Logically Constrained Decoding

Franklin Ma and Alan J. Hu
Department of Computer Science
University of British Columbia
franklin.ma@ubc.ca, ajh@cs.ubc.ca

Abstract

Constrained decoding is a state-of-the-art technique for restricting the output of a Large Language Model (LLM) to obey syntactic rules, e.g., a regular expression or context-free grammar. In this paper, we propose a method for extending constrained decoding beyond syntactic constraints, to enforcing formal, logical constraints that reflect some world model being reasoned about. We demonstrate proof-of-concept implementations for the game of chess, and for propositional resolution proofs: we constrain the LLM's decoding such that the LLM is free to output whatever tokens it wants, as long as it does not make illegal moves (chess) or unsound proof steps (resolution). We believe this technique holds promise for improving LLMs' generation of precise, formal reasoning, as is particularly necessary for mathematics.

1 Introduction

Proof is the quintessential distinguishing feature of mathematical discourse. Like other forms of argumentation, the statements in a proof must be syntactically correct and semantically meaningful, and the overall text should lead to the desired conclusion. What makes proofs distinctive is that each statement must be *sound*, i.e., it must obey formal logical rules with respect to the preceding statements. This is akin to the moves in a game or puzzle: each step must be a legal move. Many applications require such precise, correct, logical reasoning, underscoring the importance of research on NLP for mathematics.

Large Language Models (LLMs) have made extraordinary progress on mathematical reasoning, e.g., both OpenAI and Google DeepMind recently announced gold-medal-level performance on International Math Olympiad problems (Wei, 2025; Luong and Lockhart, 2025). However, even leading-edge frontier models frequently make mistakes. In this paper, we do not concern ourselves with wild hallucinations, but rather focus on formally invalid reasoning: statements that, given the precise logic of the world model underlying the reasoning, are illegal or incorrect. For example, we pitted ChatGPT-5 against the well-known chess engine Stockfish (Romstad et al., 2008–present) in a casual game.[1] ChatGPT played a solid opening, but stumbled a bit in the mid-game, reaching the position shown in Fig. 1 with ChatGPT (White) to play its 30th move. At this point, ChatGPT attempted a flagrantly illegal move, presumably

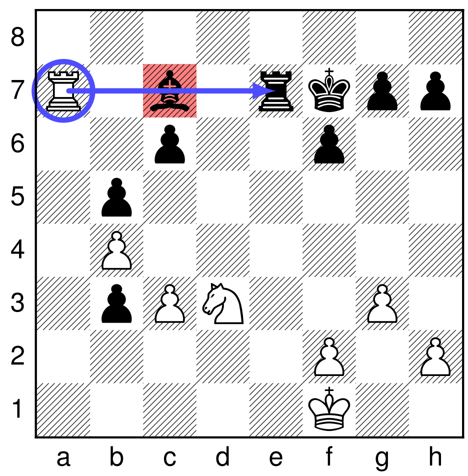

Figure 1: ChatGPT-5 (White) is playing against Stockfish 17.1 (Black). On move 30, White attempts Rxe7+ (shown in blue), i.e., taking Black's rook at e7 and attacking Black's king with White's rook at a7. This is illegal as Black's bishop on c7 is in the way.

[1]The details of the game are unimportant, but for the curious: We used Stockfish version 17.1, with default settings except a depth-limit of 15. ChatGPT-5 played White, and Stockfish played Black. The moves played were: 1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. O-O Nxe4 6. d4 b5 7. Bb3 d5 8. dxe5 Be6 9. c3 Be7 10. Re1 O-O 11. Nbd2 Bc5 12. Nxe4 dxe4 13. Qxd8 Rfxd8 14. Rxe4 Bxb3 15. axb3 Rd1+ 16. Re1 Rxe1+ 17. Nxe1 Nxe5 18. Bf4 Bd6 19. Bxe5 Bxe5 20. Nd3 Bd6 21. Re1 a5 22. Ra1 f6 23. Kf1 Kf7 24. Ke2 c6 25. g3 Bc7 26. b4 Re8+ 27. Kf1 a4 28. b3 axb3 29. Ra7 Re7, whereupon ChatGPT attempted an illegal move.

Proceedings of The 3rd Workshop on Mathematical Natural Language Processing (MathNLP 2025), pages 150–167
November 9, 2025

indicating that it had lost track of the underlying world model (the board state). Or, for a more mathematical example, we asked ChatGPT-5 to produce a resolution-based refutation proof for a pigeonhole problem with 3 pigeonholes. (See §2.2 and §4.2 for more explanation.) ChatGPT produced 74 logically sound (but sometimes useless or repetitive) resolution steps, before declaring on the 75th step, confidently but completely unfoundedly, that it had reached a contradiction and completed the "proof". We witnessed similar failures with Claude Sonnet 4.

Considerable research has boosted performance of language models on complex reasoning tasks, with techniques like chain-of-thought (Wei et al., 2023) and reinforcement learning with verified rewards (Wang et al., 2025). However, such techniques do not *guarantee* that the LLM will not produce logically illegal outputs. Furthermore, it seems inefficient to try to train language models to do fully precise, logical reasoning, when existing symbolic techniques can handle that well. For example, Pan et al. 2025 show that it is theoretically possible for a custom-programmed transformer to decide propositional satisfiability (albeit inefficiently), but that an empirically trained transformer for 3SAT generalizes and scales poorly; in contrast, existing SAT solvers routinely solve practical problem instances with millions of variables. We believe a neuro-symbolic approach — i.e., augmenting the language model with logical, symbolic reasoning — holds great promise to marry the best attributes of both approaches.[2]

Specifically, we build our work on constrained decoding, a state-of-the-art technique for restricting the output of an LLM to obey syntactic rules, e.g., a regular expression (Beurer-Kellner et al., 2023; Willard and Louf, 2023) or context-free grammar (Willard and Louf, 2023; Ugare et al., 2024).[3] We propose to lift the concept of constrained decoding beyond syntactic constraints, to enforcing formal, logical constraints that reflect some underlying world model. We demonstrate proof-of-concept implementations for the game of chess, and for propositional resolution proofs. We show that our method is easily implemented with several different open-source language models, ensuring generation of guaranteed-correct outputs, while not otherwise perturbing the language models.

2 Background

2.1 Chess

For 60 years, the game of chess has been proclaimed "the Drosophila of AI".[4] "Drosophila" refers to *Drosophila melanogaster*, a species of fruit fly that has been a favorite subject of biological research as a model organism: they are cheap and fast to raise, relatively simple for experiments and analysis, yet they can illuminate the same concepts important in larger and more relevant organisms. AI research has used chess for exactly analogous reasons, and we follow this tradition by using chess for our initial implementation and experiments.

Chess is a two-player, deterministic (i.e., there is no luck involved), perfect-information (i.e., there is no hidden information, like face-down playing cards), turn-based (i.e., the players take turns making moves) game. Each player starts with a standard set of playing pieces, arranged on the playing board in a standard configuration. One player (dubbed "White") plays the light-colored pieces, and moves first; the other player ("Black") plays the dark-colored pieces. There are a variety of types of pieces, with specific formal rules governing how each piece is allowed to move on the board, and to "capture" (remove from the board) pieces from the opposing player. For example, in the board position shown in Fig. 1, White's piece on square a7 is called a "rook" and is allowed in a single turn to move any distance vertically or horizontally, but only through empty squares. It could also move to square c7, resulting in the removal from the board of Black's "bishop" currently on that square. But it is not allowed to move past c7, because Black's bishop occupies that square and blocks further movement. Each player has one distinguished piece, called the "king", and to win the game, a player tries to reach a game state in which one is attacking the opponent's king such that they cannot prevent their king being captured (called "checkmate"). It is also possible for a game

[2]There is even intriguing neuroscience evidence in support of such an approach. In a brain imaging study on highly educated subjects, professional mathematicians used completely different neural pathways to solve math problems, whereas the non-mathematically trained subjects relied solely on their language pathways, with lower accuracy (Amalric and Dehaene, 2016).

[3]In recent work, Mündler et al. 2025 have also extended constrained decoding beyond syntactic constraints, to generate type-safe code. Our work is philosophically very much aligned with theirs.

[4]According to Ensmenger 2012, this metaphor originated with Russian mathematician Alexander Kronrod in 1965 and first appeared in print in (Simon and Chase, 1973).

to end as a draw, in which neither player wins. For example, if a player has no legal moves, but his king is not under attack, then the game ends in a draw.

Chess has a rich literature, spanning centuries. We have presented just enough concepts so that a reader unfamiliar with chess can follow the key points of this paper. We reiterate that our goal is not to produce a superior chess engine, but to use chess as an example of an underlying world model with formal rules, which we can use to constrain an LLM playing chess, such that it never makes an illegal move.

2.2 Propositional Resolution Proofs

In propositional logic, all variables are Boolean (true/false), and there are no function or predicate symbols. Many logical operators are standard, e.g., AND, OR, NOT, etc., but it is standard to assume that formulas are in conjunctive normal form (CNF): a *literal* is either a variable x, or its negation $\overline{x}$; a *clause* is the disjunction of a set of literals, e.g., $(x_1 + x_2 + \overline{x_3})$; and a formula is the conjunction of a set of clauses, e.g., $(x_1 + x_2 + \overline{x_3})(x_3 + \overline{x_4} + x_5)$. (We use $+$ to denote OR, and juxtaposition to denote AND.) Propositional logic is the foundational layer of logical reasoning, making it an ideal testbed for the reasoning capabilities of any AI system. As such, we propose that propositional logic be the drosophila of reasoning.[5]

Formally, a mathematical proof is simply a sequence of statements, leading from a set of assumptions to a desired conclusion, such that (1) each statement is logically implied by the assumptions and preceding statements in the proof, and (2) this implication can be efficiently checked, usually syntactically according to the rules of a given proof system. Specifically, in this paper, we focus on proof by *resolution*: given two clauses $(A_1 + \cdots + A_n + x)$ and $(B_1 + \cdots + B_m + \overline{x})$, where the A_i and B_j are literals, and x is some variable, then the conjunction of the two clauses implies the clause (called the "resolvent") $(A_1 + \cdots + A_n + B_1 + \cdots + B_m)$. In a proof by resolution, each statement in the proof must be the resolvent of clauses from the assumptions or previously generated proof statements. Resolution is known to be a sound (i.e., no false statement can be proven) and complete (i.e., any true statement can be proven) proof system. Without loss of generality, we further restrict ourselves in this paper to proofs by *refutation*, meaning that the desired conclusion is to imply *false*, which proves that the original assumptions are a contradiction.

For convenience in interacting with the text-based LLMs, we adopt the common convention of denoting variables simply by their number, and denoting negation using the minus sign. So the earlier example of clauses $(x_1 + x_2 + \overline{x_3})(x_3 + \overline{x_4} + x_5)$, would be denoted $(1 + 2 + -3)$ $(3 + -4 + 5)$.

2.3 Constrained Decoding

At a high level, a typical LLM works as follows:

initialize $buf \leftarrow$ initial prompt
repeat
    $dist \leftarrow \text{Softmax}(\text{Nnet}(buf))$
    sample $next\_token$ from $dist$
    append $next\_token$ to buf
until $next\_token = EOS$

where buf is the context buffer; Nnet is the neural network in the LLM that produces weights for each possible next token; $dist$ is a probability distribution over the possible next tokens, generated via some version of softmax; and EOS is the end-of-sequence token.

The goal is to constrain the LLM to generate only output that obeys some syntax rules. But given the large investment in training the network Nnet, we do not want to modify it. And given that evaluating $\text{Nnet}(buf)$ is slow, we wish to avoid any backtracking or speculative evaluation.

Constrained decoding takes advantage of this basic LLM architecture to modify only the decoding step, to mask out illegal token choices: (Changes are highlighted in green.)

initialize $buf \leftarrow$ initial prompt
initialize parser $P.\text{INIT}(buf)$
repeat
    $dist \leftarrow \text{Softmax}(\text{Nnet}(buf))$
    $mask \leftarrow P.\text{LEGALNEXTTOKENS}()$
    disallow in $dist$ any token not in $mask$
    sample $next\_token$ from $dist$
    append $next\_token$ to buf
    update $P.\text{UPDATESTATE}(next\_token)$
until $next\_token = EOS$

Here, P is some sort of parsing engine for the syntactic constraints being enforced. For example, if

[5] Pan et al. 2025 express a similar sentiment: "Boolean SAT solving captures the essence of deductive logical reasoning because: 1) Boolean logic lies as the foundation of all logical reasoning, and 2) many modern SAT solvers are inherently formal deductive systems that implement the resolution proof system."

we wish to force the LLM output to obey a regular expression, then P could maintain a finite-state automaton that tracks all states from which there is a path to an accepting state. By restricting the next token to always be in $P.\text{LegalNextTokens}()$, we guarantee that the generated output cannot violate the regular expression.

Constrained decoding has the desired properties: Nnet is not modified, and not evaluated more than necessary, and the output is *guaranteed* to obey the syntactic restrictions. An additional desirable property is that it is "minimally invasive" (Beurer-Kellner et al., 2024), meaning all legal behavior of the LLM is still allowed, with the same relative probabilities. Clever implementation can make constrained decoding very efficient. For example, "token misalignment" occurs if the LLM and parser tokenize the text stream differently; this can be solved efficiently by pre-computing what is essentially small automaton that performs a limited look-ahead at what tokens the parse is prepared to accept. (Beurer-Kellner et al., 2024; Hamilton and Mimno, 2025) It is also useful to be able to switch between constrained and unconstrained decoding, because restricting the LLM to only constrained output can limit its reasoning ability. (Banerjee et al., 2025) This can be accomplished easily by having the parser recognize specific token sequences to start and stop enforcing constraints.

3 Logically Constrained Decoding

But what if we wish to enforce richer constraints than mere syntax? For example, we might wish to generate a program that obeys specified functional properties, or a mathematical proof that is logically sound. For such an application, syntactical constraints are insufficient, because there is an underlying world model, upon which the correctness or incorrectness of an output depends. For program correctness, this world model might include the values and types of program variables, assumptions on program paths, and the semantics of various operators. For mathematical proof, the underlying model might include constraints (assumed or derived) on the domains and interpretations of all formal symbols, and the status of assumptions and proof goals.

In this paper, we propose the concept of *logically constrained decoding*. We retain the framework of constrained decoding, with its desirable attributes, but we seek to enforce formal, logical constraints that reflect some world model underlying the reasoning.

The basic idea is actually very simple. The key insight is that the parser P in (normal, syntactic) constrained decoding is *already* doing logically constrained decoding, but just for a very limited, underlying world model. For regular expressions, the world model is just a finite automaton; for CFGs, a pushdown automaton. Why not substitute a richer world model?

So, in logically constrained decoding, we generalize the parser into a symbolic constraint engine. Just as in normal constrained decoding, it watches the generated tokens, and updates the state of its internal world model. And just as in normal constrained decoding, it reasons about this world model to mask out illegal next tokens, guaranteeing that the generated output is correct with respect to this underlying world model. The difference is that this world model can be more complex, involving symbolic reasoning.

The challenge, of course, is the LegalNextTokens operation. For purely syntactic constraints, with a finite or pushdown automaton as the underlying world model, the legal next tokens can be calculated via standard automata-theoretic techniques. But for more general constraints, it's not obvious that one can compute the set of legal next tokens, i.e., tokens that are the next token in the prefix of an overall correct output. Are there any non-trivial, non-syntactic world models for which we can implement logically constrained decoding? We answer this question affirmatively with two simple, but non-trivial examples: chess and propositional resolution proofs.

Chess, as a "Drosophila" experiment, turns out to be easy, but illustrates constraining LLM output according to formal, logical rules completely unlike the syntax-focused prior work on constrained decoding. The underlying world model is simply the state of the chess board.[6] As the LLM generates chess moves, the symbolic constraint engine updates the board state. For the LegalNextTokens operations, the constraint engine solves for the set of legal next moves in the current game state, looks at the partially generated move from

[6]Chess afficianados will note that aside from the obvious positions of pieces on the board, state includes some additional information, like *en passant* pawns, castling options, etc. But this is all finite-state and well-documented, e.g., in Forsyth-Edwards notation.

the LLM (if any), and allows any token that can lead to generating one of the complete legal moves.

Propositional resolution proofs are our second, more complex example. Propositional proofs are the essence of mathematical proof, and as mentioned earlier, resolution lies at the core of practical, modern SAT solvers. In this case, the state of the underlying world model is the set of clauses that were either given as assumptions, or have been proven already. A legal "move" is a resolvent of existing clauses. (We can also enforce that the move does not generate a duplicate of an existing clause, or a useless tautology like $(x+\overline{x})$.) When the LLM is trying to generate its next move, the symbolic constraint engine enforces that each additional token maintains that the generated output be a prefix of a legal resolvent. For example, given assumptions (1) $(-1 + 2)$ (-2), the legal resolvents are (2) (generated by resolving (1) and $(-1 + 2)$), and (-1) (generated by resolving $(-1 + 2)$ and (-2)). So, the LLM is constrained[7] to generating a (next, and then either a 2 or a $-$, and then if it had generated $(2$, it must generate a) next, and if it had generated $(-$, it must generate a 1 next, etc.

So, it is possible in theory to do logically constrained decoding for two small, but non-trivial problems. But is it efficient enough to improve the accuracy of real LLMs? That is a question that must be answered empirically.

4 Empirical Results

We now evaluate how well logically constrained decoding works on our two example world models. Specifically, we explore (1) Is it easily implementable in practice on real LLMs? (2) Does it improve the quality of LLM outputs on these problems in practice? and (3) What is the impact on the LLMs' token throughput?

To answer the first question, we implemented logically constrained decoding for these two problems on several different LLM families: Qwen2.5 7B / 14B / 32B (Qwen et al., 2025), Llama 3.1 8B (Grattafiori et al., 2024), Gemma3 7B / 14B / 27B (Team et al., 2025), Phi4-mini (Microsoft et al., 2025), and Ministral-8B (Jiang et al., 2024). We selected these models because they are open-source, and fit on our computing infrastructure. (Our experiments were run on a shared cluster, using Dell EMC C4140 GPU compute nodes, with 8GB RAM per core, and NVIDIA Tesla V100 GPUs with 16GB or 32GB.) All models are instruction fine-tuned. To account for numerical instability, Gemma3 models were run with FP32, while others in FP16. 8-bit quantization was used for all Gemma3 models and Qwen2.5 32B. Code was in Python, using the HuggingFace Transformers Library (Wolf et al., 2020). Overall, we encountered no particular difficulties in implementing logically constrained decoding with these LLMs. Our code is available on GitHub[8].

We evaluate the effect on LLM output quality in §4.1 for chess, and §4.2 for resolution proofs. In §4.3, we report on the effect of logically constrained decoding on LLM performance (token throughput).

4.1 Chess Results

We first investigate how much improvement logically constrained decoding provides to LLMs to avoid illegal moves. Anecdotally, LLMs play openings well, but gradually perform worse as the game progresses. As we saw in Figure 1, even frontier models eventually attempt moves that violate the rules of chess. Thus, as our figure of merit, we look at the number of legal moves an LLM can make before it makes an illegal move. At the syntax level, we ask models to output moves in Standard Algebraic Notation (SAN). While several other notation systems exist (i.e. Long Algebraic Notation or Portable Game Notation), we consider a move valid only if it is written in SAN. More importantly, though, we check whether each move is a valid move according to the rules of chess, for the current board configuration.

To create a consistent opponent, we play LLMs against the Stockfish chess engine. We downloaded the latest 17.1 version and performed experiments across 5 difficulty settings, with a depth of 15 plies. At lower difficulties, Stockfish chooses moves more randomly.

Because of the randomness in both Stockfish and the LLMs, we play 20 games (10 as White, 10 as Black) for each model, with and without logically constrained decoding. When it is Stockfish's turn, it plays (with some randomness, depending on the difficulty) what it believes to be the best move. When it is the LLM's turn, we allow it to generate a maximum of 10 tokens, which is longer

[7] As noted earlier, the symbolic constraint engine can be designed to allow the LLM some unconstrained thinking tokens, and only constrain specific parts of the output.

[8] https://github.com/terwo/logically-constrained-decoding

Reason	Games by Type Unconstrained	Constrained	Total
Illegal Move	897	0	897
Checkmated	3	898	901
Draw	0	2	2
Total	**900**	**900**	**1800**

Table 1: Chess Game Outcomes Across All Experiments. For each of the Unconstrained condition (the original LLM) and the Constrained condition (the LLM with logically constrained decoding), we played 9 LLMs against 5 different levels of Stockfish for 20 games each, for a total of 900 games. Without the benefit of logically constrained decoding, the LLMs attempted illegal moves in 897 out of 900 games; in the other 3, the LLM lost before it could make an illegal move. With logically constrained decoding, the LLMs never made illegal moves in any game. Draws were due to the five-fold repetition rule.

than any possible SAN move in any position. For the Constrained condition, we allow the start of a legal move to have leading whitespace, and the end to have trailing whitespace, but do not allow whitespace between partial continuations of them. For example, "␣Nf4" is allowed but "N␣f4" is not. Each completed set of moves (one from both White and Black) is then formatted into the prompt for the LLM's next move. Full prompts are detailed in Appendix A.

Detailed experimental results are shown in Figure 2. On average, the unconstrained models generated roughly 5 moves before making an illegal one, whereas the logically constrained models played legal moves for much longer, until the natural end of the game. Table 1 summarizes the reasons that each game ended, over all experiments. Clearly, logically constrained decoding makes a dramatic difference in LLM correctness for chess.

4.2 Resolution Proof Results

These experiments are to assess the improvement that logically constrained decoding provides to LLMs, to prevent the generation of incorrect proof steps. Unlike in chess, there is no value in generating a longer proof — a proof is either correct or not. Therefore, we count the number of correctly generated proofs, over repeated trials, as our figure of merit.

Similar to the variety of chess notations, there are various syntaxes to represent a clause in propositional logic. We chose the notation common in Boolean SAT solving and in electrical engineering, where the plus sign $+$ symbolizes a logical OR.

As the challenge problem for the proofs, we chose the well-known Pigeonhole Problem: proving that it's impossible to place $n + 1$ pigeons in n pigeonholes if no two pigeons can share a pigeonhole. These are hard proofs: the propositional encoding for this problem has $\Theta(n^2)$ variables and $\Theta(n^3)$ clauses, and the resolution proof has an exponential lower-bound in size. We generate problem instances that encode the pigeonhole problem for 1, 2, and 3 holes. Our specific SAT encoding is shown in Appendix B.

For 1 and 2 holes, we run each model 50 times for each constraint condition (with and without logically constrained decoding). For 3 holes, due to limited time and computing resources, we could not complete the full number of trials — more details below. Similar to our implementation for playing chess, LLMs were permitted to output tokens that correspond to brackets, literals, or plus signs with leading or trailing whitespace. The LLMs are also allowed to output reasoning steps in separate lines that start with a double backslash `//`, and output the clauses for the proof on new lines. The prompts used are in Appendix A.

Discussing the results in increasing order of pigeonhole size:

1 Pigeonhole, 2 Pigeons: Many unconstrained models were able to successfully generate resolution proofs for the pigeonhole problem with only 1 hole and 2 pigeons. With the benefit of logically constrained decoding, the success rate goes to 100%. A correct resolution proof for this encoding is very short (only 2 resolvents before the empty clause), so we limited the output generation to 100 tokens. Figure 3 shows the accuracy for both constraint conditions across all models for 50 iterations.

2 Pigeonholes, 3 Pigeons: With 2 pigeonholes, the proof becomes much harder. Across 50 iterations, no Unconstrained models were able to successfully complete the resolution proof. In contrast, with logically constrained decoding, every model successfully generated a correct proof 100% of the time. Figure 4 depicts these results graphically. We limited the output generation to 1000 tokens. We also attempted this proof informally on some commercial frontier models. (We do not have the resources to do extensive experiments on these models.) ChatGPT-5 completed this proof

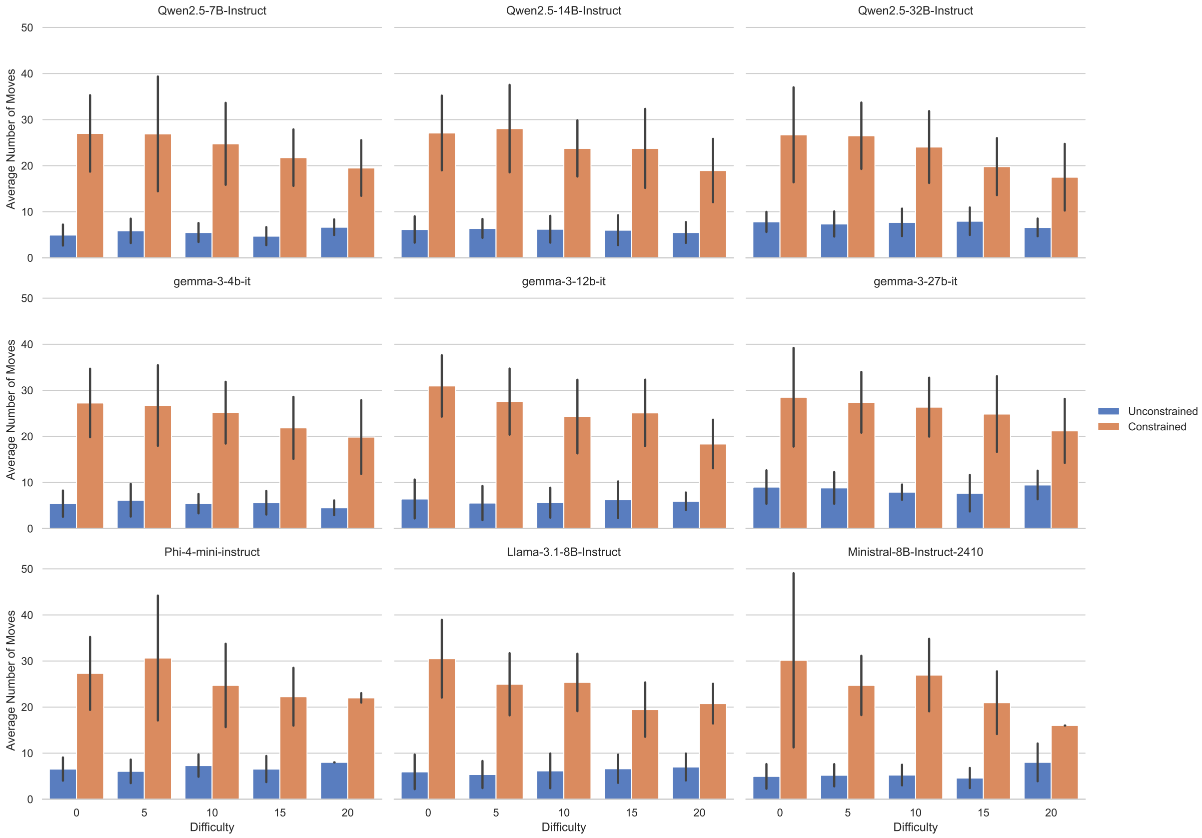

Figure 2: Detailed Experimental Outcomes for Chess Games. There are nine subgraphs here, one for each LLM, labeled above the subgraph. On each subgraph, the x-axis is the Stockfish difficulty level that was the opponent of the LLM. The y-axis is the average number of moves played, over 20 games. (The whiskers show 1 standard deviation.) For each experimental condition, the blue bar on the left is for the original, unconstrained LLM, and the orange bar on the right is with logically constrained decoding. In the unconstrained condition, the LLMs make illegal moves very soon after starting to play. With logically constrained decoding, the LLMs never make illegal moves, so last long enough to eventually lose, losing faster to the stronger Stockfish difficulty settings.

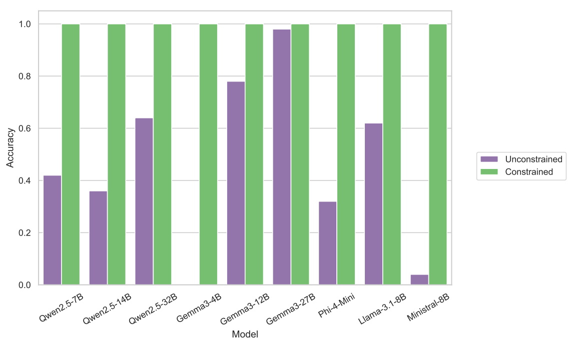

Figure 3: Results for Pigeonhole Proofs of Size 1. The y-axis is the fraction of proof attempts that were correct (out of 50 attempts). The x-axis has a pair of bars for each LLM. In each pair, the blue bar on the left is the success rate for the original, unconstrained LLM; the orange bar on the right is the success rate with logically constrained decoding. A missing bar indicates 0% correct proofs. These small LLMs are able to generate correct resolution proofs in many cases, but this improves to 100% with logically constrained decoding.

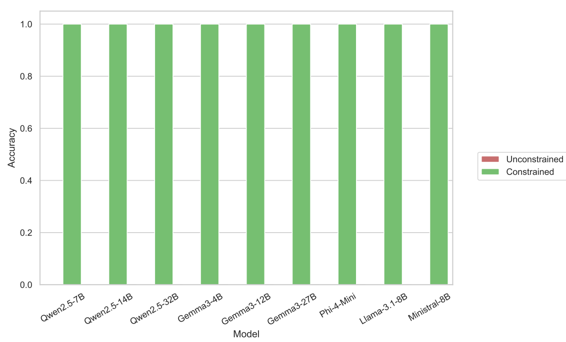

Figure 4: Results for Pigeonhole Proofs of Size 2. This graph has the same interpretation as Figure 3. However, all the blue bars are missing, because no unconstrained LLM was able to complete this proof correctly. The constrained LLM successfully completes the resolution proof on all attempts.

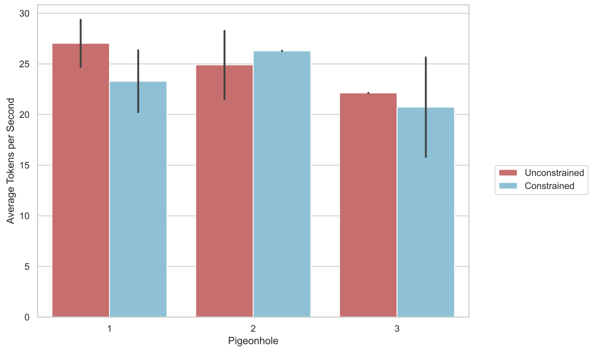

Figure 5: The average tokens per second generated by Qwen2.5-7B across both Unconstrained and Constrained conditions. The logically constrained decoding for pigeonhole instances of size 3 is optimized by enforcing the LLM to only choose possible resolvents of minimal length.

successfully, but Claude Sonnet 4 did not.

3 Pigeonholes, 4 Pigeons: Due to limited time and computing resources, we reduced the number of trials from 50 to 20 per experimental condition. None of the unconstrained models were able to successfully complete this resolution proof. We limited the output generation to 3000 tokens.

For the logically constrained LLMs, we were not able to complete the experiments. This is a long and hard proof, and as the number of clauses in the proof grew, the number of resolvents became unmanageably large. However, if we relax the goal of minimal invasiveness, we can exploit the logical structure of clauses to improve efficiency: specifically, if a partially generated clause is already a legal resolvent, it is pointless to allow the clause to grow any longer, as that only makes the clause weaker. Accordingly, we can modify the constraint engine to force the LLM not to generate a needlessly long resolvent. With this optimization, even the small Qwen2.5 7B model manages to complete the proof correctly. In contrast, as described in the introduction, ChatGPT-5 tries to make an unsound deduction in its proof attempt.

4.3 Effect on LLM Performance

As a measure of the effect of logically constrained decoding on LLM throughput, we measured the average number of tokens per second across all resolution proofs generated by Qwen2.5-7B in Figure 5. The optimization mentioned above is applied to the pigeonhole instances of size 3. The latency overhead is generally minimal in our experiments.

On the downside, our symbolic constraint engine for the resolution proofs doesn't scale well as the proof length grows, because it is trying to generate all possible resolvents. For example, the shortest proof generated by Qwen2.5-7B on the 3-pigeonhole proofs was 126 clauses long, and on this shorter proof, the throughput was 24.7 tokens per second. In contrast, the longest proof took 627 clauses, which slowed the throughput down to 5.5 tokens per second. This motivates our future work, to explore more efficient proof systems that avoid this blow-up in the number of resolvents.

5 Conclusion and Future Work

We have introduced *logically constrained decoding*, which lifts the concept of constrained decoding beyond syntactic constraints to enforcing logically correct output with respect to an underlying, formal world model. We have demonstrated proof-of-concept implementations for chess and for propositional resolution proofs, on nine different LLMs. Our technique guarantees that the small LLMs do not generate illegal outputs for the problems being solved, and enables them to generate correct outputs on problems that even state-of-the-art, proprietary frontier models solve incorrectly.

The main line for future work is to expand the applicability of our technique to additional domains, such as code generation or richer proof systems (e.g., Lean). The principle challenge is how to turn the logical constraints into something that can be enforced efficiently at the token level. For code generation, we are excited by the work of Mündler et al. (Mündler et al., 2025) on constrained generation of type-safe programs. For mathematical proof, we are currently developing a more efficient technique for a more powerful proof system than resolution.

Limitations

In empirical research on LLMs, there is always the risk of unexpected behavior due to minor variations in prompts. For example, there are many notations for Boolean OR in common use that might have appeared in training data, so an LLM might behave differently if we had prompted it to use $\vee$ or even `\lor`, instead of $+$. We have not explored varying the prompts, but we do not expect that our results would change materially. Our prompts are disclosed in Appendix A.

We can modify and perform experiments only on open-source language models, so it is unclear to what degree our results can be applied to proprietary, frontier models. Similarly, even with open-source models, we were limited by our available computing resources to using smaller models. We believe these are sufficient to demonstrate the promise of our approach, but more extensive experiments would be valuable.

In our prompts, we suggest limits to how the LLM can "think aloud" in its answers. This is purely to simplify our implementation, so that our symbolic reasoning engine can easily ignore the unstructured portions of the LLM output. Banerjee et al. 2025 show that strictly constraining LLM outputs can reduce the LLM's reasoning ability, but this can be restored by allowing the LLM to generate unconstrained output with only clearly delimited parts subject to the constraints. Our chess experiments did not allow arbitrary unconstrained "thinking" outputs, so the LLMs likely did not play as well as they might have. Nevertheless, we were evaluating LLMs only on whether moves were legal or not. Our resolution proof experiments did allow the LLMs to generate unconstrained outputs within comments, along the lines suggested by Banerjee et al.

Our current implementation for resolution does not scale to larger proofs. Even so, we are able to generate correct proofs with smaller LLMs for problems that befuddle large, frontier LLMs. As noted above, we are working on a much more efficient proof system, which should scale better.

Acknowledgements

This research was funded through an Undergraduate Summer Research Award and a Discovery Grant, both from the Natural Sciences and Engineering Research Council of Canada (NSERC). This research was also supported through the computational resources and services provided by Advanced Research Computing at the University of British Columbia.

References

Marie Amalric and Stanislas Dehaene. 2016. Origins of the brain networks for advanced mathematics in expert mathematicians. *Proceedings of the National Academy of Sciences*, 113(18):4909–4917.

Debangshu Banerjee, Tarun Suresh, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. 2025. CRANE: Reasoning with constrained LLM generation. In *42nd International Conference on Machine Learning*.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2023. Prompting is programming: A query language for large language models. *Proc. ACM Program. Lang.*, 7(PLDI).

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2024. Guiding LLMs the right way: Fast, non-invasive constrained generation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Nathan Ensmenger. 2012. Is chess the drosophila of artificial intelligence? a social history of an algorithm. *Social Studies of Science*, 42(1):5–30.

Siavash Golkar, Mariel Pettee, Michael Eickenberg, Alberto Bietti, Miles Cranmer, Geraud Krawezik, Francois Lanusse, Michael McCabe, Ruben Ohana, Liam Parker, Bruno Régaldo-Saint Blancard, Tiberiu Tesileanu, Kyunghyun Cho, and Shirley Ho. 2024. xval: A continuous numerical tokenization for scientific language models. *Preprint*, arXiv:2310.02989.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas

Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng

Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Sil Hamilton and David Mimno. 2025. Lost in space: Finding the right tokens for structured output. *Preprint*, arXiv:2502.14969.

Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. 2018. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437.

Alexey Ignatiev, Zi Li Tan, and Christos Karamanos. 2024. Towards universally accessible SAT technology. In *SAT*, pages 4:1–4:11.

Albert Jiang, Alexandre Abou Chahine, Alexandre Sablayrolles, Alexis Tacnet, Alodie Boissonnet, Alok Kothari, Amélie Héliou, Andy Lo, Anna Peronnin, Antoine Meunier, Antoine Roux, Antonin Faure, Aritra Paul, Arthur Darcet, Arthur Mensch, Audrey Herblin-Stoop, Augustin Garreau, Austin Birky, Avinash Sooriyarachchi, Baptiste Rozière, Barry Conklin, Bastien Bouillon, Blanche Savary de Beauregard, Carole Rambaud, Caroline Feldman, Charles de Freminville, Charline Mauro, Chih-Kuan Yeh, Chris Bamford, Clement Auguy, Corentin Heintz, Cyriaque Dubois, Devendra Singh Chaplot, Diego Las Casas, Diogo Costa, Eléonore Arcelin, Emma Bou Hanna, Etienne Metzger, Fanny Olivier Autran, Francois Lesage, Garance Gourdel, Gaspard Blanchet, Gaspard Donada Vidal, Gianna Maria Lengyel, Guillaume Bour, Guillaume Lample, Gustave Denis, Harizo Rajaona, Himanshu Jaju, Ian Mack, Ian Mathew, Jean-Malo Delignon, Jeremy Facchetti, Jessica Chudnovsky, Joachim Studnia, Justus Murke, Kartik Khandelwal, Kenneth Chiu, Kevin Riera, Leonard Blier, Leonard Suslian, Leonardo Deschaseaux, Louis Martin, Louis Ternon, Lucile Saulnier, Lélio Renard Lavaud, Sophia Yang, Margaret Jennings, Marie Pellat, Marie Torelli, Marjorie Janiewicz, Mathis Felardos, Maxime Darrin, Michael Hoff, Mickaël Seznec, Misha Jessel Kenyon, Nayef Derwiche, Nicolas Carmont Zaragoza, Nicolas Faurie, Nicolas Moreau, Nicolas Schuhl, Nikhil Raghuraman, Niklas Muhs, Olivier de Garrigues, Patricia Rozé, Patricia Wang, Patrick von Platen, Paul Jacob, Pauline Buche, Pavankumar Reddy Muddireddy, Perry Savas, Pierre Stock, Pravesh Agrawal, Renaud de Peretti, Romain Sauvestre, Romain Sinthe, Roman Soletskyi, Sagar Vaze, Sandeep Subramanian, Saurabh Garg, Soham Ghosh, Sylvain Regnier, Szymon Antoniak, Teven Le Scao, Theophile Gervet, Thibault Schueller, Thibaut Lavril, Thomas Wang, Timothée Lacroix, Valeriia Nemychnikova, Wendy Shang, William El Sayed, and William Marshall. 2024. Ministral 8b.

Thang Luong and Edward Lockhart. 2025. Advanced version of Gemini with Deep Think officially achieves gold-medal standard at the International Mathematical Olympiad. `https://deepmind.google/discover/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/`. [Online; accessed 2025-Aug-25].

Microsoft, :, Abdelrahman Abouelenin, Atabak Ashfaq, Adam Atkinson, Hany Awadalla, Nguyen Bach, Jianmin Bao, Alon Benhaim, Martin Cai, Vishrav Chaudhary, Congcong Chen, Dong Chen, Dongdong Chen, Junkun Chen, Weizhu Chen, Yen-Chun Chen, Yi ling Chen, Qi Dai, Xiyang Dai, Ruchao Fan, Mei Gao, Min Gao, Amit Garg, Abhishek Goswami, Junheng Hao, Amr Hendy, Yuxuan Hu, Xin Jin, Mahmoud Khademi, Dongwoo Kim, Young Jin Kim, Gina Lee, Jinyu Li, Yunsheng Li, Chen Liang, Xihui Lin, Zeqi Lin, Mengchen Liu, Yang Liu, Gilsinia Lopez, Chong Luo, Piyush Madan, Vadim Mazalov, Arindam Mitra, Ali Mousavi, Anh Nguyen, Jing Pan, Daniel Perez-Becker, Jacob Platin, Thomas Portet, Kai Qiu, Bo Ren, Liliang Ren, Sambuddha Roy, Ning Shang, Yelong Shen, Saksham Singhal, Subhojit Som, Xia Song, Tetyana Sych, Praneetha Vaddamanu, Shuohang Wang, Yiming Wang, Zhenghao Wang, Haibin Wu, Haoran Xu, Weijian Xu, Yifan Yang, Ziyi Yang, Donghan Yu, Ishmam Zabir, Jianwen Zhang, Li Lyna Zhang, Yunan Zhang, and Xiren Zhou. 2025. Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-loras. *Preprint*, arXiv:2503.01743.

Niels Mündler, Jingxuan He, Hao Wang, Koushik Sen, Dawn Song, and Martin Vechev. 2025. Type-constrained code generation with language models. *Proceedings of the ACM on Programming Languages*, 9(PLDI):601–626.

Leyan Pan, Vijay Ganesh, Jacob Abernethy, Chris Esposo, and Wenke Lee. 2025. Can transformers reason logically? a study in SAT solving. In *42nd International Conference on Machine Learning (ICML)*.

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, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Tord Romstad, Marco Costalba, Joona Kiiski, and Stockfish Community. 2008–present. Stockfish: Strong open-source chess engine. `https://stockfishchess.org/`. [Online; accessed 2025-Aug-25].

Herbert A. Simon and William G. Chase. 1973. Skill in chess. *American Scientist*, 61:394–403.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. 2024. Syncode: Llm generation with grammar augmentation. *Preprint*, arXiv:2403.01632.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. 2025. Reinforcement learning for reasoning in large language models with one training example. *Preprint*, arXiv:2504.20571.

Alexander Wei. 2025. I'm excited to share that our latest @OpenAI experimental reasoning LLM has achieved a longstanding grand challenge in AI: gold medal-level performance on the world's most prestigious math competition—the International Math Olympiad (IMO). `https://x.com/alexwei_/status/1946477742855532918`. [Online; accessed 2025-Aug-25].

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

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

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing. *Preprint*, arXiv:1910.03771.

A Full Prompts

The prompts used for experiments are listed below. There is no whitespace after the colon in all prompts.

Chess Prompts

White's Perspective

You are a chess grandmaster.
You are playing chess as White in the starting position. You are playing against a strong opponent.
Output your next, competitive move in Standard Algebraic Notation (SAN). Do not include move numbers.

White:

Black's Perspective

You are a chess grandmaster.
You are playing chess as Black after White has made their first move. You are playing against a strong opponent.
Output your next, competitive move in Standard Algebraic Notation (SAN). Do not include move numbers.

White:
`{stockfish_chosen_move}`
Black:

Figure 6: Prompts for Playing Chess Against Stockfish. If playing as Black, the first prompt will include the first move chosen by Stockfish.

Unconstrained Resolution Prompt

Generate an unsatisfiability proof for the given clause database using only resolution steps.
Rules for Clauses:
1. Each clause must start with '(' and end with ')'.
- Integers must be separated by '+' with optional spaces around it.
- Negated literals have a leading '-'
- Example: (1 + -3 + 4) 2.
Each derived clause must be valid with respect to the original database and all previously generated clauses.
- A clause C is valid if it is the resolvent of two existing clauses in the current set.
- The two parent clauses must share exactly one pair of complementary literals.
- The resolvent is formed by taking all literals from both parents except the complementary pair, with duplicate literals removed.
3. Do not repeat any clauses already in the database or previously generated.

Output Format:
- Each line is either:
a) A comment line starting with '//' followed by reasoning, OR
b) A clause line in parentheses only, with no extra text.
- No introductions, no summaries, no prose outside of comment lines.
- First non-comment line must be a clause.
- The proof must end exactly with the empty clause ().
Example:
Clause database: (1 + 2) (1 + -2) (-1 + 2) (-1 + -2) (1 + 2 + 3)
Proof:
// Resolving (1 + 2) and (-1 + 2) on literals 1 and -1 gives (2)
(2)
// Resolving (1 + -2) and (-1 + -2) on literals 1 and -1 gives (-2)
(-2)
// Resolving (2) and (-2) gives the empty clause ()
()

Now, generate an unsatisfiability proof for the following:
Clause database: `{clause_database}`
Proof:

Figure 7: The Prompt to Generate Resolution Proofs with Unconstrained Decoding

Constrained Resolution Prompt

Generate an unsatisfiability proof for the given clause database using only resolution steps.
Rules for Clauses:
1. Each clause must start with '(' and end with ')'.
- Integers must be separated by '+' with optional spaces around it.
- Negated literals have a leading '-'
- Example: (1 + -3 + 4) 2.
Each derived clause must be valid with respect to the original database and all previously generated clauses.
- A clause C is valid if it is the resolvent of two existing clauses in the current set.
- The two parent clauses must share exactly one pair of complementary literals.
- The resolvent is formed by taking all literals from both parents except the complementary pair, with duplicate literals removed.
3. Do not repeat any clauses already in the database or previously generated.

Output Format:
- Each line is either:
a) A comment line starting with '//' followed by reasoning, OR
b) A clause line in parentheses only, with no extra text.
- No introductions, no summaries, no prose outside of comment lines.
- First non-comment line must be a clause.
- The proof must end exactly with the empty clause ().
Example:
Clause database: (1 + 2) (1 + -2) (-1 + 2) (-1 + -2) (1 + 2 + 3)
Proof: (2) (-2) ()

Now, generate an unsatisfiability proof for the following:
Clause database: `{clause_database}`
Proof:

Figure 8: The Prompt to Generate Resolution Proofs with Logically Constrained Decoding

B Pigeonhole Encoding

We encode our pigeonhole principle instances with n holes and $k = n + 1$ pigeons: for each pigeon $i \in \{1, ..., k\}$ and hole $j \in \{1, ..., n\}$, we introduce a new variable variable x_{ij}, which is True if pigeon i is in hole j.

The CNF formula has two types of clauses:

1. Pigeon clauses
   For each pigeon i, it must be in some hole
   $$(x_{i1} + x_{i2} + ... + x_{in}) \text{ for each } i \in \{1, ..., k\}$$
2. Hole clauses
   For each hole j, for every pair of distinct pigeons $i \neq i'$, both must not be in the same hole
   $$(-x_{ij} + -x_{i'j}) \text{ for each } j \in \{1, ..., n\}, 1 \leq i < i' \leq k$$

Since there are $n + 1$ pigeons yet only n holes, the formula is unsatisfiable, which can be proven with resolution.

We store these formulas in the DIMACS format. We load these formulas with the PySAT package (Ignatiev et al., 2018, 2024). An example of our encoding for a pigeonhole instance of size 2 is as follows:

```
p cnf 6 9
1 2  0
3 4  0
5 6  0
-1 -3 0
-1 -5 0
-3 -5 0
-2 -4 0
-2 -6 0
-4 -6 0
```

C Tokenization

We only allow one consistent method of representing answers to each given problem (e.g., SAN for chess, using plus signs OR in resolution proofs). In our implementation, we track the generation state, and condition the valid next tokens on the current state. For example, in the resolution proof example, the logically constrained model can only generate literals after outputting a left paranthesis or plus sign.

Therefore, we ensure that no tokenizer would split legal strings across states. After investigating how each model family would tokenize text that represent legal moves / valid clauses under our specified syntax, we did not find cases where strings that are composed of multiple states would be represented as one token (e.g., "+␣3" could be tokenized separately as "+" and "␣3" but not fully as one token). Many tokenizers also represent larger numbers by splitting them up digit-by-digit (Golkar et al., 2024), and we account for this in the state transitions for SAT solving by allowing the generation state to enter a "partial literal" state.

D Sampling Parameters

We use all defaults provided by the HuggingFace Transformers library. We explicitly set temperature = 1.0, but otherwise defer to the model's default configuration (e.g., top-k, top-k, etc.).

Pigeonhole Size (n)	**Max Token Limit**
1	100
2	1000
3	3000

Table 2: Maximum Token Limits Allocated Depending on the Pigeonhole Size