Thinking Before Constraining: A Unified Decoding Framework for Large Language Models

Anonymous ACL submission

Abstract

Natural generation allows Large Language Models (LLMs) to produce free-form responses with rich reasoning, yet the lack of structure makes outputs difficult to verify. Conversely, constrained decoding ensures standardized formats but can inadvertently restrict reasoning capabilities by imposing constraints too early in the generation process. We propose a hybrid approach, namely *In-Writing*, that combines free-form reasoning and structured generation in a single call. The model first performs unconstrained reasoning and only applies structured decoding after a trigger token is generated, explicitly decoupling reasoning from formatting. We establish that our trigger-token strategies are able to virtually eradicate *premature triggering*, a failure mode in which constrained decoding interrupts ongoing reasoning. Evaluations across diverse datasets covering classification and reasoning tasks demonstrate that our approach outperforms the state-of-the-art by achieving accuracy gains of up to 27% over natural generation. Our code and results are available online[1].

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across a wide range of applications, including text completion, summarization, question answering, code generation, web navigation, data extraction, and tool use (Zhao et al., 2023; Minaee et al., 2024). Trained primarily on large natural language corpora, LLMs typically generate fluent and flexible text at inference time. As a consequence, they do not inherently guarantee adherence to predefined output structures. Although advanced LLMs often produce syntactically well-formed outputs, this behavior is not assured (Koo et al., 2024). The absence of strict structural guarantees can limit the applicability of LLMs in tasks such as schema-based information extraction, structured question answering, and many industrial use cases (Liu et al., 2024).

To address structured output requirements, grammar-constrained decoding methods (Willard and Louf, 2023; guidanceAI, 2025; Dong et al., 2024) restrict token generation by masking invalid continuations, ensuring syntactic correctness. However, these approaches may reduce expressiveness and fluency, harming generalization (Tam et al., 2024; Banerjee et al., 2025; Lee et al., 2026).

A common alternative, **NL-to-Format**, employs a two-stage pipeline where a model first generates Natural Language (NL) and a second model converts it into the target format (Tam et al., 2024; Lee et al., 2026). This approach improves flexibility but increases computational cost and redundancy, without guaranteeing adherence to predefined output structures (Lundberg and Ribeiro, 2023; Lee et al., 2026).

Another approach, **CRANE**, alternates between free-form and constrained generation using delimiter-based (« ») switching, where reasoning is enclosed within delimiters (Banerjee et al., 2025). While improving parsability, CRANE relies on complex prompting and is typically limited to single-field formats, and reintroduces concerns about reasoning under constrained decoding.

To bridge this gap, we introduce **In-Writing**, a framework that unifies natural and structured generation by allowing reasoning to proceed in free form until specific trigger tokens (e.g., `<eos>` or `{`) activate structured decoding. Unlike prior methods that tightly couple reasoning with constrained regions (e.g., CRANE) or rely on rigid pipelines (e.g., NL-to-Format), our approach decouples reasoning from formatting. We further study trigger-token strategies to mitigate *premature triggering*, a failure mode in which constrained decoding interrupts ongoing reasoning.

We evaluate our framework across diverse down-

[1]The code is appended as supplemental material. The repository will be made available upon acceptance.

Question: Alice, Bob, Claire, Dave, and Eve are dancers at a square dance. At the start of a song, they each have a partner: Alice is dancing with Helga, Bob is dancing with Ophelia, Claire is dancing with Sam, Dave is dancing with Melissa, and Eve is dancing with Lola. Throughout the song, the dancers often trade partners. First, Eve and Claire switch partners. Then, Bob and Eve switch partners. Then, Dave and Bob switch partners. Then, Alice and Bob switch partners. Finally, Alice and Claire switch partners. At the end of the dance, Bob is dancing with:
A) Helga. B) Ophelia. C) Sam. D) Melissa. E) Lola.
Answer: A

Natural Generation:
Let's break down the sequence of partner switches step by step:
1. Eve and Claire switch partners: Eve is now dancing with Sam, and Claire is dancing with Lola.
2. Bob and Eve switch partners: Bob is now dancing with Lola, and Eve is dancing with Melissa.
3. Dave and Bob switch partners: Dave is now dancing with Lola, and Bob is dancing with Melissa.
4. Alice and Bob switch partners: Alice is now dancing with Melissa, and Bob is dancing with Helga.
5. Alice and Claire switch partners: Alice is now dancing with Lola, and Claire is dancing with Helga.
At the end of the dance, Bob is dancing with... Helga!
✗

In-Writing:
Let's break down the sequence of partner switches step by step:
1. Eve and Claire switch partners: Eve is now dancing with Sam, and Claire is dancing with Lola.
2. Bob and Eve switch partners: Bob is now dancing with Lola, and Eve is dancing with Melissa.
3. Dave and Bob switch partners: Dave is now dancing with Lola, and Bob is dancing with Melissa.
4. Alice and Bob switch partners: Alice is now dancing with Melissa, and Bob is dancing with Helga.
5. Alice and Claire switch partners: Alice is now dancing with Lola, and Claire is dancing with Helga.
At the end of the dance, Bob is dancing with... Helga!
{ "final_answer": "A" }
✓

Figure 1: Comparison between Natural Generation (NL) and the proposed In-Writing method on Llama3-8B-it. Both produce identical reasoning; In-Writing additionally generates an easily parsable final answer with syntactic corrections (e.g., mapping "Helga" to choice **A**.).

stream tasks and model families, including Qwen, Llama, Gemma, DeepSeek and SmolLM models (1.5B–14B parameters). Our evaluation spans classification and reasoning benchmarks, using accuracy, parsability, and token efficiency as metrics to demonstrate effectiveness.

Our main contributions are: (1) a framework that combines natural and structured generation with minimal token overhead; (2) evidence that constrained decoding serves as a more effective parser and corrector than using separate models to extract final answers (see Figure 1); (3) an investigation of trigger-token strategies to mitigate the premature triggering problem; and (4) an empirical demonstration that forcing models to reason entirely within a constrained grammar space is less effective than our framework.

This paper is structured as follows: Section 2 introduces preliminary work on Finite Automata. Section 3 reviews related work while Sections 4, 5, and 6 present our method and experimental results respectively, demonstrating its effectiveness and robustness.

2 Preliminaries

We first introduce the formal definition of a finite automaton, also known as a Finite-State Machine (FSM) (Sipser, 1996), which is one of the central components in constrained decoding (Willard and Louf, 2023; Koo et al., 2024), as they can represent a target regular expression (or regex)[2] and guide the model to select only tokens that are consistent with that regex at each decoding step.

Definition 1 (Finite Automaton)**.** *A finite automaton, or finite-state machine is a 5-tuple* $(Q, \Sigma, \delta, q_0, F)$*, where* Q *is a finite set of states,* Σ *a finite alphabet,* $\delta : Q \times \Sigma \to Q$ *the transition function,* $q_0 \in Q$ *the start state, and* $F \subseteq Q$ *the set of accept states.*

3 Related Work

Token generation in language models follows an autoregressive process, where each token is sampled from the conditional distribution given previous tokens (Bengio et al., 2003). However, standard sampling is stochastic and does not enforce structural constraints, limiting its applicability in settings requiring strict output formats (Scholak et al., 2021; Geng et al., 2023; Tam et al., 2024). Tool

[2]We use the mathematical definition of regular expression (Sipser, 1996, Def. 1.52, p. 64).

calling enables LLMs to invoke external tools and APIs (Schick et al., 2023; Yao et al., 2022), and has also been used for structured output generation (e.g., JSON) (Liu and contributors, 2023). However, since tool calls are generated autoregressively, strict structural correctness is not guaranteed.

Hard constrained decoding is yet another technique which enforces structured generation by applying decoder logit masking during sampling (Zhang et al., 2019; Deutsch et al., 2019), as shown in Algorithm 1. Recent work formulates regex- or grammar-guided generation as a finite-state machine (FSM) (Definition 1), enabling controllable initialization and termination (Willard and Louf, 2023; Koo et al., 2024). However, hard constrained decoding has been shown to degrade LLM performance on certain tasks (Tam et al., 2024; Banerjee et al., 2025; Lee et al., 2026).

Algorithm 1: LM Token Sampling with Masking

```
sample_tokens_masked(rx, p, L) ;
 // rx: regex or grammar, p:
 prompt, L: max new tokens
s ← (p) ; // Generated token sequence
for i ← 1 to L do
    α ← LM(s);
    Construct mask m(s, rx);
    α̃ ← m ⊙ α;
    Sample t̃ from α̃ ;
    if t̃ = EOS then
        break;
    end
    s ← append(s, t̃);
end
return s;
```

A few recent works have introduced hybrid approaches such as **NL-to-Format** and **CRANE**. **NL-to-Format** uses a two-stage pipeline in which a model first generates an answer in natural language and a second model converts it into the target format (Tam et al., 2024; Lee et al., 2026). **CRANE** interleaves free-form and grammar-constrained generation via delimiter-based (« ») switching (Banerjee et al., 2025). However, requiring constrained fields to remain consistent across invocations can hinder multi-field or complex extraction with heterogeneous constraints. Moreover, the final response may not be reliably extractable, as there is no guarantee it will be generated within « » beyond prompting. This also reintroduces concerns regarding reasoning under constrained decoding.

Our work addresses the gap between reasoning expressiveness (limitation of hard-constrained decoding and CRANE, which restrict free-form reasoning), format guarantees (limitation of natural generation and NL-to-Format, which lack strict structural enforcement), and computational overhead, by proposing a unified hybrid constrained decoding framework that preserves reasoning flexibility while ensuring strict format compliance with minimal overhead.

4 In-Writing Method

In this section, we introduce **In-Writing**, a decoding framework that separates reasoning from structural formatting constraints. We first formalize our method in Section 4.1, followed by its operational decoding algorithm in Section 4.2.

4.1 Decoupled Probabilistic Formulation

Given a question q, Chain-of-Thought (CoT) generation models the answer A as being produced via latent reasoning traces R (Wei et al., 2022) which is formulated as:

$$P(A \mid q) = \sum_R P(A \mid R, q) P(R \mid q). \quad (1)$$

When a formatting constraint F is imposed via logit masking, conventional constrained decoding conditions both reasoning and answer generation on F as follows:

$$P(A \mid q, F) = \sum_R P(A \mid R, q, F) P(R \mid q, F). \quad (2)$$

Under hard-constrained decoding, invalid reasoning paths are removed from the search space, as formalized in the following equation:

$$P(R \mid q, F) = \frac{1}{Z} P(R \mid q)\, \mathbf{1}_{R \in F}, \quad (3)$$

where $\mathbf{1}_{R \in F}$ is an indicator function enforcing compliance with F (1 if $R \in F$, 0 otherwise), and Z is a normalization. Consequently, logically valid reasoning traces may be discarded due to minor structural violations.

To address this limitation, In-Writing delays formatting until reasoning is complete, effectively decoupling reasoning from formatting constraints and making it independent of the formatting condition, as formalized below:

$$P(R \mid q, F) = P(R \mid q). \quad (4)$$

This independence enables reformulating Equation 2 as follows:

$$P(A \mid q, F) = \sum_{R} P(A \mid R, q, F) P(R \mid q). \quad (5)$$

Equation 5 shows that In-Writing retains both the reasoning expressiveness of CoT decoding (Equation 1) and the formatting guarantees of hard-constrained decoding (Equation 2).

4.2 State-Based Decoding Algorithm

In-Writing is instantiated in Algorithm 2, where LLMs first generate an unconstrained reasoning trace (state -1; lines 1–6) until a trigger token is emitted, then switch to structured generation (state 0; lines 8–16). In this way, constrained decoding is applied only after reasoning is completed, effectively acting as a parser that converts the final reasoning trace into a structured output. An illustration is given in Section A.1.

Algorithm 2: LM Token Sampling with In-Writing Method

```
sample_InWriting(rx, triggers, p, L) ;
 // rx: regex or grammar, triggers:
 trigger tokens, p: prompt, L: max
 new tokens
1  s ← (p) ; // Generated token sequence
2  for i ← 1 to L do
3      α ← LM(s);
4      Sample t from α;
5      if t ∉ triggers then
6          s ← append(s, t);
7      end
8      else
9          while rx is not completed do
10             Construct mask m(s, rx);
11             α̃ ← m ⊙ α;
12             Sample t̃ from α̃;
13             s ← append(s, t̃);
14         end
15         return s;
16     end
17 end
18 return s;
```

This approach offers four key advantages: **(1) Unified pipeline:** it decouples reasoning and formatting within a single call, eliminating external verification. **(2) Guaranteed syntactic validity:** regex- or grammar-based constraints enforce schema-compliant outputs during decoding. **(3) Robustness:** outputs are reliably mapped to structured formats without complex prompting or implementation. **(4) Minimal overhead:** the formatting step introduces negligible latency.

We term this approach *In-Writing*, drawing an analogy to diffusion-based *inpainting* (Lugmayr et al., 2022), where generation is restricted to masked regions. Similarly, *In-Writing* separates reasoning from formatting by allowing free-form generation while enforcing strict syntactic constraints only on target output slots.

5 Experimental Setup

This section describes the experimental setup used to evaluate our approach. All experiments were conducted on NVIDIA A40 GPU.

5.1 Datasets

Following the evaluation settings from (Tam et al., 2024; Banerjee et al., 2025), we evaluate our hybrid approach on a diverse set of reasoning and classification benchmarks spanning numerical, symbolic, and textual outputs, using the same preprocessing and data splits as prior works.

5.1.1 Reasoning Tasks

We use the following datasets for the reasoning tasks.

GSM8K (Cobbe et al., 2021) contains grade-school math problems which require multi-step reasoning.

GSM-Symbolic (Mirzadeh et al., 2025) is a symbolic variant of GSM8K replacing numbers with variables to test compositional and algebraic generalization.

Last Letter Concatenation (Wei et al., 2022) is a symbolic task where the model concatenates the last letters of words.

Shuffled Objects (Ghazal et al., 2013) is a Big-Bench task requiring prediction of object arrangements after sequential shuffling operations.

5.1.2 Classification Tasks

We use the following datasets for evaluating over the classification tasks.

DDXPlus (Fansi Tchango et al., 2022) is a 49-class medical diagnosis task (Wu et al., 2024).

MultiFin (Jørgensen et al., 2023) is a 5-class financial text classification task.

Sports Understanding (Ghazal et al., 2013) is a BigBench task classifying sports-related sentences as plausible or implausible.

NI - Task 280 (Mishra et al., 2022) is a multiple-choice stereotype classification task based on a given paragraph, which is highly sensitive to the format of the prompt (Sclar et al., 2023).

5.2 Evaluation Metrics

We evaluate along two dimensions: correctness and parse success. Accordingly, we report **accuracy** and **parse rate** as the primary metrics.

Accuracy Metrics. For the Natural Language (NL) baseline, answers are extracted from free-form outputs using predefined prefixes (e.g., "answer is:") specified in the prompt.

For NL-to-Format, a larger LLM extracts the final answer from the first-stage NL output. For Constrained Decoding and In-Writing, answers are parsed directly from the structured output. All extracted answers are evaluated against the ground-truth answer using exact string match.

Parse Metrics. For the NL baseline, parsability is defined by the presence of the expected answer prefix (e.g., "answer is:"), followed by a syntactically valid extracted output.

For NL-to-Format, constrained decoding, and In-Writing, parsability is defined by whether the generated output is syntactically valid.

In-Writing Robustness Evaluation. In-Writing separates reasoning and formatting via trigger tokens, making it sensitive to trigger selection. We evaluate robustness across different trigger token sets. We denote **In-Writing-Base** as using `trigger_token_ids = {<eos>, {}`, and **In-Writing*** as using `<eos>` as the sole trigger token. This setup enables analysis of *premature triggering*, where early constraint activation truncates reasoning, and whether using a single `<eos>` trigger mitigates this issue.

Since both In-Writing and NL-to-Format extend the NL baseline, we additionally perform an **overlap analysis** to compare agreements and discrepancies across methods in Section 6.1.3.

5.3 Models and Prompts

We evaluate 18 open-source models released between 2024 and 2026 from five families: Qwen, Llama, Gemma, DeepSeek, and SmolLM. Model sizes range from 1.5B to 32B parameters, all run locally using the *Transformers* library.

NL-to-Format Comparison For direct comparability with Tam et al. (2024), we adopt their prompt templates and baseline models: LLaMA3-8B-Instruct (Dubey et al., 2024) and Gemma2-9B-Instruct (Team et al., 2024). We extend the evaluation with newer models, including Qwen3 (1.7B, 4B, 8B) (Yang et al., 2025), Qwen3.5 (2B, 4B, 9B) (Qwen Team, 2026), and SmolLM3-3B (Bakouch et al., 2025).

CRANE Comparison Following Banerjee et al. (2025), we use their prompt templates with a small additional instruction (see Appendix A.3.), across open-source models: Qwen2.5 (1.5B, Coder-7B, Math-7B, Coder-14B) (Qwen et al., 2025), LLaMA3.1-8B-Instruct (Dubey et al., 2024), and DeepSeek-R1 Distill variants (7B/14B Qwen, 8B LLaMA) (Guo et al., 2025).

We retain the original benchmark templates to evaluate model performance under fixed-format prompts. The NL-to-Format benchmark requires natural-language prefixes (e.g., "answer is:"), whereas CRANE enforces delimiter-based outputs (e.g., « »). These templates place our approach at a disadvantage by providing no guidance on the In-Writing output format. This setup tests whether models can map internal reasoning to required output formats without additional prompt engineering.

The prompt template is provided in Appendix A.2.

5.4 Output Schema

Although regex or grammar constraints can be specified independently, we use Pydantic[3] to automatically define and customize JSON schemas. For vanilla constrained decoding, outputs are strictly formatted as JSON with two keys: `think_step_by_step` for reasoning and `final_answer` for the answer, whereas In-Writing first generates free-form reasoning before emitting the final JSON, from which the answer is extracted via the `final_answer` field. While our approach is not limited to JSON, we focus on JSON schemas due to their simplicity and ease of conversion to formats such as XML or YAML.

5.5 Experimental Implementation Details

NL-to-Format Comparison. We report Baseline (NL) results by extracting the natural-language an-

[3] https://docs.pydantic.dev/latest/

swer from LLMs using the **"answer is:"** prefix, as specified in the prompt template.

For NL-to-Format results, the complete first-stage output (i.e., Baseline (NL)) is provided as input to a larger second-stage parser model. For reproducibility, we use the open-source Qwen3-32B instead of *claude-3-haiku-20240307* used in prior work (Tam et al., 2024).

Vanilla constrained decoding (**Constrained**) and In-Writing are implemented using Litelines (Silva, 2025), an open-source library built on top of Outlines (Willard and Louf, 2023). Litelines extends Outlines with `allow_preamble` and `trigger_token_ids`, enabling the model to reason freely outside the schema until a trigger token is generated.

CRANE Comparison. We evaluate only the In-Writing, as Baseline, Constrained, and Chain-of-Thought results are already reported by Banerjee et al. (2025).

6 Results and Discussion

In this section, we evaluate In-Writing against state-of-the-art (SOTA) methods. We compare In-Writing with two approaches: (1) a two-stage natural language-to-format (NL-to-Format) pipeline (Section 6.1) and (2) a hybrid constrained decoding approach (CRANE) (Section 6.2).

6.1 NL-to-Format Comparison: Structured Generation as an Effective Parser

6.1.1 Overall Assessment

We first evaluate In-Writing-Base and In-Writing* on seven datasets (GSM8K, Last Letter Concatenation, Shuffled Objects, DDXPlus, MultiFin, Sports, and Task 280) using the same prompts, models (LLaMA3-8B-Instruct and Gemma2-9B-Instruct), and zero-shot setting as prior work (Tam et al., 2024); results are shown in Tables 1 and 5.

We further evaluate on newer models of varying sizes, including Qwen3 (1.7B, 4B, 8B), Qwen3.5 (2B, 4B, 9B), and SmolLM3-3B, on three reasoning tasks (GSM8K, Last Letter Concatenation, and Shuffled Objects), which have been reported to exhibit notable performance degradation under constrained decoding, using the same few-shot examples as in prior work (Tam et al., 2024). The results are shown in Tables 2, 3, and 4.

From Figure 2 and Tables 1, 2, 3, 4, 5, 6, 7 and 8, we make five observations:

Task	Model	Method	Acc (%)
GSM8K	LLaMA	Baseline (NL)	66.2 ± 9.5
		Constrained	*48.9* ± 6.7
		NL-to-Format	*74.7* ± 0.6
		In-Writing*	**77.9** ± 1.3
	Gemma	Baseline (NL)	82.7 ± 7.2
		Constrained	*84.2* ± 3.7
		NL-to-Format	*86.5* ± 0.6
		In-Writing*	**86.9** ± 0.4
Last Letter	LLaMA	Baseline (NL)	41.9 ± 3.8
		Constrained	*28* ± 12.2
		NL-to-Format	*70.1* ± 5.3
		In-Writing*	**70.3** ± 3.8
	Gemma	Baseline (NL)	9.3 ± 8.1
		Constrained	*39* ± 6.8
		NL-to-Format	*56.8* ± 9.8
		In-Writing*	**58.4** ± 4.8
Shuffled Obj	LLaMA	Baseline (NL)	1 ± 1.1
		Constrained	*15.7* ± 11
		NL-to-Format	*27* ± 5.5
		In-Writing*	**39.2** ± 4.7
	Gemma	Baseline (NL)	10 ± 1.7
		Constrained	*50.5* ± 8.9
		NL-to-Format	*49.4* ± 5.8
		In-Writing*	**50.5** ± 5.6

Table 1: Zero-shot results for GSM8K, Last Letter and ShuffledObj using LLaMA3-8B-it and Gemma2-9B-it, comparing Baseline (NL), Constrained decoding, NL-to-Format, and In-Writing*. Reported values are mean accuracy ± standard deviation over 9 variations. *Italicized* results are reported by Tam et al. (2024), while **bolded** results indicate the highest accuracy.

(1) Consistency across extraction methods. Baseline (NL), NL-to-Format, and In-Writing* share identical reasoning traces; performance differences stem solely from answer extraction. We find that parser-based extraction consistently outperforms regex-based methods, while In-Writing* yields further gains of up to 27% over LLM-based parsing.

(2) Parser dependence and limitations. NL-to-Format performance depends strongly on the parser (prompt or model) used in the second stage. While generally effective, parser quality varies across tasks and may fail (e.g., DDXPlus), leading to significant performance drops. See Appendix A.4 for details. It is also important to consider multi-field structured extraction, which is critical for industrial use cases and is not reliably followed by natural generation (Liu et al., 2024).

(3) Variance in constrained decoding. Vanilla constrained decoding exhibits high variance across tasks. Although larger and newer models reduce this gap, they still underperform other methods, consistent with prior work (Tam et al., 2024; Lee et al., 2026), leaving a substantial performance gap.

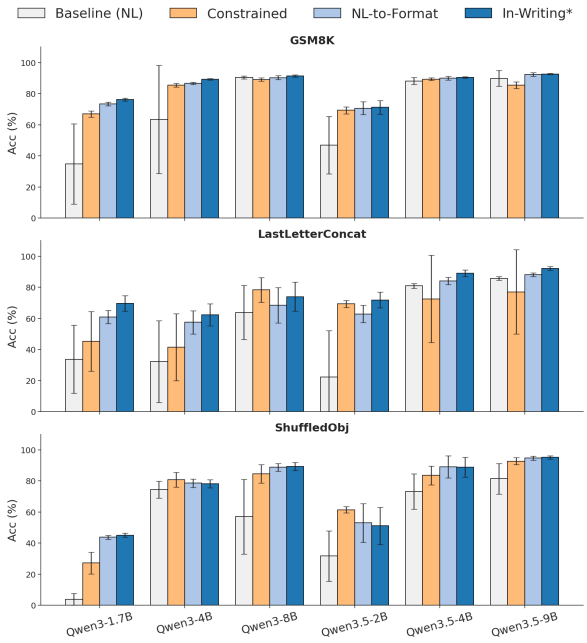

Figure 2: Zero-shot comparison for GSM8K, Last Letter, and ShuffledObj using various Qwen3 and Qwen3.5 model sizes, comparing Baseline (NL), Vanilla Constrained, NL-to-Format, and In-Writing*. In-Writing* performs best in nearly all settings.

In some cases, JSON-style constraints improve performance under identical prompts, but this gain is largely attributable to increased token generation.

(4) Premature triggering - a failure mode of constrained decoding. In-Writing-Base results indicate that premature triggering truncates reasoning and reduces output quality, especially on mathematically intensive tasks such as GSM8K (over 30% degradation compared to In-Writing*), whereas its impact on classification tasks is minimal.

(5) Effectiveness of In-Writing*. By simply setting <eos> as the unique trigger token, In-Writing* mitigates premature triggering - a failure mode of constrained decoding. In-Writing* generally outperforms NL-to-Format and vanilla constrained decoding across model scales, achieves 100% format validity without requiring larger models or explicit formatting guidance, and incurs only minimal overhead (5–20 tokens).

6.1.2 Parsibility Analysis

From Section A.4.1 and Table 6, we observe that prompting the desired output format is necessary but not sufficient for robust parsing, as the NL baseline exhibits substantial variation in parse rates across model sizes.

Vanilla constrained decoding and NL-to-Format improve parse rates; however, performance still varies notably depending on token budget or parser model and prompt design.

In contrast, In-Writing achieves 100% parse rate with low inference overhead, as only the formatting component requires constrained construction, while the reasoning segment remains unconstrained and is easily controlled via a maximum token limit. This makes the proposed framework computationally efficient while guaranteeing correct formatting.

6.1.3 Overlap analysis between NL-to-Format and In-Writing

With identical prompts, NL-to-Format and In-Writing* produce identical reasoning traces and differ only in the formatting step. This motivates analyzing the extraction capability of the two methods under the same reasoning process. We therefore introduce an overlap analysis with four metrics: joint success ($\cap$), where both methods correctly parse the answer; joint failure (Neither), where neither is correct; and two asymmetric cases, where only one method is correct.

From Figure 3 and Table 7, we observe that In-Writing* correctly parses many cases where NL-to-Format fails, whereas the reverse is much less common. It achieves up to an 11.8% improvement, driven by both stronger parsing and an inherent correction capability (Figure 1). This syntactic correction emerges from token masking induced by predefined constraints (Eq. 5). These results highlight In-Writing's effectiveness for accurate, format-compliant extraction and correction without additional models or post-hoc processing. Further analysis is provided in Appendix A.4.

6.1.4 Token Efficiency Analysis

As shown in Table 8, the number of newly generated tokens increases consistently from Baseline (NL) to NL-to-Format and In-Writing*. NL-to-Format adds 2–5 tokens, while In-Writing* adds 5–20 tokens. In contrast, vanilla constrained decoding is inconsistent, as it introduces JSON syntax tokens at the start of generation, thereby altering the reasoning trace.

There are two key differences between NL-to-Format and In-Writing*: (1) The parser in NL-to-Format only extracts the answer without enforcing structured format requirements (e.g., JSON or YAML), resulting in lower additional token cost; however, when format requirements are included,

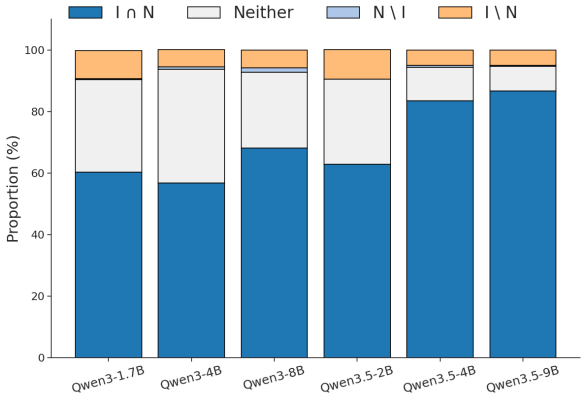

Figure 3: Overlap analysis on LastLetter between In-Writing* (**I**) and NL-to-Format (**N**), showing joint success (**I** ∩ **N**), joint failure (Neither), and asymmetric success cases (**N** \ **I**, where NL-to-Format is correct and In-Writing* is incorrect; **I** \ **N**, where In-Writing* is correct and NL-to-Format is incorrect). Values are reported in percentages. In–Writing* recovers most NL-to-Format failures, whereas the converse is far less common.

this cost becomes comparable to In-Writing*; (2) NL-to-Format relies on an additional LLM for parsing, which takes the parser prompt and the first model's output (NL) as input, leading to substantially higher input token usage and computational overhead.

Overall, an additional 5–20 tokens (which scales with the complexity of the desired grammar) is a reasonable trade-off for improved format correctness and performance.

6.2 CRANE Comparison: Limitations of Grammar-Constrained Decoding in LLM Reasoning

We evaluate In-Writing* on GSM-Symbolic using the same model families, and eight-shot setting as prior work (Banerjee et al., 2025). Unlike the previous reasoning tasks, GSM-Symbolic requires models to generate symbolic solution expressions. Since the provided reference solutions are not always universally valid, we manually evaluate model outputs against the ground-truth formulae. Results are shown in Table 9. Evaluation details and additional discussion are provided in Appendix A.3.

CRANE constrains reasoning to grammar-restricted fields (« »), which can limit symbolic reasoning capability. In particular, restricting the set of valid operators prevents models from generating necessary functions such as `round()`, while also encouraging incorrect use of permitted operators such as `int()`. In contrast, In-Writing* allows unrestricted reasoning while constraining only final answer extraction. Under nearly identical prompts, it consistently outperforms CoT due to improved parsing success. Overall, our approach yields performance improvements of up to 32% compared to CRANE.

7 Conclusion

We present In-Writing, a unified framework that decouples reasoning expressiveness from guaranteed output formatting within a single inference call, reframing constrained decoding not as a generation method but as a post-reasoning parser and corrector. Our experiments across a wide range of reasoning and classification benchmarks show that the proposed approach improves performance, even with misaligned prompts, while maintaining comparable or lower computational overhead. By simply designating `<eos>` as a unique trigger token, we show that *premature triggering* can be mitigated, preventing disruption of the reasoning trace.

This work systematically studies In-Writing under a deliberately suboptimal prompting setup to evaluate its robustness. Future work should explore prompt optimization strategies that could further enhance its performance and stability, making it an even more effective framework for structured decoding.

Limitations

A key limitation of this study is the lack of prompt optimization for In-Writing. We focus exclusively on a setting in which no explicit guidance is provided for the desired output format of In-Writing.

As shown in Table 2 and Table 3, increasing the number of few-shot examples improves Baseline performance but degrades both NL-to-Format and In-Writing. This effect is not attributable to differences in reasoning traces, which remain identical across methods, but rather to increased adherence to the **"answer is:"** prefix, which disproportionately benefits the baseline extraction mechanism. Table 7 further shows marginal cases where NL-to-Format succeeds while In-Writing fails. This is expected, as no prompting strategy is specifically tailored to In-Writing extraction. We anticipate that these limitations could be mitigated through more carefully designed prompts optimized for In-Writing.

References

Elie Bakouch, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Lewis Tunstall, Carlos Miguel Patiño, Edward Beeching, Aymeric Roucher, Aksel Joonas Reedi, Quentin Gallouédec, Kashif Rasul, Nathan Habib, Clémentine Fourrier, Hynek Kydlicek, Guilherme Penedo, Hugo Larcher, Mathieu Morlon, Vaibhav Srivastav, Joshua Lochner, and 4 others. 2025. SmolLM3: smol, multilingual, long-context reasoner. https://huggingface.co/blog/smollm3.

Debangshu Banerjee, Tarun Suresh, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. 2025. Crane: Reasoning with constrained llm generation. *arXiv preprint arXiv:2502.09061*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Daniel Deutsch, Shyam Upadhyay, and Dan Roth. 2019. A general-purpose algorithm for constrained sequential inference. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 482–492.

Yixin Dong, Charlie F Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. 2024. Xgrammar: Flexible and efficient structured generation engine for large language models. *arXiv preprint arXiv:2411.15100*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Arsene Fansi Tchango, Rishab Goel, Zhi Wen, Julien Martel, and Joumana Ghosn. 2022. Ddxplus: A new dataset for automatic medical diagnosis. *Advances in neural information processing systems*, 35:31306–31318.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. 2023. Grammar-constrained decoding for structured nlp tasks without finetuning. *arXiv preprint arXiv:2305.13971*.

Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. 2013. Bigbench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*, pages 1197–1208.

guidanceAI. 2025. guidance: A guidance language for controlling large language models. GitHub repository.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Rasmus Jørgensen, Oliver Brandt, Mareike Hartmann, Xiang Dai, Christian Igel, and Desmond Elliott. 2023. Multifin: A dataset for multilingual financial nlp. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 894–909.

Terry Koo, Frederick Liu, and Luheng He. 2024. Automata-based constraints for language model decoding. *arXiv preprint arXiv:2407.08103*.

Ivan Yee Lee, Loris D'Antoni, and Taylor Berg-Kirkpatrick. 2026. The format tax. *arXiv preprint arXiv:2604.03616*.

Jason Liu and contributors. 2023. Instructor: Structured outputs for large language models. https://github.com/instructor-ai/instructor. Accessed: 2025-12-06.

Michael Xieyang Liu, Frederick Liu, Alexander J Fiannaca, Terry Koo, Lucas Dixon, Michael Terry, and Carrie J Cai. 2024. " we need structured output": Towards user-centered constraints on large language model output. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pages 1–9.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. 2022. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11461–11471.

Scott Lundberg and Marco Tulio Ribeiro. 2023. The art of prompt design: Prompt boundaries and token healing. *Medium, may*.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196*.

Iman Mirzadeh, Keivan Alizadeh-Vahid, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2025. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *International Conference on Learning Representations*, volume 2025, pages 94743–94765.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the*

Association for Computational Linguistics (Volume 1: Long Papers), pages 3470–3487.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Qwen Team. 2026. Qwen3.5: Towards native multimodal agents.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2023. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. *arXiv preprint arXiv:2310.11324*.

Alonso Silva. 2025. Litelines. https://github.com/alonsosilvaallende/litelines. Accessed: 2025-12-07.

Michael Sipser. 1996. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29.

Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. 2024. Let me speak freely? a study on the impact of format restrictions on large language model performance. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1218–1236, Miami, Florida, US. Association for Computational Linguistics.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, and 1 others. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Brandon T Willard and Rémi Louf. 2023. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*.

Cheng-Kuang Wu, Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Vivian Chen, and Hung-yi Lee. 2024. Streambench: Towards benchmarking continuous improvement of language agents. *Advances in Neural Information Processing Systems*, 37:107039–107063.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Hao Zhang, Richard Sproat, Axel H Ng, Felix Stahlberg, Xiaochang Peng, Kyle Gorman, and Brian Roark. 2019. Neural models of text normalization for speech applications. *Computational Linguistics*, 45(2):293–337.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, and 1 others. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).

A Appendix

A.1 Illustration of the In-Writing Framework

A simple In-Writing illustration is shown in Figure 4, which demonstrates the process using a simple regex matching either "Yes." or "No.". In standard constrained decoding (Willard and Louf, 2023), generation starts in state 0 and is directly guided by an FSM that masks invalid tokens. In contrast, In-Writing performs free-form generation in state -1 and only switches to FSM-guided decoding after a trigger token, restricting outputs to regex- or grammar-constrained tokens. Tokenization may vary across models, but the underlying mechanism is unchanged.

A.2 NL-to-Format Comparison: Experimental Setting Discussion

A.2.1 Prompt Templates and Experimental Settings for NL-to-Format Comparison Experiments

For a direct comparison between the work of Tam et al. (2024) and our approach, we use the exact same prompt template and content. The base prompt template, in which all content is provided under the user role, is as follows:

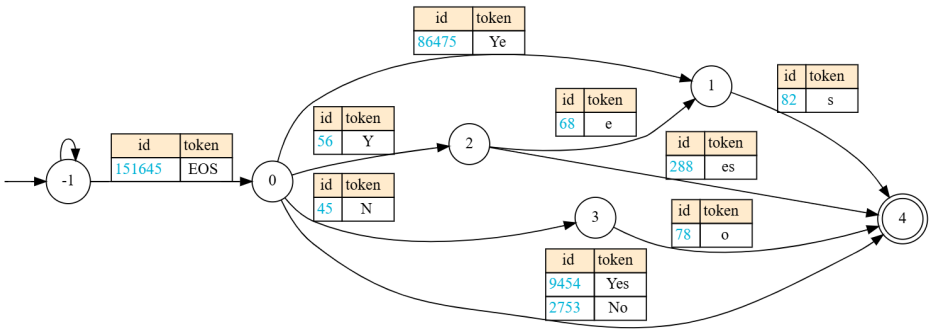

Figure 4: Illustrative example of the In-Writing approach. The model first generates unconstrained reasoning and then switches to guided decoding (state 0) once a trigger token is produced.

Follow the instruction to complete the task: `{task_description}`
Instruct: `{format_description}`
Few-shot examples: `{few_shots}`
Question: `{question}`

A.2.2 Prompt Inconsistencies in Previous Work

By investigating the outputs of the models provided in previous work (Tam et al., 2024), where constrained decoding was shown to harm LLM performance, we found several errors and inconsistencies. In the Shuffle Object Task Description Variations, task variation 1 and task variation 2 include seven answer choices, but the prompt incorrectly instructs the model to choose from only four choices:

Task Description Variation 1:

> In this task, you are tasked to answer the following commonsense knowledge task. Read carefully each of the last questions and think step by step before answering. Make sure the answer only contains one of these **four** choices: A, B, C, D, E, F, G.

Task Description Variation 2:

> Read carefully each of the last questions and think step by step before answering. Make sure the answer only contains one of these **four** choices: A, B, C, D, E, F, G. In this task, you are tasked to answer the following commonsense knowledge task.

Format variations are used to instruct LLMs to reason first and then provide answers in a specific format. These variations are intended to be task-independent and should therefore remain consistent across tasks. However, in the previous work (Tam et al., 2024), the format variations are sometimes kept unchanged and sometimes vary across tasks, which raises concerns about evaluation consistency.

> **Format Variations for GSM8K and Last Letter Concatenation**
> **Variation 1** Instruct: Provide your output in the following text format: Answer: <reasoning first>. The final answer is <answer>
> **Variation 2** Instruct: Provide your output in the following text format: Step by step reasoning: ... Answer: The final answer is ...
> **Variation 3** Instruct: Provide your output in the following text format: Answer: <think step by step>. The final answer is <answer>

> **Format Variations for Shuffled Objects**
> **Variation 1** Instruct: Now, take a deep breath and work on this problem step by step to derive the most likely choice. Provide your output in the following valid text format: Answer: ...reasoning here... The answer is ...
> **Variation 2** Instruct: Provide your output in the following text format: Step by step reasoning: ... Answer: The final answer is ...
> **Variation 3** Instruct: Now, take a deep breath and work on this problem step by step to derive the most likely answer. Provide your output in the following valid text format: Answer: [think step by step] The answer is [answer here]

We also identify some missing or uninformative few-shot examples in the work of Tam et al. (2024). For instance, in the Shuffled Object task, few-shot examples are not aligned with the target dataset. Examples include:

- **Question**: "Many people live in Ethiopia. The people are very thin and good at distance running."
  Response: `answer = "race"`
- **Question**: "The Norwegian man was boring."
  Response: `answer = "race"`

We therefore restrict the evaluation of Shuffled Objects to the zero-shot prompting setting.

We use the same experimental settings as prior work (Tam et al., 2024): `seed=1`, `temperature=0`, and `do_sample=False`, while default values are applied to all other unspecified parameters.

A.3 CRANE Comparison: Experimental Setting Discussion

A.3.1 Prompt Templates Used in CRANE Comparison Experiments

We use the same Chain-of-Thought prompt as in prior work (Banerjee et al., 2025), with a small instruction appended to the end of the prompt: *Then Output the final answer as a symbolic expression in JSON format exactly like this:* `"final_answer": <symbolic expression>`. Since the original prompt uses « » for reasoning, this encourages extraction of the final answer only. We keep the original few-shot examples unchanged.

A.3.2 GSM-Symbolic dataset

By inspecting the GSM-Symbolic dataset used in prior work, we identify several inconsistencies between question requirements and gold answers, including operator misuse (e.g., `//` vs. `/` or `int()`) and missing required operations such as `round()`. In many cases, the ground-truth solutions rely on `//` or `int()` to force integer-valued expressions. However, using floating-point division with `/` is often more mathematically sound and aligns better with how LLMs naturally generate solutions. Moreover, when rounding is required, `round()`—which rounds to the nearest integer—is generally more appropriate than truncation via `int()` (see Examples 1 and 51).

We also identify ambiguities in several questions where substituting symbolic variables with concrete values does not preserve a well-defined notion of correctness (see Example 55).

Example 1
Question: A fog bank rolls in from the ocean to cover a city. It takes {t} minutes to cover every {d} miles of the city. If the city is {y} miles across from the oceanfront to the opposite inland edge, how many minutes will it take for the fog bank to cover the whole city?
Answer: `y//d*t`
Example 51
Question: "{name} has a {product} stand at the {location}. He sells three kinds of {product}s: {item1}, {item2}, and {item3}. He usually sells {item1} for {cur}{p1} per {unit}, {item2} for {cur}{p2} per {unit}, and {item3} for {cur}{p3} per {unit}. {name} has no change today, so he has decided to round all his prices to the nearest dollar. If {name} sells {n1} {unit}s of {item1}, {n2} {unit}s of {item2}, and {n3} {unit}s of {item3}, how much will he make?
Answer: `n1*int(p1) + n2*int(p2) + n3*int(p3)`
Example 55
Question: {name} is learning to write and decides to keep re-writing the {alphabets} until she knows it. She writes it in full {n1} time(s), writes {frac} of it once, then rewrites everything she has already written. How many letters has {name} written in total?
Answer: `2 * ({alphabets} * {n1} + {alphabets} * {frac})`

A.3.3 Evaluation of GSM-Symbolic Experimental Results

Due to frequent operator misuse or omission in the ground-truth annotations (see A.3.2), we manually evaluate only the formulas parsed from LLM outputs against the gold answers. In this evaluation, we do not distinguish between semantically equivalent division operators (e.g., `/`, `//`, and `frac`), nor between rounding-related operations such as `int()` and `round()`.

In some cases, parsed formulas contain ambiguities between currency and numerical values (e.g., {cur}{p1}), where such expressions are interpreted as raw numerical inputs. We conservatively mark these cases as incorrect even when the overall structure is reasonable.

For example:

Gold Answer: `n1*int(p1) + n2*int(p2) + n3*int(p3)`

Parsed Answer: `n1*int({cur}{p1}) + n2*int({cur}{p2}) + n3*int({cur}{p3})`

A.4 Experimental Results and Discussion

A.4.1 Full Experimental Results

We first re-evaluate the experiments of Tam et al. (2024) using both the same prior models and additional extension models (Tables 2, 3, 4, and 5), along with parsability results (Table 6). We then further analyze the overlap between the NL-to-Format approach from prior work and our method (Table 7), followed by a token-efficiency analysis (Table 8).

We also compare In-Writing with CRANE—another hybrid constrained decoding framework—in Table 9.

A.4.2 Limitations of LLMs as Parsers

One limitation of using a larger LLM as a parser is its high computational cost and latency, which can be impractical (Tam et al., 2024; Lee et al., 2026). This raises the question of why a smaller model would be used at all if a larger model is available for parsing.

We also observe from the logs of Tam et al. (2024) that LLM-based parsers may **correct** model outputs rather than only parse them. In the LastLetter task, models often insert hyphens or whitespace between letters; removing these can incidentally turn an incorrect output into a correct one. In such cases, the parser effectively modifies the prediction rather than extracting it.

Finally, LLM-based parsing is prompt-dependent and can be unstable. In DDXPlus logs, parsers sometimes add conversational prefixes (e.g., "the final answer that matches the given list of valid diagnoses is") before outputting **acute dystonic reactions**, making the result harder to parse despite being correct.

A.4.3 In-Writing Is Not a Perfect Framework

As shown in Table 7, cases where In-Writing is correct but NL-to-Format is incorrect are substantially more frequent than the reverse, especially in the LastLetter task. This improvement comes not only from better parsing, but also from the auto-correcting effect of constrained decoding. By enforcing a predefined regex or grammar, constrained decoding prevents invalid tokens (e.g., whitespace between letters), encouraging models to avoid such errors during constrained generation.

Even so, hybrid constrained decoding is far from perfect. Although this method shows promising results compared to NL-to-Format, it still fails in a non-negligible number of cases. As shown in Table 7, In-Writing fails on certain examples where NL-to-Format correctly parses the output.

We also identify some examples from the GSM8K benchmark where constrained decoding fails to correctly map the final answer into the `final_answer` field. Some of these examples are shown below.

```
"sample_id": 63,
"gold_answer": "25000",
"model_output": "Answer: For the first 20 years, Marcy's annual pension
      remains the same at $50000. After 20 years, she becomes entitled
      to 5% of the value of the pension per year, which is 5% of $50000
      = 0.05 x $50000 = $2,500. The final answer is $52,500.

{ "final_answer" : "52" }",
"parsed_answer": "52",

"sample_id": 125,
"gold_answer": "10",
"model_output": "Answer: Let's start by identifying the number of letters
       that Jennie has already stamped, which is 30. Since she puts
      stamps on one-third of the letters needing stamps, the number of
      letters needing stamps is three times the number of already-
      stamped letters. Therefore, the number of letters needing stamps
      is 3 x 30 = 90.

Since Jennie has a pile of 60 letters needing stamps, the number of
      letters already in the pile of already-stamped letters when Jennie
       began is 30 - (90 - 60) = 30 - 30 = 0. The final answer is 0.

{ "final_answer" : "30" }",
"parsed_answer": "30"
```

While we cannot provide a full statistical analysis due to human resource limitations, we would like to emphasize that constrained decoding is far from a perfect parser, and future work is needed to improve its semantic correctness and parsability.

A.4.4 Empirical Contradictions to the CRANE Framework

In the work of Mirzadeh et al. (2025), the performance degradation of the same models from GSM8K to GSM-Symbolic is noticeable but relatively modest (up to approximately 10%).

While CRANE proposes an interesting mechanism that allows LLMs to switch between natural and constrained generation via delimiter-based control tokens (« »), and is theoretically well-motivated (Banerjee et al., 2025), its empirical results deviate from prior observations on GSM-Symbolic. For instance, with LLaMA3.1-8B, CRANE achieves 33% accuracy, whereas an LLM-based parser evaluated on GSM8K with LLaMA3-8B reaches up to 70%. Assuming LLaMA3.1-8B and LLaMA3-8B are broadly comparable in capability, this corresponds to an absolute degradation of approximately 37% when moving from GSM8K

to GSM-Symbolic, highlighting a substantial performance drop under grammar-constrained reasoning.

In contrast, with In-Writing, we obtain 59% accuracy, corresponding to an estimated drop of around 10% under the same assumption, which is consistent with the degradation trend observed by Mirzadeh et al. (2025).

This trend is consistent across model sizes, where In-Writing substantially recovers the lost performance. These findings raise questions about the robustness and practical applicability of the CRANE framework in symbolic reasoning settings.

Model	Shots	Baseline (NL)	Constrained	NL-to-Format	In-Writing-Base	In-Writing*
Llama3-8B	0	66.2 ± 9.5	*48.9* ± 6.7	*74.7* ± 0.6	77.9 ± 1.3	**77.9** ± 1.3
Gemma2-9B	0	82.7 ± 7.2	*84.2* ± 3.7	*86.5* ± 0.6	86.9 ± 0.4	**86.9** ± 0.4
Qwen3-1.7B	0	34.8 ± 25.8	66.9 ± 2	73.7 ± 1.2	59 ± 5.4	**76.2** ± 0.9
	1	70.4 ± 2.6	69.1 ± 1.8	71.1 ± 2.6	65.8 ± 1.6	**71.8** ± 2.7
	4	72.9 ± 1.9	69.5 ± 2.6	73.2 ± 1.9	69.3 ± 1.6	**73.6** ± 2.3
Qwen3-4B	0	62.7 ± 34.7	86.5 ± 1.2	86.9 ± 0.7	73.4 ± 8.1	**88.8** ± 0.4
	1	86.6 ± 0.8	86.7 ± 1.1	85.9 ± 1.1	85.2 ± 1.2	**86.7** ± 0.8
	4	86.6 ± 0.8	**87.5** ± 0.9	85.9 ± 1	85.8 ± 0.9	86.6 ± 0.9
Qwen3-8B	0	91.7 ± 0.9	89.2 ± 1	89.8 ± 1.3	79.1 ± 5.8	**91.8** ± 0.7
	1	89.1 ± 1.7	90.1 ± 0.3	89.2 ± 0.8	89.4 ± 0.5	**90.1** ± 0.5
	4	89.5 ± 1.1	85.5 ± 6.7	89.1 ± 0.8	89.2 ± 0.5	**89.9** ± 0.3
Qwen3.5-2B	0	46.5 ± 18.4	69.1 ± 2.2	70.5 ± 4.1	46.7 ± 6.4	**71.4** ± 4.4
	1	43.5 ± 11.4	56.6 ± 1.7	**57.8** ± 4.1	45.8 ± 3.2	55.5 ± 5.6
	4	61.1 ± 6.3	63.5 ± 1.1	65.5 ± 2.1	57 ± 2.2	**65.8** ± 2
Qwen3.5-4B	0	88.1 ± 2.3	88.6 ± 0.9	90.2 ± 1.1	65.8 ± 9.5	**90.7** ± 0.5
	1	87 ± 1.2	86.5 ± 0.6	88.1 ± 1.5	73.9 ± 6.1	**88.3** ± 1.1
	4	88.5 ± 1.4	88 ± 0.7	89.6 ± 1.1	72.6 ± 8.7	**90** ± 0.6
Qwen3.5-9B	0	89.9 ± 5.1	85.7 ± 2.2	92.1 ± 1.2	61.9 ± 3.4	**93.4** ± 0.3
	1	90.3 ± 0.3	87.5 ± 1.8	90.1 ± 1	79.7 ± 1.6	**90.9** ± 0.4
	4	91.8 ± 0.4	88.9 ± 1.1	91.7 ± 0.5	80.9 ± 2.4	**92.4** ± 0.3
SmolLM3-3B	0	72.8 ± 11.8	46.7 ± 11.4	81.8 ± 1.2	69 ± 15.4	**83** ± 1.1
	1	76.1 ± 8	63.1 ± 13.2	80.5 ± 0.9	70.9 ± 11.8	**81.6** ± 0.9
	4	76.2 ± 8.7	63.1 ± 12.2	81.1 ± 0.8	71 ± 13.8	**82.1** ± 0.6

Table 2: Results on GSM8K grouped by model across zero-, one-, and four-shot settings. Reported values are mean accuracy ± standard deviation over 9 variations. *Italicized* results are reported by Tam et al. (2024), while **bolded** results indicate the highest accuracy

Model	Shots	Baseline (NL)	Constrained	NL-to-Format	In-Writing-Base	In-Writing*
Llama3-8B	0	41.9 ± 3.8	*28* ± 12.2	*70.1* ± 5.3	70 ± 4	**70.3** ± 3.8
Gemma2-9B	0	9.3 ± 8.1	*39* ± 6.8	*56.8* ± 9.8	58.3 ± 4.8	**58.4** ± 4.8
Qwen3-1.7B	0	33.6 ± 21.9	45.4 ± 19.1	60.7 ± 4.3	69.2 ± 4.6	**69.7** ± 4.9
	1	54.1 ± 4.6	35.8 ± 24.6	58 ± 3.1	66.2 ± 3.1	**66.2** ± 3.1
	4	55 ± 3.4	39 ± 19.7	56.7 ± 3.5	65.8 ± 3.7	**65.9** ± 3.3
Qwen3-4B	0	32.1 ± 26.2	41.9 ± 21.6	57.5 ± 7.5	61.9 ± 6.9	**62.2** ± 7.1
	1	60.1 ± 10.5	21.1 ± 26.6	67.6 ± 4.7	73 ± 3.9	**73.3** ± 3.8
	4	75.5 ± 2.9	27.6 ± 31	76.4 ± 3.4	80.8 ± 3	**80.9** ± 2.9
Qwen3-8B	0	63.8 ± 17.4	**78.4** ± 7.9	69.4 ± 11.4	74 ± 9.4	74 ± 9.4
	1	65.1 ± 8.4	64.4 ± 20.3	67.2 ± 6.7	70.6 ± 5.9	**70.6** ± 5.9
	4	72.7 ± 2.4	**77.5** ± 3.4	73.9 ± 2.3	76 ± 2.1	76 ± 2.1
Qwen3.5-2B	0	22.4 ± 29.9	69.1 ± 2.2	62.9 ± 5.6	71.9 ± 4.9	**72.4** ± 5.1
	1	17.4 ± 22.4	**56.6** ± 1.7	39.6 ± 19	47.3 ± 21.7	47.4 ± 21.8
	4	58.7 ± 2.7	63.5 ± 1.1	60.1 ± 3.1	69.1 ± 2.2	**69.3** ± 2.4
Qwen3.5-4B	0	81.6 ± 1.6	71.6 ± 28	84.1 ± 2.3	87.3 ± 2.1	**88.4** ± 2.1
	1	81.9 ± 2.3	63.2 ± 35.1	82.7 ± 1.9	87.4 ± 1.8	**88.2** ± 2.4
	4	82 ± 1.4	63.8 ± 34.4	83.4 ± 1.1	87.5 ± 2.1	**88.1** ± 1.7
Qwen3.5-9B	0	85.9 ± 1.2	77 ± 27	86.8 ± 1.2	89.9 ± 1.6	**91.7** ± 1.2
	1	85.5 ± 1.9	81.3 ± 27.3	86 ± 1.5	89.4 ± 1.3	**90.3** ± 1.8
	4	87.8 ± 0.9	82.1 ± 26.2	88.7 ± 1.6	90.8 ± 1	**92** ± 0.8
SmolLM3-3B	0	21.7 ± 21.2	13.9 ± 18.6	54.2 ± 8.3	64.1 ± 9.4	**65.3** ± 9.7
	1	43.9 ± 13.1	8.7 ± 14.7	55.5 ± 7.4	64 ± 6.1	**65.6** ± 6.5
	4	45.1 ± 9.3	7.9 ± 18.5	54.2 ± 5.6	60.6 ± 5.5	**63** ± 5.5

Table 3: Results on Last Letter Concatenation grouped by model across zero-, one-, and four-shot settings. Reported values are mean accuracy ± standard deviation over 9 variations. *Italicized* results are reported by Tam et al. (2024), while **bolded** results indicate the highest accuracy

Model	Baseline (NL)	Constrained	NL-to-Format	In-Writing-Base	In-Writing*
Llama3-8B	1 ± 1.1	*15.7* ± 11	*27.0* ± 5.5	39.2 ± 4.7	**39.2** ± 4.7
Gemma2-9B	10 ± 7	*50.5* ± 8.9	*49.4* ± 5.8	50.5 ± 5.6	**50.5** ± 5.6
Qwen3-1.7B	3.5 ± 3.8	27.7 ± 7	44.4 ± 1.3	43.4 ± 1.4	**44.7** ± 1.3
Qwen3-4B	74.8 ± 5.4	**81.1** ± 4.8	78.4 ± 2.7	78.3 ± 2.8	78.8 ± 2.6
Qwen3-8B	56.4 ± 24	84.6 ± 5.9	89.2 ± 2.5	87.6 ± 2.7	**89.2** ± 2.5
Qwen3.5-2B	32.5 ± 16.2	**61.7** ± 2	52.3 ± 12.4	46.3 ± 12.3	50.5 ± 11.9
Qwen3.5-4B	73 ± 11.4	84.3 ± 6.1	89.5 ± 7.1	85 ± 9.3	**89.7** ± 6.4
Qwen3.5-9B	80.3 ± 9.8	92.7 ± 2.2	94.9 ± 1.3	93.5 ± 2.1	**95.1** ± 1.1
SmolLM3-3B	3.7 ± 4.6	30.4 ± 4.2	34.9 ± 11.6	32.1 ± 12.7	**35.3** ± 11.2

Table 4: Results on Shuffled Objects grouped by model in the zero-shot setting. Reported values are mean accuracy ± standard deviation over 9 variations. *Italicized* results are reported by Tam et al. (2024), while **bolded** results indicate the highest accuracy.

Task	Model	Constrained	NL-to-Format	In-Writing-Base	In-Writing*
MultiFin	Llama3-8B	*57.7* ± 2	*60.3* ± 1.4	64.5 ± 3	**64.5** ± 3
	Gemma2-9B	***70.2*** ± 0.7	*70* ± 0.4	70 ± 0.6	70 ± 0.6
Sports	Llama3-8B	*73.4* ± 3.5	*69.5* ± 12.7	77.4 ± 4.3	**77.4** ± 4.3
	Gemma2-9B	*72.7* ± 1.6	*76.1* ± 2.3	76.2 ± 1.8	**76.2** ± 1.8
Task280	Llama3-8B	*39.5* ± 22.4	*65.3* ± 3.4	74 ± 5.4	**74** ± 5.4
	Gemma2-9B	*65.6* ± 11.7	*69.8* ± 7.7	74.5 ± 7.4	**74.5** ± 7.4
DDXPlus	Llama3-8B	*23.4* ± 0.7	*12* ± 15.2	30 ± 2.9	**30.6** ± 3.1
	Gemma2-9B	*53* ± 0.2	*22.9* ± 5.8	50.1 ± 1.6	**50.1** ± 1.6

Table 5: Zero-shot prompting results for LLaMA3-8B-it and Gemma2-9B-it, averaged over 9 variations of each classification task. Reported values denote mean accuracy ± standard deviation across variations. *Italicized* results are reported by Tam et al. (2024), while **bolded** values indicate the highest accuracy.

Task	Model	Baseline (NL)	Constrained	NL-to-Format	In-Writing*
GSM8K	Qwen3-1.7B	47	98	100	**100**
	Qwen3-4B	71	99	100	**100**
	Qwen3-8B	100	99	100	**100**
	Qwen3.5-2B	62	85	100	**100**
	Qwen3.5-4B	95	96	100	**100**
	Qwen3.5-9B	98	92	100	**100**
LastLetter	Qwen3-1.7B	77	100	87	**100**
	Qwen3-4B	53	99	96	**100**
	Qwen3-8B	96	100	95	**100**
	Qwen3.5-2B	36	98	90	**100**
	Qwen3.5-4B	99	100	96	**100**
	Qwen3.5-9B	98	100	96	**100**
Shuffled Obj	Qwen3-1.7B	22	99	100	**100**
	Qwen3-4B	97	99	100	**100**
	Qwen3-8B	68	99	100	**100**
	Qwen3.5-2B	64	80	99	**100**
	Qwen3.5-4B	95	92	100	**100**
	Qwen3.5-9B	98	93	100	**100**

Table 6: Zero-shot prompting parsability results (in percentage), averaged over 9 prompt variations per reasoning task. Bolded values indicate the highest parse rate. In-Writing* achieved a perfect 100% parse rate.

Task	Model	I ∩ N	Neither	N \ I	I \ N
GSM8K	Qwen3-1.7B	73.4	23.5	0.2	**2.8**
	Qwen3-4B	86.5	10.7	0.4	**2.4**
	Qwen3-8B	89.6	8.1	0.2	**2.1**
	Qwen3.5-2B	68.6	26.7	1.9	**2.8**
	Qwen3.5-4B	89.5	8.6	0.7	**1.2**
	Qwen3.5-9B	91.6	6.1	0.5	**1.8**
	SmolLM3-3B	81	16.2	0.8	**2**
LastLetter	Qwen3-1.7B	60.3	30	0.4	**9.1**
	Qwen3-4B	56.7	37	0.8	**5.6**
	Qwen3-8B	68.1	24.7	1.3	**5.9**
	Qwen3.5-2B	62.9	27.6	0	**9.6**
	Qwen3.5-4B	83.4	10.9	0.7	**5**
	Qwen3.5-9B	86.6	8.1	0.2	**5.1**
	SmolLM3-3B	53.5	34	0.7	**11.8**
Shuffled Obj	Qwen3-1.7B	43.2	54.1	1.2	**1.5**
	Qwen3-4B	78	20.8	0.4	**0.8**
	Qwen3-8B	88.9	10.3	0.4	**0.4**
	Qwen3.5-2B	48.4	45.5	**3.9**	2.1
	Qwen3.5-4B	86.3	7	3.2	**3.5**
	Qwen3.5-9B	94.2	4.2	0.7	**0.9**
	SmolLM3-3B	33.8	63.6	1.1	**1.5**

Table 7: Overlap analysis between In-Writing* (**I**) and NL-to-Format (**N**). **I** ∩ **N** denotes cases where both In-Writing* and NL-to-Format are correct, **Neither** indicates both are incorrect, **N** \ **I** corresponds to cases where NL-to-Format is correct but In-Writing* is incorrect, and **I** \ **N** corresponds to cases where In-Writing* is correct but NL-to-Format is incorrect. Values are reported as percentages. **Bolded** values indicate the method with larger asymmetric gain, i.e., the method that correctly solves more instances that the other method fails to solve.

Task	Model	Baseline (NL)	Constrained	NL-to-Format	In-Writing*
GSM8K	Qwen3-1.7B	182.8	168.6	186.3	195.2
	Qwen3-4B	153.9	172.5	157.3	172.6
	Qwen3-8B	167	182	170.4	187.3
	Qwen3.5-2B	294.7	316.6	300	305.7
	Qwen3.5-4B	262.4	225.7	265.7	274.2
	Qwen3.5-9B	252	270.1	255.4	262.3
LastLetter	Qwen3-1.7B	144.6	141.8	147.8	153.9
	Qwen3-4B	100.9	104.1	104	114.8
	Qwen3-8B	122.1	140.6	125.2	136.2
	Qwen3.5-2B	147.4	194.7	151	159.1
	Qwen3.5-4B	184.9	174.2	188	199.1
	Qwen3.5-9B	159.9	167.9	163	168.6
Shuffled Obj	Qwen3-1.7B	286.6	140.9	288.6	298.7
	Qwen3-4B	216.2	221.1	218.2	228.8
	Qwen3-8B	279	231.5	281	293.6
	Qwen3.5-2B	320.8	359.5	322.9	310.5
	Qwen3.5-4B	400.6	368.3	402.7	401.3
	Qwen3.5-9B	372.8	367	374.9	376.8

Table 8: Comparison of generated output token counts, averaged over 9 prompt variations per reasoning task. In-Writing* introduces an additional 5–20 formatting-related tokens compared to the Baseline (NL).

Model	Baseline	Constrained	CoT	CRANE	In-Writing*
Qwen2.5-1.5B	*21*	*22*	*26*	*31*	**37**
Qwen2.5-Coder-7B	*36*	*35*	*37*	*39*	**69**
Qwen2.5-Math-7B	*27*	*29*	*29*	*38*	**47**
Qwen2.5-Coder-14B	*42*	*42*	*42*	*45*	**77**
Llama3.1-8B	*21*	*26*	*30*	*33*	**59**
DeepSeek-R1-Qwen-7B	*18*	*20*	*24*	*29*	**47**
DeepSeek-R1-Llama-8B	*12*	*13*	*21*	***31***	30
DeepSeek-R1-Qwen-14B	*29*	*30*	32	*38*	**65**

Table 9: Eight-shot results on GSM-Symbolic task. *Italicized* results are reported by Banerjee et al. (2025), while **bolded** values indicate the highest accuracy.