

How efficient is SEM-CTRL compared to reasoning models?

Task	Algorithm	N_{tokens}
SGS	o1-preview	1,639
	DeepSeek-R1	927
	o4-mini	705
	SEM-CTRL	**250**
CR	o1-preview	3,185
	DeepSeek-R1	3,017
	o4-mini	1,191
	SEM-CTRL	**123**
Blocks	o1-preview	2,458
	DeepSeek-R1	2,346
	o4-mini	1,545
	SEM-CTRL	**589**

Avg tokens prer sample, Llama 70B

How efficient is SEM-CTRL compared to reasoning models?

Task	Algorithm	N_{tokens}
SGS	o1-preview	1,639
	DeepSeek-R1	927
	o4-mini	705
	SEM-CTRL	**250**
CR	o1-preview	3,185
	DeepSeek-R1	3,017
	o4-mini	1,191
	SEM-CTRL	**123**
Blocks	o1-preview	2,458
	DeepSeek-R1	2,346
	o4-mini	1,545
	SEM-CTRL	**589**

Avg tokens prer sample, Llama 70B

SEM-CTRL reduces token usage by up to an **order of magnitude**:

- CR: **25.8**× fewer tokens than o1-preview
- **24.5**× fewer than DeepSeek-R1
- **9.7**× fewer than o4-mini

How efficient is SEM-CTRL compared to reasoning models?

Task	Algorithm	N_{tokens}
SGS	o1-preview	1,639
	DeepSeek-R1	927
	o4-mini	705
	SEM-CTRL	**250**
CR	o1-preview	3,185
	DeepSeek-R1	3,017
	o4-mini	1,191
	SEM-CTRL	**123**
Blocks	o1-preview	2,458
	DeepSeek-R1	2,346
	o4-mini	1,545
	SEM-CTRL	**589**

Avg tokens prer sample, Llama 70B

SEM-CTRL reduces token usage by up to an **order of magnitude**:

- CR: **25.8**× fewer tokens than o1-preview
- **24.5**× fewer than DeepSeek-R1
- **9.7**× fewer than o4-mini

This efficiency stems from semantic pruning: **1–15 valid tokens per step** vs. 128k+ in unconstrained search and unconstrained reasoning tokens.

How well does SEM-CTRL work without semantics and rewards?

Alg.	Model	V_{CFG}
Base	Llama 1B	75.0%
	Llama 70B	96.9%
Xgrammar	Llama 1B	**100.0%**
	Llama 70B	**100.0%**
API	DeepSeek-R1	98.4%
	o4-mini	98.4%
SEM-CTRL	Llama 1B	**100.0%**
	Llama 70B	**100.0%**

JSON parsing task results

Here SEM-CTRL operates **without MCTS** and uses an ASG encoding only a CFG: demonstrating broader applicability to semi-structured generation.

How well does SEM-CTRL work without semantics and rewards?

Alg.	Model	V_{CFG}
Base	Llama 1B	75.0%
	Llama 70B	96.9%
Xgrammar	Llama 1B	**100.0%**
	Llama 70B	**100.0%**
API	DeepSeek-R1	98.4%
	o4-mini	98.4%
SEM-CTRL	Llama 1B	**100.0%**
	Llama 70B	**100.0%**

JSON parsing task results

Here SEM-CTRL operates **without MCTS** and uses an ASG encoding only a CFG: demonstrating broader applicability to semi-structured generation.

- Even Llama 70B (96.9%) and reasoning models (98.4%) fail to guarantee perfect syntactic validity

How well does SEM-CTRL work without semantics and rewards?

Alg.	Model	V_{CFG}
Base	Llama 1B	75.0%
	Llama 70B	96.9%
Xgrammar	Llama 1B	**100.0%**
	Llama 70B	**100.0%**
API	DeepSeek-R1	98.4%
	o4-mini	98.4%
SEM-CTRL	Llama 1B	**100.0%**
	Llama 70B	**100.0%**

JSON parsing task results

Here SEM-CTRL operates **without MCTS** and uses an ASG encoding only a CFG: demonstrating broader applicability to semi-structured generation.

- Even Llama 70B (96.9%) and reasoning models (98.4%) fail to guarantee perfect syntactic validity

Confirms SEM-CTRL's reliable constraint enforcement across the full spectrum — from basic syntactic to complex semantic tasks.

Does fine-tuning complement SEM-CTRL?

Model	FT	A	Seq.
1B (Greedy)	0%	0%	–
	100%	14%	–
1B (SEM-CTRL)	0%	76%	71.1
	20%	**90%**	30.0
	100%	88%	**13.2**
70B (SEM-CTRL)	0%	**98%**	19.1

Blocksworld ablation dataset. Seq. = avg sequences explored.

Does fine-tuning complement SEM-CTRL?

Model	FT	A	Seq.
1B (Greedy)	0%	0%	–
	100%	14%	–
1B (SEM-CTRL)	0%	76%	71.1
	20%	**90%**	30.0
	100%	88%	**13.2**
70B (SEM-CTRL)	0%	**98%**	19.1

Blocksworld ablation dataset. Seq. = avg sequences explored.

1. SEM-CTRL without FT (**76%**) far exceeds best greedy+FT (14%)

Does fine-tuning complement SEM-CTRL?

Model	FT	A	Seq.
1B (Greedy)	0%	0%	–
	100%	14%	–
1B (SEM-CTRL)	0%	76%	71.1
	20%	**90%**	30.0
	100%	88%	**13.2**
70B (SEM-CTRL)	0%	**98%**	19.1

Blocksworld ablation dataset. Seq. = avg sequences explored.

1. SEM-CTRL without FT (**76%**) far exceeds best greedy+FT (14%)
2. FT and SEM-CTRL are **complementary**: accuracy $\rightarrow$ 90% with just 20% data; search efficiency improves **5.4**$\times$

Does fine-tuning complement SEM-CTRL?

Model	FT	A	Seq.
1B (Greedy)	0%	0%	–
	100%	14%	–
1B (SEM-CTRL)	0%	76%	71.1
	20%	**90%**	30.0
	100%	88%	**13.2**
70B (SEM-CTRL)	0%	**98%**	19.1

Blocksworld ablation dataset. Seq. = avg sequences explored.

1. SEM-CTRL without FT (**76%**) far exceeds best greedy+FT (14%)
2. FT and SEM-CTRL are **complementary**: accuracy $\rightarrow$ 90% with just 20% data; search efficiency improves **5.4**$\times$
3. 1B + SEM-CTRL + moderate FT **rivals 70B** while being 1.4$\times$ more sample efficient

Does fine-tuning complement SEM-CTRL?

Model	FT	A	Seq.
1B (Greedy)	0%	0%	–
	100%	14%	–
1B (SEM-CTRL)	0%	76%	71.1
	20%	**90%**	30.0
	100%	88%	**13.2**
70B (SEM-CTRL)	0%	**98%**	19.1

Blocksworld ablation dataset. Seq. = avg sequences explored.

1. SEM-CTRL without FT (**76%**) far exceeds best greedy+FT (14%)
2. FT and SEM-CTRL are **complementary**: accuracy $\rightarrow$ 90% with just 20% data; search efficiency improves **5.4**$\times$
3. 1B + SEM-CTRL + moderate FT **rivals 70B** while being 1.4$\times$ more sample efficient

SEM-CTRL significantly **reduces the amount of fine-tuning** required to reach a target performance level — robust when training is prohibitive, synergistic when FT is feasible.

Empirical Takeaways

1. **Guaranteed Validity:** SEM-CTRL achieves **100% semantic validity** across all tasks and model sizes — something no baseline or reasoning model can ensure

Empirical Takeaways

1. **Guaranteed Validity:** SEM-CTRL achieves **100% semantic validity** across all tasks and model sizes — something no baseline or reasoning model can ensure
2. **Parameter Efficiency:** Llama 1B with SEM-CTRL consistently outperforms greedy/BoN with Llama 70B, and matches or exceeds SOTA reasoning models on most tasks

Empirical Takeaways

1. **Guaranteed Validity:** SEM-CTRL achieves **100% semantic validity** across all tasks and model sizes — something no baseline or reasoning model can ensure
2. **Parameter Efficiency:** Llama 1B with SEM-CTRL consistently outperforms greedy/BoN with Llama 70B, and matches or exceeds SOTA reasoning models on most tasks
3. **Token Efficiency:** Up to **25.8**$\times$ fewer tokens than reasoning models, thanks to semantic pruning reducing branching factor to 1–15

Empirical Takeaways

1. **Guaranteed Validity:** SEM-CTRL achieves **100% semantic validity** across all tasks and model sizes — something no baseline or reasoning model can ensure
2. **Parameter Efficiency:** Llama 1B with SEM-CTRL consistently outperforms greedy/BoN with Llama 70B, and matches or exceeds SOTA reasoning models on most tasks
3. **Token Efficiency:** Up to **25.8**$\times$ fewer tokens than reasoning models, thanks to semantic pruning reducing branching factor to 1–15
4. **Complementary to Fine-Tuning:** Significantly reduces the amount of FT data needed; improves both accuracy and search efficiency when combined

Conclusion

Summary of Contributions

1. **Domain-independent framework** using ASGs for a comprehensive hierarchy of token-aligned constraints (syntactic $\rightarrow$ context-sensitive $\rightarrow$ semantic)

Summary of Contributions

1. **Domain-independent framework** using ASGs for a comprehensive hierarchy of token-aligned constraints (syntactic $\rightarrow$ context-sensitive $\rightarrow$ semantic)
2. **Efficient token-level MCTS** exploring only semantically valid trajectories (branching factor 1–15)

Summary of Contributions

1. **Domain-independent framework** using ASGs for a comprehensive hierarchy of token-aligned constraints (syntactic $\rightarrow$ context-sensitive $\rightarrow$ semantic)
2. **Efficient token-level MCTS** exploring only semantically valid trajectories (branching factor 1–15)
3. **Empirical results** across SGS, CR, JSON parsing, and Planning:
   - Small LLMs (1B) outperform larger models and SOTA reasoning models
   - 100% semantic validity guaranteed; up to 25.8$\times$ token efficiency

Summary of Contributions

1. **Domain-independent framework** using ASGs for a comprehensive hierarchy of token-aligned constraints (syntactic $\rightarrow$ context-sensitive $\rightarrow$ semantic)
2. **Efficient token-level MCTS** exploring only semantically valid trajectories (branching factor 1–15)
3. **Empirical results** across SGS, CR, JSON parsing, and Planning:
   - Small LLMs (1B) outperform larger models and SOTA reasoning models
   - 100% semantic validity guaranteed; up to 25.8$\times$ token efficiency
4. **Versatile inference-time mechanism:** Strong standalone performance without fine-tuning, and synergistic gains when combined with training — significantly reducing FT data requirements

Summary of Contributions

1. **Domain-independent framework** using ASGs for a comprehensive hierarchy of token-aligned constraints (syntactic $\rightarrow$ context-sensitive $\rightarrow$ semantic)
2. **Efficient token-level MCTS** exploring only semantically valid trajectories (branching factor 1–15)
3. **Empirical results** across SGS, CR, JSON parsing, and Planning:
   - Small LLMs (1B) outperform larger models and SOTA reasoning models
   - 100% semantic validity guaranteed; up to 25.8$\times$ token efficiency
4. **Versatile inference-time mechanism:** Strong standalone performance without fine-tuning, and synergistic gains when combined with training — significantly reducing FT data requirements

General pattern

`SEM-CTRL` transforms general-purpose off-the-shelf LLMs into domain-specialized models at inference time, with guaranteed semantic validity and without fine-tuning.

Thank you!

Published in TMLR (03/2026)

OpenReview: openreview.net/forum?id=ICUHKhOISN

m.albinhassan23@imperial.ac.uk