[THINKING] INSIDE THE BRACKETS: GRAMMAR-CONSTRAINED DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM reasoning is a core driver of model performance, but it is costly. Practitioners typically use prompting and fine-tuning as axes of control to reduce the thinking-verbosity of reasoning models. We identify and exhibit grammar-constrained decoding (GCD) as a third axis of control. We structure reasoning traces by a small context-free grammar at decode-time (e.g. open a subgoal, derive, add a fact, check, close) which is compatible with free-text payloads. On GPQA-Diamond and AIME24, GCD on Qwen3.6-27B reduces median generated tokens by 98% and 94%, while holding or improving accuracy over chain of thought. While GCD reliably makes reasoning succinct, we further identify and characterise cases where controlling reasoning is detrimental, or ineffective.

1 INTRODUCTION

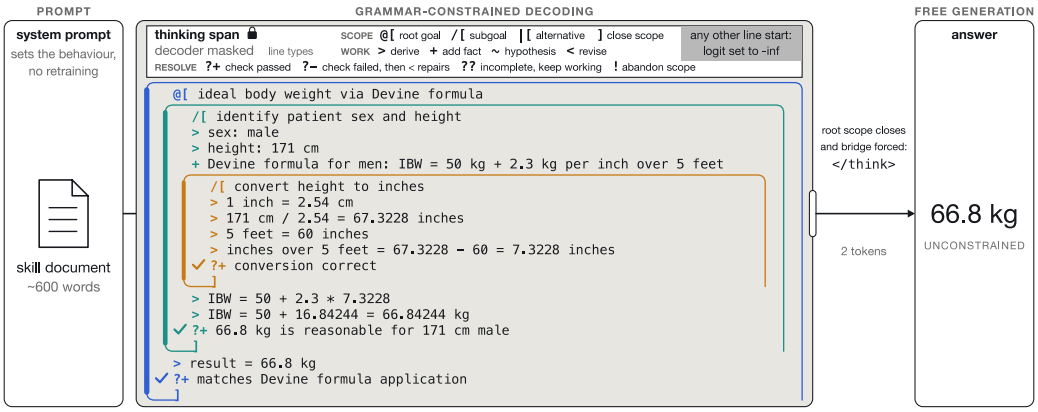

Figure 1: By dynamically constraining the decoder to a grammar that shorthands the discourse structure and functions of recurring natural language constructions in reasoning traces, we can obtain shorter and easily regexable (thus mechanically auditable) reasoning traces.

Chain-of-thought [CoT] (Wei et al., 2022) enjoys test-time-scaling (Snell et al., 2024; Muennighoff et al., 2025; DeepSeek-AI, 2025): longer reasoning traces generally yields better performance. Notably for modern consumer-grade open-weight LLMs, small model sizes can punch above their weight class by leveraging this phenomenon, training models to think longer. However, this results in a latency and cost problem, where quality comes at the time and cost of generating long reasoning traces.

The length of reasoning traces can be directly mitigated by standard 'thinking-effort' parameters, but this gives up on test-time-scaling. There are primarily two ways in practice to have your cake and eat it too, effectively increasing the quality-density of reasoning traces. the easier is prompting, such as Chain-of-Draft (Xu et al., 2025), which instructs the model to use a succinct format for its traces. The harder is a weight-finetune, such as ThinkingCap (Lasocki et al., 2026). We propose a third axis, at the decoder step.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

In the remainder, we survey related work, then introduce the GCD technique (Section 2) along with experimental data for it compared against and combined with prompting and finetuning (Section 3). We close with a discussion of limitations and future directions (Section 4).

Novel Contributions

- Grammar-Constrained Decoding. While constrained decoding (Geng et al., 2023; Willard & Louf, 2023; Beurer-Kellner et al., 2024; Dong et al., 2024; Scholak et al., 2021) and the idea of structuring reasoning traces according to grammars are not novel, their combination here along with symbolic shorthands appears to be.
- An apples-to-apples analysis of three axes of reasoning-interventions: GCD, Prompt, and Finetune, along with practical recommendations.

Related work

Constrained decoding. In contrast to extant methods, we constrain trace structure and keep its content in natural language. Masking the next-token distribution with a regular expression or context-free grammar is cheap enough to run at every step, whether by indexing the vocabulary against an automaton, prechecking context-independent tokens or caching grammar-derived masks (Willard & Louf, 2023; Dong et al., 2024; Ugare et al., 2024; Guidance AI, 2026b;a), and these tools are aimed at outputs a program consumes, e.g. JSON, code, extraction and parsing (Geng et al., 2023). Abstract Chain-of-Thought constrains the decoder inside the reasoning span, but it constrains the content of the reasoning trace rather than its structure, replacing the chain with a reserved codebook whose tokens are unreadable by design (Ramji et al., 2026). Tam et al. (2024) found that format restrictions lowered reasoning performance and that stricter formats cost more. CRANE responds by alternating spans, leaving the reasoning free and constraining only the output (Banerjee et al., 2025), whereas we do the reverse.

Controlling the length of thinking via weights or prompts. In contrast to weight finetunes, our method needs no training, and unlike a prompt, our structural requirements cannot be ignored because they are enforced by the decoder. Some attempts to control the cost of thinking traces proceed at the level of weights, e.g. by forcing a trace to stop or continue after fine-tuning on a small set (Muennighoff et al., 2025), by training with reinforcement learning to meet a requested length (Aggarwal & Welleck, 2025), or by training on compressed traces that skip low-value tokens or pair long and short chains (Xia et al., 2025; Kang et al., 2024). ThinkingCap fine-tunes Qwen3.6-27B for shorter traces (Lasocki et al., 2026) and is our experimental comparison. Other attempts proceed via prompt intervention. These set a token budget estimated from the problem (Han et al., 2024), ask for brief drafts (Xu et al., 2025) or supply sketching styles (Aytes et al., 2025). The motivation common to both is that long-reasoning models spend heavily on easy problems for little gain (Chen et al., 2024).

Typed and structured traces. Chain of thought (Wei et al., 2022) is free text, and program-of-thought and PAL move the reasoning into code for an interpreter to run (Chen et al., 2022; Gao et al., 2022). We strike a middle ground, such that GCD keeps natural-language payloads but within machine-friendly structure. More restrictively, typing a reasoning trace is thoroughly explored, although types are usually recovered post-hoc. ReasonOps (Lee et al., 2026) clusters sentence-initial pivots across tens of thousands of traces and recovers seven operators that map nearly one to one onto ours, which we take as independent validation of a hand-written set. Step-Tagging (Belkhiter et al., 2025) tags step types online with a sentence classifier, and typed schemas have been proposed under formalisms such as (Perrier, 2025). The difference here is in the origin of the types. Recovery has a cost associated with the an annotator error rate, reported at 70 to 76 % agreement on held-out samples (Lee et al., 2026), while a grammar gives the types exactly and at no classifier cost. Moreover, exact types are only worthwhile if they carry information, and ours by design record what the model would have written anyway, since operators of the grammar are the ones native traces already use.

Function	Native marker	GCD	Base % traces	Base /10k w.	ThinkingCap /10k w.
Goal	`I need to`	`@[`	76.4	8.6	10.0
Subgoal	numbered bold heading	`/[`	100.0	17.2	24.7
Alternative	`Alternatively`, `what if`	`|[`	22.7	1.6	1.4
Closure	`I'll stick with`/`keep`	`]`	70.0	9.7	7.2
Derivation	`So`, `Therefore`	`>`	25.8	2.1	1.6
Re-reading	`the prompt/constraint says`	`+`	76.8	11.2	8.9
Caveat	`To be safe`	`+`	60.5	4.9	3.8
	`One (minor) thing`	`+`	53.6	3.5	3.1
Hedge	`maybe`, `perhaps`	`~`	36.1	3.5	4.0
Interruption	`Wait,`	`<`	83.3	13.3	9.8
	`Actually,`	`<`	67.0	13.7	7.7
	`wait a minute`, `hmm`, `hold on`	`<`	$\leq$0.4	0.0	$\leq$0.1
Simplification	`I'll just`	`<`	53.2	8.5	6.4
Checking	`let's verify`/`check`	`?+`/`?-`	71.7	8.2	8.1
	`Final check`	`?+`	67.0	4.9	4.4
Verdict	$\surd$	`?+`	9.9	3.8	24.2
	`No.`	`?-`	27.0	10.8	15.8
Admission	`my mistake`, `I miscounted`	`?-`	2.6	0.1	0.1
Incomplete	`not sure`	`??`	0.4	0.1	0.0
Abandonment	`I give up on`	`!`	1.7	0.1	0.2

Table 1: Discourse markers in native thinking traces of Qwen3.6-27B (8-bit) on IFBench (233 traces, 561k words) and of its succinctness fine-tune ThinkingCap (300 traces).

2 Grammar-Constrained Decoding

When we inspect reasoning traces, we can see they are needlessly verbose and contain repeated semantically similar phrases in natural language that serve a small repertoire of discourse functions (Schiffrin, 1987), which can be replaced with shorthand (Table 1). Outside of discourse, we observe that reasoning traces naturally raise goals and subgoals, which typically nest, and which models tend to pursue to closure before considering other subgoals. We can model this nesting by bracketing structure. While by no means canonical, we can obtain the following grammar:

```
  trace  ::=  @[ text↩ body ]↩
  scope  ::=  /[ text↩ body ]↩  |  |[ text↩ body ]↩
   body  ::=  !↩  |  item+ resolve
   item  ::=  scope  |  (> | + | ~ | <) text↩
resolve  ::=  ?+ text↩
           |  ?- text↩ < text↩ item* resolve
           |  ??  text↩ item+ resolve
           |  !↩
```

Figure 2: The GCD grammar. Bold symbols are fixed by the grammar, and *text* is free generation up to a newline ↩. The mask ends at the root's closing bracket, after which the answer is unconstrained.

In prose, we always allow the reasoner to open new nested subgoals. Otherwise, we force the reasoner to mark every newline with the intent for that line, whether it is e.g. a deduction (`>`), a hedge (`~`), etc. We make room for signalling uncertainty (`??`) and revision/revisiting (`?+`).

By using a preamble skill document (Appendix A), we instruct the model to think according to the grammar, in particular noting the discourse functions corresponding to reserved symbols. We do not depend solely on the skill document to ensure format correctness: instead, we dynamically constrain decoding using the pushdown automaton that corresponds to the grammar (Figure 3).

The skill document allows the model to transition between states when it is done with free text, and the state-dependent masking forces compliance with the grammar. The time-costs of these interventions are negligible, and they result in reasoning-compactions as in Figure 4.

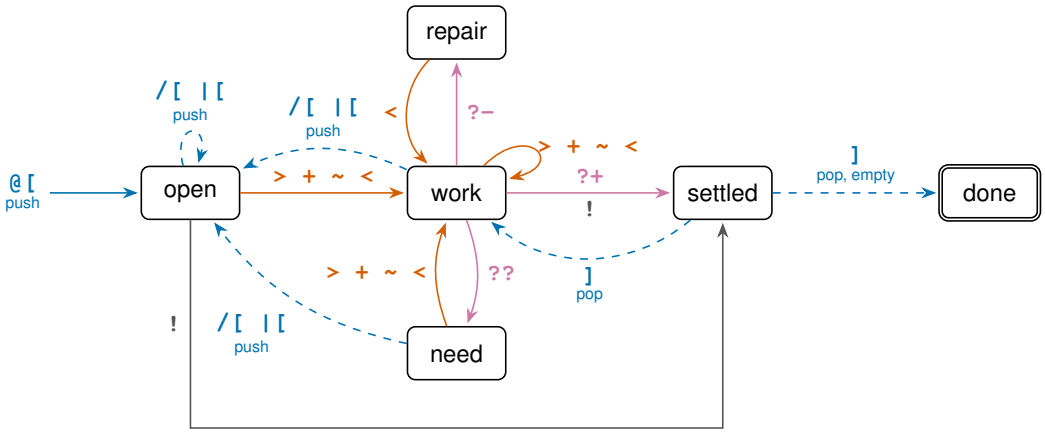

Figure 3: The GCD grammar as a pushdown automaton, where the stack counts nested contexts. Dashed edges use the stack: an opener pushes a scope, and] pops it, returning to work in the parent, or to done once the stack is empty.

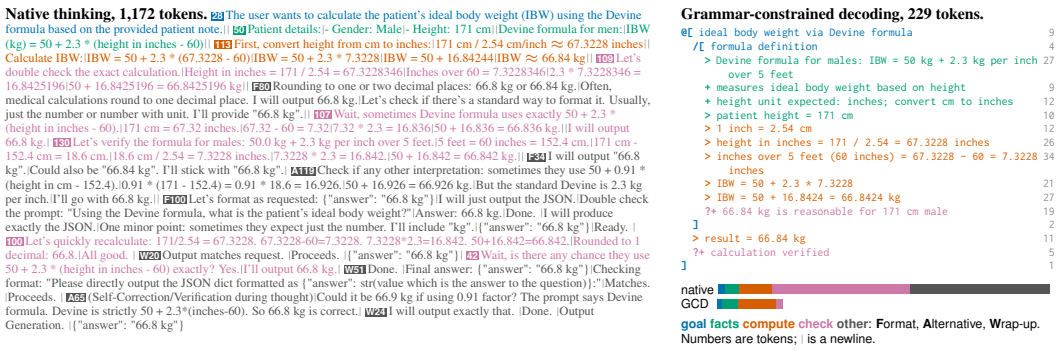

Figure 4: The same MedCalc item, solved by ThinkingCap (Qwen3.6-27B, 8-bit) with native thinking (left, in full) and with GCD (right). Both reach the answer after 191 tokens; native thinking then spends 981 more re-checking it, GCD 38.

3 Experiments

We evaluate all methods on Qwen3.6-27B (Qwen Team, 2026) in 8-bit MLX[1]. Throughout, we use the provider-recommended hyperparameters of temperature 1.0, top-p 0.95, top-k 20. For the fine-tuning method (FT) we use ThinkingCap (Lasocki et al., 2026), an open-weight fine-tune of the same model for short reasoning traces, and the prompting method (P) is Chain of Draft (Xu et al., 2025), which uses a short prompt to impose a succinct structure on the reasoning of the model. We evaluate each of these methods, GCD, and all combinations of techniques, to investigate their effect individually, and in combination. GCD includes a prompt that instructs the model to use the grammar (Section 2), and this prompt overlaps semantically with prompt-only methods. To combine GCD with P, redundancies were removed so brevity instructions were not repeated.

We evaluate our methods on the following datasets.

1. **GPQA-Diamond** (Rein et al., 2024): graduate-level multiple-choice questions in biology, chemistry and physics, written by domain experts, with four options each. Total cap 32K tokens.

[1]At the time of writing, this is the only model the authors are aware of that permits a head-to-head comparison against a finetune (ThinkingCap) for shorter reasoning traces.

2. **AIME24** (Mathematical Association of America, 2024): problems from the 2024 American Invitational Mathematics Examination, each with an integer answer from 0 to 999. Two samples per question scored. Total cap 32K tokens.

3. **MedCalc-Bench-Verified** (Khandekar et al., 2024; 2025): patient notes, paired with a question to compute a clinical quantity with one of 55 calculators, such as a lab-derived value, a risk or severity score, a date or a dose. We employ greedy decoding, with a cap of 8K tokens.

4. **IFBench** (Pyatkin et al., 2025): writing prompts with instructions to follow verifiable output constraints, such as counts, formats or required words, each checked by code; we use the first 233 of 300, with a fixed random order. We employ greedy decoding, with a cap of 8K tokens.

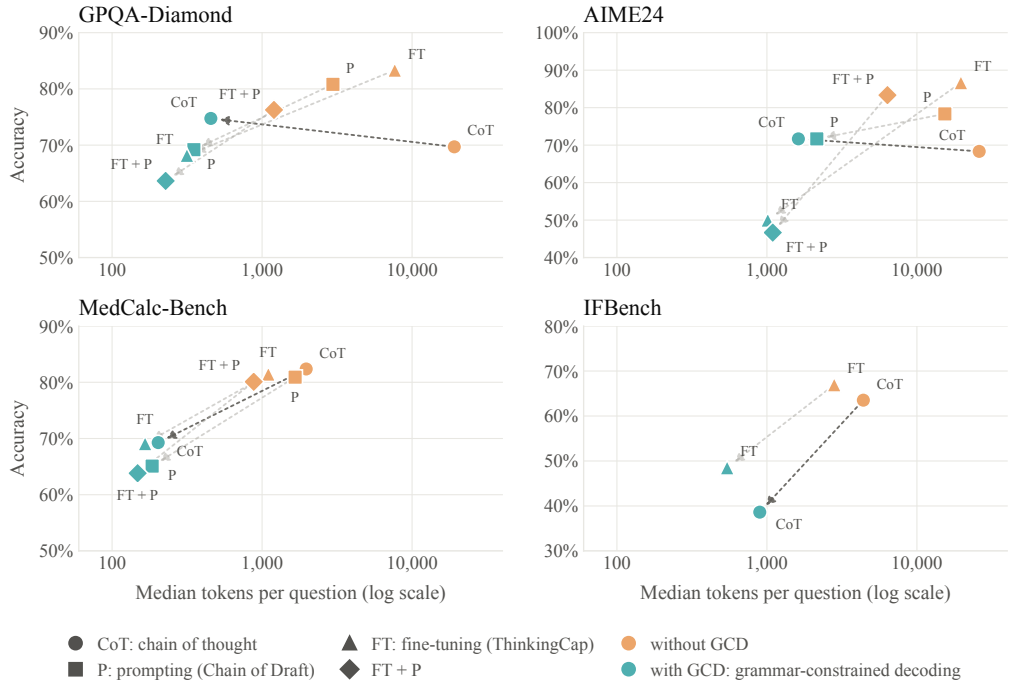

Figure 5: Accuracy against median generated tokens (log scale) for each method without GCD (orange) and with it (teal). Arrows join each method to the same method with GCD. Leftwards is faster, upwards is better.

For GPQA-Diamond (Figure 5, top left), all axes of control, applied independently, improve answer quality while reducing token counts substantially. GCD achieves the greatest reduction in token counts (98%), while increasing accuracy by 5 points. The same pattern holds on AIME24, where GCD reduces the token count by 94% and increases accuracy by 3 points. Combinations of techniques do not yield monotonic effects. Combining P and FT uses fewer tokens than either alone, with accuracy between that of CoT and the better of the two. Adding GCD to P or FT removes most of their gain: on GPQA-Diamond both combinations fall slightly below CoT, and on AIME24 P + GCD scores the same as GCD alone while FT + GCD falls to 50.0%. Combining all three techniques consistently yields the weakest performance.

For MedCalc-Bench and IFBench (Figure 5, bottom row), we see a regression in quality when using GCD, while retaining the token reduction benefits. Token counts were reduced by 90% and 80%, with a reduction in performance of 13 and 25 points. For MedCalc-Bench in particular, none of the methods, independently or in combination, improves quality over the baseline CoT. We discuss possible reasons for this in the following section.

In Figure 6 we plot the performance each of the methods on the same datasets, with responses truncated at increasing lengths. A higher curve indicates better response quality at the same token budget. Across all datasets, GCD dominates the quality curve at low token budgets, rising very fast, but reaches a lower maximum than P and FT in all cases. P and FT have curves shaped very similarly to that of CoT, but rise earlier; they peak at about the same level as CoT on MedCalc-Bench and IFBench, and higher on GPQA-Diamond and AIME24.

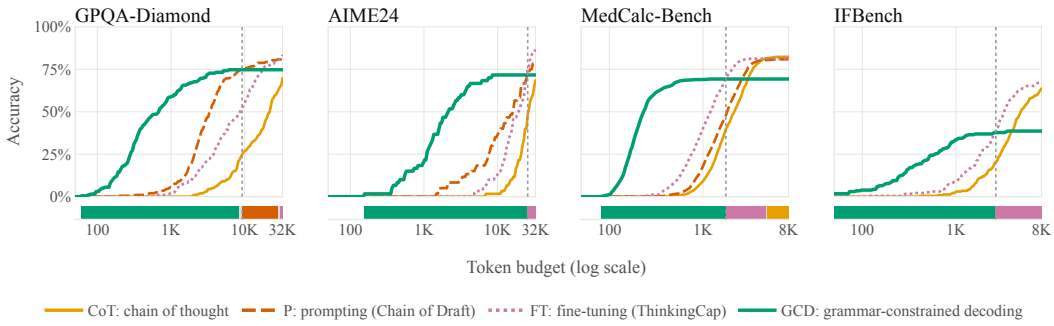

Figure 6: Accuracy against a token budget for CoT, P, FT and GCD. Accuracy at budget B counts traces that finished within B generated tokens with a correct answer; we compute it from the traces above. The strip under each panel shows the most accurate method at each budget, and the dashed vertical line marks where GCD stops being the most accurate.

4 Discussion

Among the individual methods, GCD gives the best accuracy at low token budgets across all datasets. At the full budget it matches or improves on CoT on GPQA-Diamond and AIME24, and it hurts accuracy substantially on IFBench and MedCalc-Bench. To investigate this, Figure 7 plots accuracy against token budget for the CoT baseline on all four datasets. The two datasets where GCD holds or improves quality are those where the model shows test-time scaling over long contexts: the accuracy of CoT is still rising at the 32K cap. On MedCalc-Bench, CoT saturates at about 5K tokens, well inside its cap, which suggests there is little headroom to shrink the reasoning trace without hurting quality. While IFBench is not saturated at 8K tokens, it shows a curve that rises much faster than the GPQA-Diamond and AIME24, suggesting similarly, that GCD is most useful when CoT accuracy rises with token budget over a longer context.

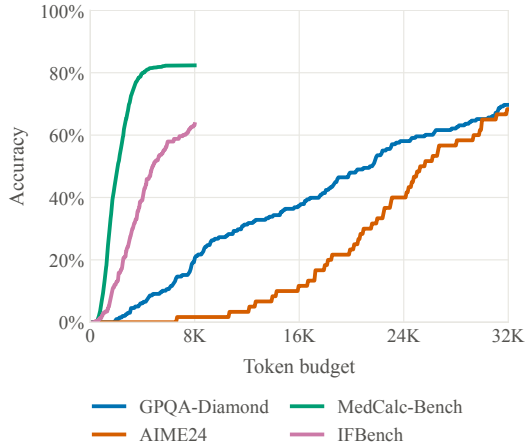

Figure 7: Accuracy of CoT against a token budget, computed as in Figure 6. MedCalc and IFBench end at their 8K cap.

P and FT compose sub-additively but non-destructively: on GPQA-Diamond and AIME24 their combination uses fewer tokens than either alone and stays above CoT in accuracy. GCD composes poorly with the other methods at the full budget, reducing token counts across the board at a significant cost to accuracy. Our experiments thus indicate that when token budgets are small, GCD, alone or with FT, is the dominant choice; on GPQA-Diamond and AIME24 GCD alone also holds the accuracy of CoT at a small fraction of its tokens; and when accuracy must approach the best available, FT alone or FT + P is preferable.

4.1 LIMITATIONS

Due to resource constraints, we ran limited sampling, and did not test other families or sizes. Our comparison with FT methods is currently limited to a single open-weight model, and other fine-tunes may compose differently with GCD. Moreover, the grammar is also a single design, and since we did not ablate its operators we cannot say in this work which of them produces the token saving. Accordingly, we make no broader claims about the comparative advantages of GCD in more generality.

Our token caps bound what CoT can reach (Figure 7). On GPQA-Diamond and AIME24 its accuracy is still rising at the 32K cap, where it stops on only 80.3% and 68.3% of items, and on IFBench it is still rising at the 8K cap. A larger cap could raise the accuracy of CoT on these three datasets, so our comparison of GCD with CoT holds at these caps only. On MedCalc, CoT levels off well within its cap. Our experiments show that at very small token budgets, GCD offers the best quality, at the expense of a lower plateau: it is however ins this regime where succinctness is most valuable.

FURTHER WORK

The strengths of GCD to exploit further are its non-exclusivity and versatility. As a separate axis of reasoning control, it is compatible in principle with task-specialised prompt engineering techniques such as GEPA (Agrawal et al., 2026), and task-specific fine-tuning. For resource-constrained users of LLMs running local models, GCD is an attractive option for latency- and cost-reduction.

GCD moreover suggests a novel design-space of specialised grammars. For work where model reasoning must be inspectable and justified in volume, GCD offers regex hooks for easier processing, addressing calls for greater legibility into uses of LLMs for decision-making. For applications where the model ought to interface with external computations (such as calculators to avoid arithmetic errors, or more general tool-calls or MCPs), GCD offers a balance of flexibility and user-control.

AI USE STATEMENT

Coding agents (Claude Fable and Opus) assisted with writing code, finding related work, and LaTeXhousekeeping. All substantive decisions in research and writing were made by the authors.

REFERENCES

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning, 2025. URL https://arxiv.org/abs/2503.04697.

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2507.19457.

Simon A. Aytes, Jinheon Baek, and Sung Ju Hwang. Sketch-of-thought: Efficient LLM reasoning with adaptive cognitive-inspired sketching, 2025. URL https://arxiv.org/abs/2503.05179.

Debangshu Banerjee, Tarun Suresh, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. CRANE: Reasoning with constrained LLM generation, 2025. URL https://arxiv.org/abs/2502.09061.

Yannis Belkhiter, Seshu Tirupathi, Giulio Zizzo, and John D. Kelleher. Step-tagging: Toward controlling the generation of language reasoning models through step monitoring, 2025. URL https://arxiv.org/abs/2512.14332.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Guiding LLMs the right way: Fast, non-invasive constrained generation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 3658–3673, 2024. URL https://proceedings.mlr.press/v235/beurer-kellner24a.html.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks, 2022. URL https://arxiv.org/abs/2211.12588.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, et al. Do NOT think that much for 2+3=? on the overthinking of o1-like LLMs, 2024. URL https://arxiv.org/abs/2412.21187.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. XGrammar: Flexible and efficient structured generation engine for large language models, 2024. URL https://arxiv.org/abs/2411.15100.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models, 2022. URL https://arxiv.org/abs/2211.10435.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured NLP tasks without finetuning, 2023. URL https://arxiv.org/abs/2305.13971.

Guidance AI. Guidance: A guidance language for controlling large language models, 2026a. URL https://github.com/guidance-ai/guidance. Software repository, accessed 19 September 2026.

Guidance AI. Low-level guidance (llguidance), 2026b. URL https://github.com/guidance-ai/llguidance. Software repository, accessed 19 September 2026.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware LLM reasoning, 2024. URL https://arxiv.org/abs/2412.18547.

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. C3oT: Generating shorter chain-of-thought without compromising effectiveness, 2024. URL https://arxiv.org/abs/2412.11664.

Nikhil Khandekar, Qiao Jin, Guangzhi Xiong, Soren Dunn, Serina S. Applebaum, Zain Anwar, et al. MedCalc-Bench: Evaluating large language models for medical calculations, 2024. URL https://arxiv.org/abs/2406.12036.

Nikhil Khandekar et al. MedCalc-Bench-Verified. Dataset, Hugging Face Hub, 2025. URL https://huggingface.co/datasets/nsk7153/MedCalc-Bench-Verified. Corrected-label successor to MedCalc-Bench. Accessed 19 September 2026.

Karol Lasocki, Adam Osusky, Jan Lindauer, Adam Jirkovsky, Filip Mihal, Ondrej Platek, David Herel, Luka Ihnatchenko, Vojtech Bartek, Jiri Jirak, and Tomas Mikolov. ThinkingCap-Qwen3.6-27B model card, 2026. URL https://huggingface.co/bottlecapai/ThinkingCap-Qwen3.6-27B. Original fine-tuned model card.

Daniel Lee, Owen Queen, and James Zou. Reasonops: Operator segmentation for llm reasoning traces, 2026. URL https://arxiv.org/abs/2605.29192.

Mathematical Association of America. American invitational mathematics examination (AIME) 2024, I and II. Problem set distributed as HuggingFaceH4/aime_2024 on the Hugging Face Hub, 2024. URL https://huggingface.co/datasets/HuggingFaceH4/aime_2024. Accessed 19 September 2026.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025. URL https://arxiv.org/abs/2501.19393.

Elija Perrier. Typed chain-of-thought: A curry-howard framework for verifying llm reasoning, 2025. URL https://arxiv.org/abs/2510.01069.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following, 2025. URL https://arxiv.org/abs/2507.02833.

Qwen Team. Qwen3.6-27B model card, 2026. URL https://huggingface.co/Qwen/Qwen3.6-27B. Model card, accessed 19 September 2026.

Keshav Ramji, Tahira Naseem, and Ramón Fernandez Astudillo. Thinking without words: Efficient latent reasoning with abstract chain-of-thought, 2026. URL https://arxiv.org/abs/2604.22709.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. In *First Conference on Language Modeling (COLM)*, 2024. URL https://arxiv.org/abs/2311.12022.

Deborah Schiffrin. *Discourse Markers*, volume 5 of *Studies in Interactional Sociolinguistics*. Cambridge University Press, Cambridge, 1987. doi: 10.1017/CBO9780511611841.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models, 2021. URL https://arxiv.org/abs/2109.05093.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung yi Lee, and Yun-Nung Chen. 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*, pp. 1218–1236, 2024. doi: 10.18653/v1/2024.emnlp-industry.91. URL https://aclanthology.org/2024.emnlp-industry.91/.

Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. SynCode: LLM generation with grammar augmentation, 2024. URL https://arxiv.org/abs/2403.01632.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2022. URL https://arxiv.org/abs/2201.11903.

Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models, 2023. URL https://arxiv.org/abs/2307.09702.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. TokenSkip: Controllable chain-of-thought compression in LLMs, 2025. URL https://arxiv.org/abs/2502.12067.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by writing less, 2025. URL https://arxiv.org/abs/2502.18600.

A GRAMMAR-CONSTRAINED DECODING SKILL

Compact Reasoning

Name: `compact-reasoning` **Purpose:** Use a bracketed symbolic reasoning dialect to minimise thinking tokens without losing substantive reasoning, then answer in the format the question asks for.

A reply has two spans written in two different languages. The thinking block is **bracketed RDL**: minimal substantive payload plus explicit control operators, never narrated prose. Everything after the thinking block is the answer in the format the question asked for, and carries no operators at all.

Control grammar

- `@[` opens the root goal.
- `/[` pushes a subgoal.
- `|[` pushes an alternative branch.
- `>` derives a new proposition.
- `+` states a fact the following lines need.
- `~` records a tentative hypothesis.
- `?+` records a check that passed; the scope may then close.
- `?-` records a check that failed; the next line must begin `<` and repair it.
- `??` records a check that did not complete; continue in the same scope.
- `<` revises, repairs or rejects the current line of reasoning.
- `]` closes the current scope.
- `!` abandons a scope; it stands alone, with no payload.

An opening bracket occurs only in `@[`, `/[` or `|[` at the start of a line. A closing `]` stands alone on its line. Every line is an operation: there are no blank lines, no indentation and no line without a payload. A scope needs at least one work line before its first check, and may close only after `?+` or `!`. Every scope opened must be closed, and nothing follows the root `]`.

The shape of a line

1. Begin with exactly one `@[` root scope, on the first line.
2. A payload is one clause: a relation, a substitution, a value with its unit, a named object, a verdict. Keep it under about fifteen words, and let it end with its line.
3. A payload states; it does not deliberate. No question to yourself, no aside, no `wait`, `maybe`, `let us`, `actually`. Deliberation is carried by the operator you choose, not by talking through it.
4. Prefer fragments, symbols, identifiers and domain terminology to sentences. Do not restate an established fact, repeat a result already in hand, or rehearse the answer format inside the trace. The current proposition is the implicit antecedent: name a referent only when omitting it could confuse it with another live referent.

Scopes

5. Open a `/[` or `|[` only for work that needs three or more lines of its own. Ruling a candidate out in one line is a `>` line in the scope you are in, and candidates ruled out for the same reason share one line.
6. `?-` and `<` continue the scope they are in; they do not close it. Only `]` closes a scope, and it closes exactly one. Never open a second scope before writing the `]` that closes the first, and count the `]` you still owe before opening any scope.

7. An alternative you have finished with is closed, not left behind: `!` then `]` when it leads nowhere, `?+` then `]` when it settles something. One level of scope below the root is almost always enough.

Checks and stopping

8. `?+` asserts that the check succeeded. Make it independent: recompute by a second route, test a sign, magnitude or limit, or substitute back. A line that only restates the number is not a check. One such check settles a scope; do not check the same thing twice.

9. A check that fails is `?-` and then one `<` repair. If the repair still does not reconcile, stop repairing: record on one `?+` line the best-supported result together with what did not reconcile, and close, or `!` and close if nothing is supported. Never attack the same discrepancy a third time.

10. A scope that reaches about a dozen work lines without a check has lost the thread. Take the best-supported result to a `?+` and close it. Do not enumerate candidates already rejected, re-open settled ground, or invent a payload to fill a line.

Closing the trace

11. The root goal is the whole of what the final answer must state, not an intermediate on the way to it. If the question asks for several named parts, name them all in the root payload and give each its own `/ [`; the root closes only when every part is settled.

12. The root's last `?+` line states the established answer, in full, with the check that confirms it. Close every open scope, then the root. Nothing may be left to work out after the root `]`.

After the thinking block

13. The thinking block ends at the root `]` and the dialect ends with it. No line after it may begin with `@ [`, `/ [`, `| [`, `>`, `+`, `~`, `?+`, `?-`, `??`, `<`, `]` or `!`, and no scope may be opened there.

14. Write the final answer in exactly the format named by the instruction at the end of the question, and use that format even when the answer is a single symbol, seems obvious, or has just been stated in the trace. If the instruction names a wrapper, a field or a key, reproduce it literally.

15. Do not re-derive, re-check or re-present the working, and do not restate the question. An instruction to reason or work step by step is already satisfied by the thinking block. Keep this span under forty words besides the required wrapper; the wrapper alone is a complete answer span.

Examples

```
@[ smallest group
> x+x+(x-10)=200
> x=30
?- 2*30+(30-10)=80 != 200
< 3x=210
> x=70; smallest=60
?+ 70+70+60=200; smallest is 60
]
```

The thinking block ends at that `]`. What follows is the requested format carrying 60, and nothing else.

```
@[ x and y
/[ x
> 3x=210
?+ 3*70=210; x=70
]
/[ y
|[ y from the ratio
> ratio needs a quantity not given
!
]
> y=2x-80
?+ 2*70-80=60; y=60
]
?+ both parts settled; x=70, y=60
]
```

Optimise for the shortest balanced trace that settles the whole root goal, then state the answer in the requested format and stop.